A Metadata Management Scheme for Middleware-based Multidatabase Management *

J. Tan
Monash University, Australia
jtan@csse.monash.edu.au

A. Zaslavsky
Monash University, Australia
azaslavs@csse.monash.edu.au

C. A. Ewald
Dept. of the Premier and Cabinet, Queensland, Australia
catherine.ewald@premiers.qld.gov.au

A. Bond
Distributed Systems Technology Centre
bond@dstc.edu.au

Abstract

Federated or multidatabase systems, built up from multiple autonomous database systems, require a basic level of interoperability among the component database systems. They must also provide a reasonable degree of functionality similar to that of modern database management systems, e.g., queries. Many years of research by other groups have gone into this area, but not many of the proposed solutions have been commercially implemented due to inherent complexities and relatively low probability of success. This paper discusses our interoperability solution that supports ad-hoc federations on a framework that adheres to technology standards. We hope to attract interest in ad-hoc federations which would be more responsive than rigid solutions. Through a standards-based solution, we hope to present an alternative approach that is practicable to implement. Our prototypes involve two key standards from the Object Management Group: the Common Object Request Broker Architecture (CORBA) and the Meta Object Facility (MOF). We also use the Extensible Markup Language (XML) for database metadata expression and interchange. We will show that our approach and framework are feasible based on the results of a prototyping effort.

1 Introduction

Many existing solutions for heterogeneous information integration are considered inadequate in providing viable results. Many are restrictive, or perhaps are limited in functionality [19, 22].

Older solutions in the literature combine the *schemas* of component databases to form a composite or integrated schema. A database *schema* refers to a model that describes the database structure and contents [3]. *Schema integration* is the process of combining the schemas of participating databases to form an integrated schema [1]. The resulting composite is used as the basis for a virtual system to access multiple databases, often referred to as a *multidatabase system* [13, 24, 30]. There are variants:

1. *Global schema systems* rely on one integrated *global schema*, which combines the schemas of all component databases. However, the complexity in resolving conflicts increases with the number of schemas involved, usually making the global schema approach undesirable [5, 15, 19, 22].

2. *Federated database systems* use several *federated* or *import schemas* instead, each constructed from the *export schemas* of some of the component databases [30]. Database administrators would define *export schemas* over the portions of their database that they wish to share. This gives them control over how much of their data is available to global users. Import schemas are defined based on the specific needs of users. Since import schemas need not cover all the component databases, the

federated approach requires a simpler process compared to the global schema approach.

3. *Multidatabase language systems* provide access through an access language for flexibility. On the other hand, users are expected to deal with the complexity involved in formulating how to access the component databases and how to combine the results [22, 23].

A logical concern is that database schemas should be expressed using a data model that captures adequate database semantics. This is crucial in determining how the databases can mesh with one another. Solving conflicts in database semantics is regarded as the most difficult hurdle in the interoperability of heterogeneous databases [3, 9, 23], because semantics are not as easy to represent and manipulate as schemas.

Multidatabase systems, data warehouses and other systems where heterogeneous databases are made to cooperate require some form of mapping between the global system and the component databases. This involves *metadata*, i.e., data about data, to describe databases. Even in multidatabase language systems, it is necessary to handle and associate metadata from participating databases when access commands are invoked. Metadata support is needed in the resolution of semantic issues since metadata provides a means of understanding and manipulating the databases.

We are focusing on two things that pertain to the interoperability of heterogeneous databases:

- how to enable *ad-hoc* federations of databases, and
- what kind of metadata support is necessary to provide ad-hoc features.

Our solution therefore centers around metadata management and the necessary logic to manipulate metadata for ad-hoc federations of databases. We advocate a hybrid approach involving some language facilities and no global schema. Our ad-hoc federations take the form of global queries or global views over multiple, autonomous data sources which are probably heterogeneous. We use the terms "federation", "federated databases" and "federated system" interchangeably to refer to the interoperable system of data sources that need not encompass all component data sources. We use the term "data sources" interchangeably with "databases" because we foresee that the source of data need not be a traditional database managed by a DBMS. It should also be possible to involve semi-structured or even unstructured data sources, but issues pertaining to this are not discussed in this paper.

Whereas ad-hoc queries are generally associated with purely language-based solutions, ours is not. Rather than give users the burden of figuring out the access and integration logic, we address most of those issues through *mediators* [16, 17, 33]. Our *active meta objects* exhibit *autonomous* behavior, i.e., their actions are not limited to built-in programming [26], and perform the tasks of mediation. We do not promise full automation of the federation process, since this has been proven by others to be impossible. We believe, however, that providing semi-automated ad-hoc federation is a reasonable alternative. By "ad-hoc" we pertain to an aspect of responsiveness to federation requests, such that either accurate results are gathered, or the system can competently negotiate for a reasonable compromise. However, our discussion on the ad-hoc behavior will be limited, as this paper focuses on the metadata facility instead.

2 Our Tools

It is important to us that we use standards-based tools, because adherence to standards encourage consistency in an organization's development efforts over time, increasing interoperability with legacy and future components. The Object Management Group (OMG) is a leading organization that advocates object management standards for application development. The World Wide Web Consortium (W3C) is the established body that develops World Wide Web standards to ensure that the Web realizes its full potential and to promote interoperability.

OMG standards are embodied within the Object Management Architecture (OMA). Within the OMA, the Common Object Request Broker Architecture (CORBA) [34, 31], the OMG's middleware standard, provides interoperable communication via Object Request Brokers (ORBs). This allows distributed objects to communicate, and allows these objects to be implemented on different programming and operating system platforms. Using CORBA also allows us to make use of existing CORBA technologies and OMG standards such as the Meta Object Facility (MOF) [18] and the Common Warehouse Metamodel (CWM) [7]. The W3C Extensible Markup Language (XML) was developed to be the universal format for structured documents on the Web. It provides product-neutral interchange of data and encourages sharing [4, 20, 32].

We also rely on JDBC (Java Database Connectivity), a Java API for uniform remote access to databases [11]. CORBA is essentially platform independent, so JDBC is not the only option. However, JDBC is the best to use when working with

LEVEL	CONTENTS
M3: the MOF model	defines metamodels
M2: metamodels, meta-meta information	meta-types, meta-relations and meta-schemas
M1: meta-information	types, type relations and type schemas
M0: information	entities and relationships, grouped by domain

Table 1. Levels of the MOF abstract model.

Java, which is the implementation platform required by the current version of the MOF product suite we are using, dMOF 1.0.1.

2.1 The OMG MOF

CORBA provides the middleware backbone in the OMA reference model, and the Meta Object Facility (MOF) provides the repository for *meta-information*, i.e., information about information [6]. The abstract model of the MOF has four levels, shown in Table 1 and explained below:

1. At level M0, the universe of information for a particular domain consists of entities and the relationships they may have with one another. This is similar to concepts in the Entity-Relationship (ER) data model, which is commonly used for conceptual database design.

2. At the M1 level, entities are classified into *types*, which may be involved in *type relations* with other types and may define *type schemas* using some *type language* and *type system*. This is similar to the concept of entity types, relationship types and database schemas in the ER data model.

3. At the M2 level, types are classified into *meta-types*, which may be involved in *meta-relations* and, using a *meta-type system*, define *meta-schemas*. We also refer to this level as the *meta-meta* level where we define *metamodels*, which describe how models are described. For instance, the OMG CWM specifies metamodel packages for database schema interchange, e.g., the ER data model [7]. These metamodels can be used to associate concepts between data models, e.g., entity types in the ER data model are implemented as relations in the Relational data model.

4. At level M3 is the MOF model that defines how to define metamodels.

Metadata management with the MOF begins with the metamodel defined at the M2 level of the MOF abstract model. A repository is then generated to handle metadata based on this metamodel. The metadata tools are then written to manage metadata, i.e., define, store and manipulate metadata.

The dMOF product suite from the Distributed Systems and Technology Centre (DSTC) implements most of the features in the MOF specification. dMOF relies on metamodels defined using the Meta Object Definition Language (MODL) [6, 7]. Using the MODL2MOF compiler, a metamodel repository is generated that stores the metamodel. Meta object persistence is provided when a database backend is coupled with dMOF. The compiler also generates the repository object that manages the repository. This object is referred to as the *moflet*.

Factory objects are generated along with the repository to instantiate metadata packages [8]. Of course, since MOF is a CORBA service, everything that dMOF generates is in IDL, before being implemented using some implementation platform. The version we are using relies on Borland Visibroker for Java to handle the translation from IDL to Java.

A relevant aspect of the MOF standard is that MOF metadata may be implemented [8]. Advantages are obvious in having actual data (M0 level) work with metadata (M1 level). Many issues of *schematic discrepancy* [10, 14, 27, 29] can be resolved if data and metadata can exist on the same plane. A good example of schematic discrepancy between two databases is when an object that is data on one database is implemented as metadata, e.g., an attribute, on another.

The MOF specification also allows meta objects to be programmed with behavior. This is a key feature in MOF, as will become clear when active meta objects are explained in Section 3.1.

2.2 XML

While XML is not a human-readable format [2], it is a computer-readable format for structured data. XML documents include definitions of the data structure. This enables software data interchange, assuming that there is agreement on the metadata definitions. *Document type definitions* or DTDs are the predominant means of specifying conventions of XML documents. A better means eventually became necessary for more complex types of data, such as databases. So the W3C came up with the *XML schema*

Listing 1. Simple DTD for the relational database model

```
<!ELEMENT rdb (tables,comment)>
  <!ATTLIST rdb
        name CDATA #REQUIRED
        ConnectProtocol CDATA #REQUIRED
        URL CDATA #REQUIRED
        UserName CDATA #REQUIRED
        PassWord CDATA #REQUIRED>
<!ELEMENT tables (table+)>
<!ELEMENT table (columns, keys)>
  <!ATTLIST table
        name CDATA #REQUIRED
        TableType
          (base|query|view) "base"
        UpdateType (ro|rw) "ro" >
<!ELEMENT columns (column+)>
<!ELEMENT column (#PCDATA)>
  <!ATTLIST column
      data_type (char|string|boolean|
      short|long|float|fixed|date) "string"
      unique (true|false) "false"
      not_null (true|false) "false" >
<!ELEMENT keys (column+)>
<!ELEMENT comment (#PCDATA)>
```

format, which is still under development. However, it is expected to become a full standard for defining how XML data involving complex structures are specified.

Listing 1 shows a DTD for a simple relational database. This can also be expressed using the XML Schema format, but we did not include the XML schema version of this DTD to conserve space. Listing 2 shows a portion of the relational schema of a database DLSUDB in XML format, which conforms to the DTD in Listing 1.

Assuming that several software systems conform to the same DTD or XML schema, i.e., they have an understanding of the same metadata, they can exchange relational database data. Note that the DTD shown includes information about how to connect to the databases. Assuming that they are accessible on the network, it is possible to develop software that can access them without explicitly writing separate code for each database. The metadata about these databases can instead be referred to prior to data access.

XML is an established standard. It has been in popular use over the Web for quite a while, and has also gained ground in enterprise applications. A number of DBMSs support the XML format for data interchange. XML also teams up with the MOF in the XML Metadata Interchange (XMI) standard [32].

2.3 Why Not UML?

One may argue that the Unified Modeling Language (UML) is a better choice than MODL for defining metamodels. It can give users a visual appreciation of metamodels and database schemas. However, dMOF does not currently support UML input. XML parsers such as the Java XML API, JAXP, are more readily available. Furthermore, since XML is an excellent interchange format in the first place, there is no reason why XML data cannot be rendered in visual notations for frontend tools.

Our examples will also show that our metamodels are expressed first in UML before being hand-coded in MODL. On the other hand, we prefer using ER diagrams to express our database conceptual schemas, primarily out of habit.

3 The Metadata Facility

Several approaches attack the problems of interoperability at the metadata level not only for databases but also in enterprise modeling and data warehousing [7, 9, 12, 21, 25, 28]. We developed a database metadata facility on top of CORBA using the MOF. We started with metamodels. We are currently using a relational metamodel and others can be put in place easily, e.g., an object-oriented metamodel. All metamodels are handled by one metamodel repository manager, the moflet, which uses persistent storage to store metamodel definitions. All metadata, derived from the metamodels, are managed by and embodied within meta objects. We then embedded active behavior in our meta objects and give them the roles of mediators [16, 17, 33]. Through our behavioral design, we hope to provide a feature-rich system of metadata management. However, the focus of this paper is on the metadata management we have developed, so we will not be discussing the behavior behind ad-hoc federations in detail.

3.1 Active Meta Objects

Figure 1 shows a conceptual framework for the development of ad-hoc federations. The overall solution we are developing consists of the following key features:

1. We provide a metadata facility that allows the instantiation of *active meta objects*, which can persistently store metadata. There are two types

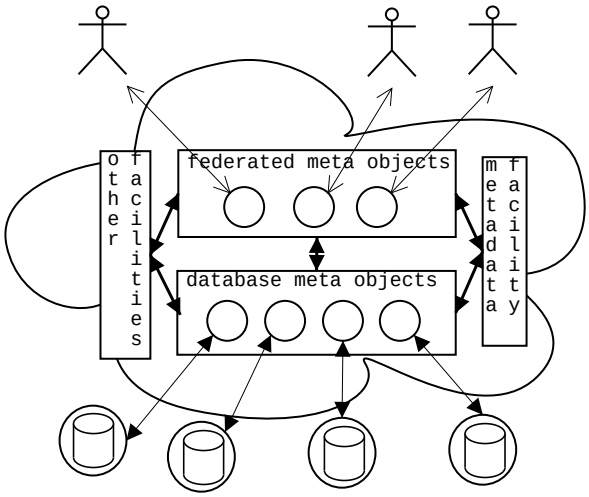

Figure 1. A Conceptual Framework.

of active meta objects: *database meta objects* and *federation meta objects.*

2. We "attach" a database meta object to one database. The primary purpose of the database meta object is to assimilate as much of the associated database's metadata as possible. This will make the database meta object a point of access to the database.

3. Federation meta objects embody federations, which may be in the form of global queries or views. Such meta objects will actively seek information from database meta objects and use the metadata gathered to realize the federation that they were created for.

4. Ad-hoc federations are realized through the autonomous behavior of federated meta objects to respond to an invocation. While the basic protocol for ad-hoc response is programmed in, the actions taken will be based on current circumstances. The ideal response is to produce the correct results, e.g., of a query. If accuracy is not possible, the federation meta object will indicate this and negotiate for additional time and guarantee a proportional increase in accuracy. At the worst case, sufficient time for human intervention will be negotiated for. It is also possible for the federation client to require an immediate response with the indicated low accuracy rating of the results, of course.

Figure 1 also shows that other facilities can use the metadata facility, e.g., the CORBA Trader Service.

3.2 Database Metadata

Database metadata is handled by a database meta object. An example of the XML format of database metadata was shown earlier in Listing 2. Database meta objects will assimilate such metadata in one of two ways:

- Either the database meta object will autonomously explore the database associated with it, and assimilate all gathered metadata, or
- in case the associated database doesn't have sufficient functionality, e.g., it cannot respond to metadata queries, then the database administrator may have to "spoon-feed" the metadata in XML format to the database meta object.

3.3 Accessing the Data Source

Whether or not the database system can respond to metadata queries from database meta objects, the access or connection details must be known from the start. Otherwise, there is no way to reach the database. The connection details may take several forms:

- As in our metadata sample in Listing 3, we can use a reference to a CORBA object by which queries can be invoked as method calls.
- The database administrator may create views in the backend for sharing data. The views can then be queried, or stored queries may be accessed, via some direct protocol such as JDBC.
- Data can be invoked via server-side features on some website, e.g., Java servlets. This has the advantage of shielding the database from direct access.

Listing 3. XML global query definitions over DLSUDB and CSSEDB

```
<?xml version='1.0'?>
<!DOCTYPE rqset SYSTEM "rquery.dtd">
<rqset name="rqset1"
  ConnectProtocol="CORBA"
  URL="IOR:0000000000000003149444c3a6d6f662
  746f72792f53696d706c655265706f7369746f72
  00000000058617469635 04f41002020200000000"
  UserName="" PassWord="">
<results>
<resultset name="people()"
  definition=
```

```
  "(SELECT uid,uname,dept,'CSSE' AS org
    FROM CSSEDB.user ) UNION
   (SELECT sid AS uid, sname AS uname,
    faculty AS dept, 'DLSU' AS org
    FROM DLSUDB.staff)">
<columns>
  <column data_type="short" unique="true"
   not_null="true">uid</column>
  <column data_type="short" unique="false"
   not_null="true"> uname</column>
  <column data_type="string" unique="false"
   not_null="true"> dept</column>
  <column data_type="string" unique="false"
   not_null="true"> org</column>
</columns>
</resultset>
<resultset name="team()"
 definition=
 "(SELECT grpnum,grpname, NULL AS rank
   FROM cssedb) UNION
   SELECT gnum AS grpnum, gname AS grpname,
   rank FROM dlsudb)">
<columns>
  <column data_type="short" unique="true"
   not_null="true">grpnum</column>
  <column data_type="string" unique="false"
   not_null="true">grpname</column>
  <column data_type="short" unique="false"
   not_null="false">rank</column>
</columns>
</resultset>
</results>
</rqset>
```

We are currently using direct access via JDBC as it is the most convenient and flexible. However, this may present serious security compromises, unless security mechanisms are in place.

3.4 Federation

How is federation carried out? Since we already have the metadata and connection details for the data sources, the stage is set. The composed *federation schema*, e.g., a global query or view definition, will define the federation. This must be coupled by the implementation of the methods that correspond to specific result sets from the federated databases.

We wish to integrate some information from two databases. Each database was described in XML as in Listing 2. Each database is represented by a database meta object. We define two global queries against them in Listing 3, `people()` and `teams()`. We limited it to two queries to conserve paper space. Note that both queries are named as CORBA method calls. This means that the CORBA object identified by the reference in the global query definition must already exist and had to be written by someone who is authorized to choose how and which part of the data sources may be queried. The specific queries are invoked using the two method calls named above. The XML code defines the SQL-like definitions for each query as well as the metadata describing the contents of the query results.

How is the composed schema as in Listing 3 generated and used? We start with the instantiation of a new federation meta object. It must then become aware of the federation for which it was created. We can manually write the federation schema as in Listing 3 then "feed" it in as input to the federation meta object. This is how we did it in our initial prototype. A better option is to enable the processing of *imprecise federation schemas* from which the precise federation schema can be inferred semi-automatically. This may involve some interaction with the user but this is usually preferable compared to writing the precise query definitions by hand. This is a non-trivial feature but there has been reasonable success for some projects described in literature, e.g., concepts of *imprecise queries* [3] or *query discovery* [17]. In our system of active meta objects, the *federation discovery* process will involve interaction among federation meta objects and database meta objects. Two things are of importance here:

- The database meta objects must have absorbed sufficient metadata from the databases associated with them. This will make it possible for the federation meta object to explore the metadata and assimilate them for the sake of the federation.
- The federation meta object must have the ability to learn from the database meta objects, analyze the imprecise federation definitions and infer the corresponding precise definitions. Successful inferences will generate the federation schema.

Compromise scenarios arise when the accuracy and completeness of database metadata is insufficient. This is where ad-hoc behavior becomes necessary to provide a reasonable response to the federation invocation, despite the possibility that the accuracy of the results will not be ideal. This fact will, of course, be made known to the client, i.e., the user, who invoked it.

The processing of the queries or views, i.e., the federations, must be implemented at the data source backend. Several ways of providing data access have been discussed in Section 3.3.

How are conflicts and dissimilarities handled? Many concepts must be successfully employed that already

exist in the literature and can be adopted. Some are basic transformation mechanisms to resolve data model differences, scale differences and data type differences. Other solutions exist to resolve schematic discrepancy. There are many with varying degrees of automation using ontologies and other aids that resolve semantic issues one way or another [3, 17, 23]. Inventing our own solutions to address these conflicts is unnecessary and beyond the scope of our project. Those solutions already proven by others can mostly be adopted in our flexible and portable framework.

4 Conclusion

This paper presented the metadata facility that we developed to support the solution we are working on for the ad-hoc federation of multiple data sources across a network. We illustrated how this is made possible using MOF-based metadata management and active meta objects. We explained how the database meta objects extract metadata from data sources and thereby act on behalf of data sources in ad-hoc federations. We also explained how a federated system is generated via the interaction of federation meta objects and database meta objects. Creating active meta objects is almost trivial in our approach as they are instantiated as needed via MOF factory objects. Using the MOF as the foundation of metadata handling, and XML as the interchange and metadata format, makes it possible to handle database metadata and information integration in a more convenient manner.

On the other hand, there are several issues left unresolved in the project. We are still in the process of prototyping the learning mechanism behind active meta objects. This is actually beyond the scope of our study, but we can easily adopt existing learning algorithms. The same is true for the ad-hoc protocols that will ensure that a response is always available for any given federation invocation. We also used XML DTDs rather than XML schemas, whereas XML schemas are better-suited for database schemas than DTDs, although the standard has not been finalized. We also handled input directly via XML documents, rather than use a graphical frontend for our metadata definition, model-browsing and articulation [5], which is the ideal scenario.

The next step is to prototype active meta objects with sophisticated, if adopted, learning algorithms. Our experiments have already proven that they can extract database metadata via JDBC, and store the metadata. We must then also program behavior to handle inaccuracy in federation results.

Acknowledgment. The work reported in this paper has been funded in part by the Co-operative Research Centre Programme through the Australian Government's Department of the Industry, Science and Resources.

References

[1] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4), Dec. 1986.

[2] B. Bos. XML in 10 points. In http://www.w3c.org/1999/XML-in-10-points, 1999.

[3] M. W. Bright, A. R. Hurson, and S. Pakzad. Automated resolution of semantic heterogeneity in multidatabases. *ACM Trans. on Database Systems*, 19(2):212–253, June 1994.

[4] A. Ceponkus and F. Hoodbhoy. *Applied XML: a toolkit for programmers.* John Wiley & Sons, Inc., 1999.

[5] W. Cheung and C. Hsu. The model-assisted global query system for multiple databases in distributd databases. *ACM Transactions on Information Systems*, 14(4):421–470, 1996.

[6] S. Crawley, S. Davis, J. Indulska, S. McBride, and K. Raymond. Meta-meta is better-better! In *IFIP WG 6.1 Int. Working Conf. on Distributed Applications and interoperable Systems (DAIS'97)*, 1997.

[7] Common Warehouse Metamodel (CWM) specification. OMG Document ad/2000-01-01, February 2000.

[8] *dMOF 1.0 User Guide.* University of Queensland, Australia, 2000.

[9] N. Georgalas. Integrating distributed data over their semantic identity. In *Proc. Association for Information Systems 1997 Americas Conf. on Heterogeneous Database Interoperability & Data Warehousing*, Indianapolis, Indiana US, August 15-17 1997.

[10] L. M. Haas, R. J. Miller, B. Niswonger, M. T. Roth, P. M. Schwarz, and E. L. Wimmers. Transforming heterogeneous data with database middleware: Beyond integration. *IEEE Data Engineering Bulletin*, 22(1):31–36, 1999.

[11] G. Hamilton, R. Cattell, and M. Fisher. *JDBC Database Access with Java.* Addison-Wesley, 1997.

[12] C. Hsu, editor. *Enterprise integration and modeling: the metadatabase approach.* Kluwer Academic Publishers, 1996.

[13] A. R. Hurson, M. W. Bright, and S. H. Pakzad, editors. *Multidatabase Systems: An Advanced Solution for Global Information Sharing.* IEEE Computer Society Press, 1993.

[14] R. Krishnamurthy, W. Litwin, and W. Kent. Interoperability of heterogeneous databases with schematic discrepancies. In *Proc. First Int'l Workshop*

on Interoperability in Multidatabase Systems, pages 144–151, 1991.

[15] W. Litwin and A. Abdellatif. Multidatabase interoperability. *IEEE Computer*, 19(2):10–18, Dec. 1986.

[16] L. Liu, L. L. Yan, and M. Özsu. Interoperability in large-scale distributed information delivery systems. In A. D. et.al, editor, *Workflow Management Systems and Interoperability (NATO Asi Series. Series F, Computer and Systems Sciences, Vol 164)*. Springer Verlag, 1997.

[17] R. Miller, L. M. Haas, and M. A. Hernández. Schema mapping as query discovery. In *Proc. 26th VLDB Conf.*, Cairo, Egypt, 2000.

[18] Meta object facility (MOF) specification v1.3. Object Management Group (OMG), March 2000.

[19] R. Mühlberger and M. E. Orlowska. A business process driven multidatabase integration methodology. In *Proc. of the 6th Int. Workshop on Database Re-engineering and Interoperability*, 1995.

[20] A. Nakhimovsky and T. Myers. *Professional Java XML Programming with Servlets and JSP*. Wrox Press Ltd, 1999.

[21] S. Navathe and A. Savasere. A schema integration facility using object-oriented data model. In O. A. Bukhres and A. K. Elmagarmid, editors, *Object-oriented multidatabase systems: a solution for advanced applications*, chapter 4, pages 105–128. Prentice-Hall, 1996.

[22] S. B. Navathe and M. J. Donahoo. Towards intelligent integration of heterogeneous information sources. In *Proc. 6th Int. Workshop on Database Re-engineering and Interoperability*, 1995.

[23] A. Ouksel and I. Ahmed. Coordinating knowledge elicitation to support context construction in cooperative information systems. In *Proc. First IFCIS Int. Conf. on Cooperative Information Systems (CoopIS'96)*, June 1996.

[24] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, 2nd edition*. Prentice-Hall, 1999.

[25] M. P. Papazoglou, L. Marinos, and N. G. Bourbakis. Distributed heterogeneous information systems and schema integration. In *Proc. Int'l Conf. on Databases, Parallel Archtectures, and Their Applications (PARABASE-90)*, pages 388–397, 1990.

[26] S. Russel and P. Norvig. *Artificial intelligence: a modern approach*. Prentice-Hall, 1995.

[27] F. Saltor, M. G. Castellanos, and M. Garcia-Solaco. Overcoming schematic discrepancies in interoperable databases. In D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, editors, *IFIP DS-5 Semantics of Interoperable Database Systems*, volume 1, pages 184–198, Nov. 1992.

[28] D. G. Schwartz. *Cooperating Heterogeneous Systems*. Kluwer Academic Publishers, 1995.

[29] A. Sheth and V. Kashyap. So far (schematically) yet so near (semantically). In D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, editors, *IFIP DS-5 Semantics of Interoperable Database Systems*, volume 1, pages 282–301, Nov. 1992.

[30] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous and autonomous database systems. *ACM Computing Surveys*, 22(3):183–236, Sept. 1990.

[31] A. Vogel and K. Duddy. *Java programming with CORBA, Second Edition*. John Wiley & Sons, Inc., 1998.

[32] Website: the XML Metadata Interchange (XMI). http://www.omg.org/technology/xml/.

[33] L. L. Yan. Towards efficient and scalable mediation: The AURORA approach. Student paper, CASCON'97, Nov. 1997.

[34] Z. Yang and K. Duddy. CORBA: A platform for distributed object computing (a state-of-the-art report on OMG/CORBA). Published in a link from the Object Management Group (OMG) website. http://www.infosys.tuwien.ac.at/Research/Corba /archive/intro/OSR.ps.gz, 1999.

Listing 2. Partial XML metadata definition for DLSUDB

```
<?xml version='1.0'?>
<!DOCTYPE rdb SYSTEM "relationalDB.dtd">
<!-- Document Contents -->
<rdb name="dlsudb"
  ConnectProtocol="JDBC"
  URL="jdbc:postgresql://localhost/dlsudb"
  UserName="guest"
  PassWord="any">
<tables>
<table name="borrows" TableType="base"
 UpdateType="ro">
<columns>
  <column data_type="short" unique="false"
    not_null="false">sid</column>
  <column data_type="short" unique="false"
    not_null="false">eqnum</column>
  <column data_type="string" unique="false"
    not_null="false">dateborrowed</column>
  <column data_type="string" unique="false"
    not_null="false">datereturned</column>
</columns>
<keys>
  <column>sid</column>
  <column>eqnum</column>
  <column>dateborrowed</column>
</keys>
</table>

<table name="equiplog" TableType="base" UpdateType="ro">
<columns>
  <column data_type="short" unique="false" not_null="false">eqnum</column>
  <column data_type="string" unique="false" not_null="false">eventdate</column>
  <column data_type="string" unique="false" not_null="false">eventtime</column>
  <column data_type="boolean" unique="false" not_null="false">urgent</column>
  <column data_type="string" unique="false" not_null="false">personresponsible</column>
  <column data_type="string" unique="false" not_null="false">dateresolved</column>
  <column data_type="string" unique="false" not_null="false">eventdesc</column>
</columns>
<keys>
  <column>eqnum</column>
  <column>eventdate</column>
  <column>eventtime</column>
</keys>
</table>

<table name="equipment" TableType="base" UpdateType="ro">
<columns>
  <column data_type="short" unique="false" not_null="false">eqnum</column>
  <column data_type="string" unique="false" not_null="false">description</column>
  <column data_type="string" unique="false" not_null="false">vendor</column>
  <column data_type="float" unique="false" not_null="false">cost</column>
  <column data_type="short" unique="false" not_null="false">gnum</column>
</columns>
<keys>
  <column>eqnum</column>
</keys>
</table>

<table name="memberof" TableType="base" UpdateType="ro">
<columns>
  <column data_type="short" unique="false" not_null="false">sid</column>
  <column data_type="short" unique="false" not_null="false">gnum</column>
  <column data_type="short" unique="false" not_null="false">datejoined</column>
</columns>
<keys>
  <column>sid</column>
```