Information integration among Heterogeneous and Autonomous Applications

Benabdelkader, A.

Publication date
2002

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Appendix A

Application of Database and Middleware Standards in GFI$_2$S

The GFI$_2$S federated architecture, as conceived and described in Chapter 6, presents an open and flexible solution towards a generic approach for information exchange and data integration. GFI$_2$S uses object-oriented standards and middleware solutions to the extent possible, and suggests their extension when standards are not available.

- From the database development perspective, object-oriented database standards are used in GFI$_2$S to support the portability of database schemas across conforming ODBMSs, to exchange objects between applications, and to provide database documentation.

- From users, applications, and database accesses perspectives, the GFI$_2$S system targets a comprehensive solution, based on Web standards and middleware solutions to facilitate the access to the heterogeneous and autonomous data sources.

This appendix describes the use of object-oriented and Web standards to facilitate the exchange of information among different applications and databases. Namely, two aspects are emphasized. First, full and rich representation of the schema concepts, query language, and data representation are supported through the object-oriented technologies and standards e.g. ODL, OQL, and OIF, as described in subsections of section A.1. Second, user facilities and data transparency are supported through Web standards and Middleware technologies e.g. ODBC, Java, and XML, as described in subsections of section A.2. Some of the problems that face the standardization process are also addressed and discussed within these sections.

A.1 Object-Oriented Standards and Extensions Adaptation for GFI$_2$S

This section illustrates the benefits gained when deploying database standards for the management of information and data integration. It also addresses extensions to these standards to better support the integration/interoperation process for the schema modeling, the query
Object-Oriented standards are defined and specified to better support the portability of database schema and database objects across conforming ODBMSs. Portability in object database management systems would allow an application program that is written to access one ODBMS to be able to access another ODBMS, as long as both ODBMSs support the ODMG standard faithfully.

The various components of the ODMG specification, from which the integration (interoperation) process can benefit, include [CBB+00]:

- An Object Model, which gives database capabilities including definition of relationships, extents, collection classes, and concurrency control.

- An Object Definition Language (ODL), which allows defining a database schema in a programming-language in terms of object types, attributes, relationships, and operations.

- An Object Query Language (OQL), which includes support for object sets and structures and supports object identity, complex objects, path expressions, operation invocation, and inheritance.

- Language Bindings to Java, C++, and Smalltalk, which extends the respective language standards to allow the storage of persistent objects: each binding includes support for OQL, navigation, and transactions.

Currently, ODMG is the only standard interface that allows to store Java objects directly using a standard API that is completely database independent, indifferently of the underlying storage mechanism, being a relational or an object database. If the database system has an interface that conforms with ODMG, it can store objects directly using the standard Java API.

In conjunction to object-oriented databases, object-relational DBMSs emerged as a way of enhancing the capabilities of relational DBMSs with some of the features that appeared in object-DBMSs. These features include wide variety of data types, complex objects, and audio/video data streams. Examples of object-relational DBMSs, which are emerging nowadays, include Informix Universal Server, Oracle, Matisse, and DB2 Universal Server.

The extension of database standards can also be adapted, to properly support the mapping constructs and the derivation operations within a federation of heterogeneous databases. The following sub-sections describe in more details the application of the ODMG standard to schema integration and systems interoperation.

### A.1.1 Object Definition Language – ODL

The object definition language (ODL) provides the semantic power for schema definition and offers a standard, which can be extended to support the mechanism of schema integration in the area of federated databases. The extensions to the ODL are expected to address the definition of export and integrated schemas, and the provision of a set of operations serving the need for mapping and derivation. Some research work started a decade ago considering the extensions of ODMG-ODL standard, to also support the schema integration and its evolution. The research is also advancing in the direction of federated schema in ODMG.
[BFN 94], extending the ODMG for federated databases [Rad 96, KJR 98], and extending ODL for object-oriented views integration [RKB 01].

Extension of ODL by researchers and developers in the area of federated databases provides a powerful mechanism, semantically rich, for integrating schemas from multiple heterogeneous data sources, and to define a virtual database that can be queried just like any stand-alone/centralized database. Extended ODL can be used for export and integrated schema specification from two perspectives. It can be successfully used for: (a) export and integrated schemas definition, and (b) the mapping specifications for the derived schemas, namely operations for the data translation e.g. join, select, and intersect. In GFI_2S, Export and Integrated schemas definition comply to the ODL syntax, with only the exception that an export schema is a subset of a local schema, while the integrated schema is a subset of the union of the local schema with a number of imported schemas. Similarly, the derivation operations, for export and integrated schemas, are based on a schema derivation language, which extends the ODL syntax and benefits from other derivation languages developed to better support the information integration among heterogeneous and distributed applications. More details regarding the federated derivation primitives and some examples are provided in section 6.2.2.3.

Hereafter, we present a simple data modeling example that will be used within the next sub-section. In this example from the scientific domain, an Experiment is defined by its: domain, results, date, and it is performed by a (Scientist). the scientist is in turn, defined by his name, (working) field, and the Experiments that he/she performs. In a relational model, this can be represented as follow:

Experiment (Exp_ID, Sc_ID, Domain, Results, Date)
Scientist (Sc_ID, Exp_ID, Name, Field)

Using the ODMG-ODL language, the complete definition for this simple scientific application is presented as follow:

Interface Experiment:
  attribute String Domain;
  attribute String Results;
  attribute Date Date;
  relationship SET<Scientist> PerformedBy
    Inverse Scientist::Performs;
}

Interface Scientist:
  attribute String Name;
  attribute String Field;
  relationship SET<Scientist> Performs Inverse
    Experiment::PerformedBy;
}

The links between the two entities in the relational model are expressed through the notion of foreign keys (e.g. Exp.ID and Sc.ID). While, in object-oriented model, links are explicitly expressed through the relationship concept (e.g Performs and PerformedBy).
A.1.2 Query Languages – SQL, SQL3, and OQL

Query languages are considered to facilitate the interaction with the database and permits the management and maintenance of its instances. Within the area of database query languages, SQL is continuing its evolution towards a new standard called SQL3, which is extended to deal simultaneously with tables from the relational model and classes from the object model. SQL3 language extends SQL standard by incorporating object-oriented capabilities and features such as complex data types, user-defined routines, inheritance, and indexing extensions.

Similarly to SQL3, the Object Query Language (OQL), enhances the data types, predicates, relational operations, triggers, user-defined types, transaction capabilities, user-defined routines, and extends the SQL language to include object-oriented capabilities. Its syntax for queries is similar to the syntax of the relational standard query language SQL, with some additional features for object model concepts, such as object identity, complex objects, inheritance, polymorphism, path expression, and cross-reference relationships.

To illustrate an example of these extensions to the SQL standard language, let us consider the example from the scientific domain presented at the end of the previous section. An SQL query, for instance, requesting experiment results performed by a scientist named ‘John’ before ‘May 12, 2001’, involves condition predicates upon the two entities Experiment and Scientist. Using the relational model, the query request can be formulated as follow:

```
SELECT Results FROM Experiment WHERE Date <= '12-05-2001' AND Sc_ID IN
  (SELECT Sc_ID FROM Scientist WHERE Name = "John"
  )
```

While using an extended Matisse-SQL, which is based on the object model, a query for the same request, is formulated as follow:

```
SELECT Results FROM Experiment WHERE
  Date <= '12-05-2001' AND PerformedBy.Scientist.Name = "John"
```

SQL Queries in object-oriented/object-relational DBMSs are formulated slightly different than in a standard SQL. Matisse-SQL, for instance, is enhanced to deal with the challenging applications of today by incorporating the object-relational concepts, which mainly concern object identity, inheritance, encapsulation, path expression, and support for multiple data types and complex objects.

Thus, when it comes to extending the SQL standard, different DBMS developers use their specific extension mechanisms, which differ from one DBMS to another. To overcome this issue, there must be certain kind of consensus (or standardization) among DBMS developers concerning these extensions. At least a common agreement regarding the common extensions such as: object identifier, inheritance, path expression, and cross-relationship references.

Hereafter, we illustrate some of the object-oriented/object-relational extensions to the SQL language. These examples are demonstrated using the Matisse DBMSs, and address the concepts of object identifiers, inheritance, cross-reference relationships, and path expression mentioned above.
Matisse DBMS uses the keyword OID within a select query to retrieve the object identifier of the database objects: and uses the keyword ONLY within a select query, to only retrieve the direct instances of a given class. As such, in the example below, the query “select only * from scientist” will not retrieve instances that are subclasses of Scientist. The third example below illustrates the usage of relationships and path expression concepts.

\[
\text{OID} : \text{Select OID. } * \text{ from Experiment} \\
\text{Inheritance} : \text{select only } * \text{ from Scientist} \\
\text{Relationship & Path expression} : \text{select } * \text{ from Experiment where PerformedBy.Scientist.Name= 'John'}
\]

Among the factors that have made SQL a successful standard is its simplicity, especially for non-expert database users. OQL lacks this feature, especially in some cases when querying complex data models involving relationships and complex data types (structures, lists, arrays, blobs, etc.). The manner, in which, queries are defined is very hard to understand, even by users that are quite familiar with database terminology.

Within OQL, we believe that the language must be strongly based on the use of object identifiers, cross-relationship references, and path expression. The following example, for instance, retrieves the Experiments Performed By a Scientist with the OID='0x124'.

\[
\text{SELECT * FROM Experiment E} \\
\text{WHERE E.PerformedBy.Scientist.OID = '0x124'}
\]

### A.1.3 Object Interchange Format - OIF

Cattel et al. [CBB+00] defines the object interchange format (OIF) as a specification language used to dump and load the current state of an ODBMS to/from one or more files. Therefore, the OIF format can be used to exchange persistent objects between ODBMSs, seed data, provide documentation, and derive test suites. Since OIF allows the specification of persistent classes and their states, it can also be considered as a facility for information exchange between database systems and among heterogeneous applications.

In comparison to the XML standard, on certain aspects, OIF provides a similar powerful facility for data representation and information exchange between databases and applications. However, two major and distinct issues separate these two standards from each other. The first issue concerns the format for the data representation, where the OIF format only preserve the types, attributes, and relationship identifiers, as provided by the ODL definition of the ODBMS schema. While, the XML format uses additional naming tags for data representation, which requires extra effort for their handling. The second issue, relates to the consideration and the use of these standards. Even considering the first issue, which seems in favor of the OIF format, the XML standard is far more better and widely used by users and groups in several application domains, than the OIF standard that still has not had the chance for strong considerations by these groups and developers.

During the development of some projects related to the MegaStore framework, presented in Chapter 4, and also related to the Virtual Laboratory project, presented in Chapter 5; an OIF facility is developed serving the requirements of information exchange between databases and applications. The development of such a tool is motivated by the need for adequate facility for archives and backups, for information exchange between different applications, and for preserving the data consistency: specially, to facilitate the projects...
in which new versions of DBMSs are released or when a different operating system and platform are to be considered.

The OIF tool developed for these projects (called Mt.Oif), is specific only for Matisse ODBMS, supports the database back-ups and recovery, and provides a facility for data exchange between different versions of Matisse ODBMSs with full support for different platforms (e.g. Sun Solaris, Linux, and Windows). All these tasks are performed through the single Mt.Oif set of code. Following is the syntax of the Mt.Oif tool:

```
Mt.Oif DB@host [user psswd] in|out [oif file]
Usage: Mt.Oif DB@host [user psswd] in|out [oif file]
DB : Database Name
Host : Host Name
user : User Name
psswd : User Password
in : load an oif file
out : generate an oif file
File : File Name that contains data in OIF format to be loaded/generated
```

The following example command Dumps the data from the database Sc.Experiment, which runs on amelie.wins.uva.nl, and generates a backup file, named Sc.Exp.Backup.oif.

```
Mt.Oif Sc.Experiment@amelie.wins.uva.nl out Sc.Exp.Backup.oif
```

The loading of the generated OIF file into another database called Sc.Exp.Bk, which runs on the other host carol.wins.uva.nl, passes through the “in” parameter as follow:

```
Mt.Oif Sc.Exp.Bk@carol.wins.uva.nl in Sc.Exp.Backup.oif
```

The Mt.Oif tool handles all the rich and complete data types supported by Matisse ODBMS, which include among other types: Date and TimeStamps, lists and arrays, large and binary objects, and audio and video streams. As such, the audio and video streams are treated within the OIF format as lists of elements, similar to the way they are handled by the Matisse DBMS itself.

### A.2 Web Standard and Middleware Adaptation for GFI₂S

In addition to ODMG, usage of standards like Java, CORBA, and XML can provide interoperability and portability to applications involving databases with different database models and systems. This section addresses a number of emerging information technologies from the perspective of how they could support higher levels of interoperability. Among the emerging technologies, which are relevant to interoperability and more related to the subject of this chapter, we cover in this section:

- Object Database Connectivity ODBC,
- Multi-platform applications development using Java,
- XML standard for information exchange
A.2. Web Standard and Middleware Adaptation for GFI\(_2\)S

More standards relevant to interoperability are further emerging nowadays for information handling technologies, among which, we enumerate CORBA\(^1\), Jini\(^2\), DCOM\(^3\) (Distributed Component Object Model), SOAP\(^4\) (Simple Object Access Protocol), WAP\(^5\) (Wireless Application Protocol), and High-Portability Programming Languages. These technologies are however not addressed within this dissertation, due to space limitation. Readers interested in investigating these specific technologies in depth can refer to [TB 00, WAP 00, Vin 97].

A.2.1 Object Database Connectivity - ODBC

Providing common interfaces to heterogeneous databases has been a challenging issue in the domain of data access and information retrieval through standards. Open Database Connectivity (ODBC) provides a certain level of standard access using SQL as a standard language for interaction with the underlying database. Figure A.1 illustrates the data access mechanism using ODBC standard, which offers an open facility that provides a common set of API calls to manipulate databases. Using ODBC, users are able to develop a single application program that can access different DBMSs. This mechanism allows developers to build and distribute a client-server type of application without being restricted to a specific DBMS.

![Application Access to Remote Database via ODBC](image)

In addition, ODBC is considered as a good interface for supplying data. A summary of advantages for using ODBC over native database APIs include:

- Providing an open standard, which is already supported by many development groups and organizations in research, industry, and academia.
- Enabling access, from a common set of code, to data from different relational and object-oriented database systems (e.g. Oracle, Matisse, Sybase, and MySQL).

Although ODBC allows developers to build and distribute client-server applications without targeting a specific DBMS, it has many limitations when used as a programming interface. For instance, consider the case that in many cases, different DBMS vendors support the common basic functionalities defined by standard ANSI SQL 92. However, the various

---

\(^1\)Common Object Request Broker Architecture [http://www.omg.org/]
\(^2\)Jini\(^\text{TM}\) Connection Technology Executive Overview [http://www.sun.com/jini/overview/]
\(^3\)DCOM - Distributed Component Object Model [http://www.microsoft.com/com/tech/DCOM.asp]
\(^5\)WAP Forum Specifications [http://www.wapforum.org/what/technical.htm]
extensions and added-values to different database systems has forced DBMS developers in this area to extend the SQL standard to support their added value features and new built-in data types. This is the case, especially with the appearance of object relational and object-oriented databases, in which the extensions mainly concern object identifiers, abstraction, inheritance, and cross reference relationships. The database system developers therefore find themselves forced to support the new challenging features also via the extensions of the ODBC, which is by origin relational. It is at this level where the differences between the different ODBC drivers become more important and this is one of the reasons, which explains some differences between different ODBC drivers, each specific for a database management system.

Still the use the ODBC standard is of a significant importance, since even if there are differences between the different DBMSs when addressing the object-oriented/object-relational extensions, these differences are minor and does only require minor changes in the development codes. In other words, changing the application’s program to access a different database will only require slight changes, which concern the object-oriented/object-relational extension. Therefore, applications can be easily ported from one database to another by switching the ODBC driver and making the required changes instead of rewriting the entire application.

A.2.2 Use of JAVA for Application Programming

Java is the most promising computer language to increase the ability to share software easily across heterogeneous applications of different types. With the appearance of the Internet and the Web, Java fulfills its potential as an object-oriented programming language suitable for supporting networked environment, and becomes the de facto standard programming language for Web applications.

The Java object-oriented technology helps in research and development as an important software language for implementing interoperable information management systems, in particular, when used, with its associated Internet technologies (e.g. J2EE\(^6\), JDBC\(^7\), and EJB\(^8\)), in web-based applications. It gives users the flexibility and dynamism, and brings a good potential for making legacy systems appear in a more Internet-friendly language. Java applets, for instance, can act as front-ends for accessing the capabilities of remote legacy system, where Internet users can access those legacy systems without having to create mapping code, and without caring about knowledge of the special access mechanisms that are hidden from them. Therefore, the obvious benefit for Java applications developers is that the combination of application and database programming into a single environment means that developers only have to deal with one data model.

There are emerging types of applications, largely driven by Java and the Web, where direct object storage, whether to a relational or an object database, is a clearly superior solution. Java fully supports this feature through the ODMG standard and best suites in environments that need to provide connectivity to a variety of DBMS servers and heterogeneous databases and that require significantly high level of concurrently connected users, where performance and scalability are required.

---

\(^6\)J2EE: Java 2 platform, Enterprise Edition
\(^7\)JDBC is a trademark name, often though to stand for Java Database Connectivity
\(^8\)EJB: Enterprise Java Beans
A.2.3 Use of XML for Information Exchange

Very rapidly, since its ratification by the World Wide Web Consortium (W3C) in 1998, XML is already becoming the *de facto* standard for data communication and information exchange among distributed organizations and applications. One of the main applications that benefits from XML is thus, the information exchange and data integration among heterogeneous databases. Similar to ODMG and Java standards, the advantages of using XML standard is to reduce the number of wrappers serving the interoperation among heterogeneous and distributed databases and applications.

Regarding the integration approach of GFI<sub>2</sub>S, presented in this chapter, its architecture can be augmented with the XML standard for data exchange representation. In such architecture, the data transformation to XML format will be performed at different sites of the federation, where the local XML wrapper processes and transforms the data from a specific representation to an XML notation. Meanwhile, the merging of data is a global process, to be performed at the federated layer, and then organizing the data to fit the integrated schema that is defined at the federated layer. This approach allows the unification of data transformation process at the federated level, where a universal module can transform and merge the returned sub-results simply, by using the XML data and the mapping specifications defined for it.

Integrating XML data across applications and databases is of great interest for the database community; efficient techniques for integrating XML data across local- and wide-area networks are an important research focus. However, the consideration of XML in databases and in particular for information integration, raises several challenges for database research. Having XML focusing only on the *syntax* for data representation only partially advances the prospects of their integration. To support information integration at the *semantic* level however, there must be a certain type of agreement among all involved database nodes upon specific DTDs in XML. For instance, if the XML data does not conform to a fixed schema; names and meanings of the used tags are arbitrary and the data is self-describing in XML documents. According to Alon Levy [AL 99], there are several issues that need to be considered to enable such integration. Among these issues we mention a few important ones; some of these issues are already being addressed by current research.

- **Languages for describing the contents and capabilities of XML sources**, which provide the semantic mapping between the data in the source and the relations in the mediated schema. The main challenges involve (1) the restructured data appearing in XML is richer than in relational data, (2) scaling up to a very large number of XML sources must be supported, and (3) exploiting the knowledge conveyed by accompanying DTDs is required.

- **Query reformulation algorithms**, which require the development of algorithms for efficiently reformulating user queries, posed on a mediated schema, to queries that refer to different underlying XML data sources. The known techniques for reformulation from the relational case do not extend easily to languages for querying XML documents.

- **Translation among DTDs**, which provides the proper tools and facilities to translate XML data conforming to one DTD into an XML document conforming to a different DTD, presumably with semantically related content.

- **Obtaining source descriptions**, which develops methods for automatically (semi-automatically) computing source descriptions for newly introduced XML data sources; becomes significant when the number of data sources grows.
Some challenging research work on using XML for information exchange is reported in [SFP 00]. Some advances are also evolving in the areas of information management and data integration [AL 99. MMA 99. BF 01], views definition [Abt 99], scientific data archiving [PWD+99], and graphical querying languages [SFP+99]. Among the developed systems, which deploy the XML syntax and the XML query engine: the Tukwila data integration system [IHW 01] can be mentioned, which is designed specifically for processing network-bound XML data sources, and Tox [BBM+01]: a repository for XML data and metadata, which supports real and virtual XML documents.