ELEKTRİK, VOL.6, NO.1, 1998, ©TÜBİTAK EMO

An Interoperability Infrastructure for Developing Multidatabase Systems*

Asuman Doğaç, Gökhan Özhan, Ebru Kılıç, Fatma Özcan,
Sena Nural, Sema Mancuhan, Cevdet Dengi, Pınar Köksal,
Uğur Halıcı, Budak Arpınar, Cem Evrendilek, Vahid Sadjadi
Software Research and Development Center
Computer Engineering Department, METU
06531 Ankara-TURKEY
e-mail: asuman at srdc.metu.edu.tr

Abstract

A multidatabase system (MDBS) allows the users to simultaneously access autonomous, heterogeneous databases using a single data model and a query language. This provides for achieving interoperability among heterogeneous, federated DBMSs. In this paper, we describe the interoperability infrastructure of a multidatabase system, namely METU Interoperable DBMS (MIND). The architecture of MIND is based on OMG distributed object management model. It is implemented on top of a CORBA compliant ORB, namely, ObjectBroker. The interface of the generic database object is defined in CORBA IDL and multiple implementations of this interface, one for each component DBMSs, namely, Oracle 7, Sybase, Adabas D and MOOD (METU Object-Oriented Database System) are provided. The main components of MIND which are built on this infrastructure are a global query manager, a global transaction manager, a schema integrator, interfaces to supported database systems and a graphical user interface.

Keywords: multidatabase system, distributed object management, object management architecture, CORBA, interoperability

1. Introduction

In today's enterprises information is typically distributed among various platforms which could be immediately made available for many users through the existing computer networks. Therefore, there is a need to access and share data across these systems. However, database autonomy and heterogeneity still form a severe bottleneck for the development of effective interoperable information systems. Two or more resources are interoperable if they can interact to execute tasks jointly. There is a growing need for tools to maximize the portability, reusability and interoperability of arbitrary computing services while keeping the autonomy of the pre-existing databases in a federated approach. Heterogeneity in underlying systems makes this integration very difficult if not impossible. the heterogeneity exists at three basic levels. The first is the platform level. Database systems reside on different hardware, use different operating systems and communicate with

* This work is partially being supported by the Turkish State Planning Organization, Project Number: AFP-03-12DPT.95K120500, by the Scientific and Technical Research Council of Turkey, Project Number: EEEAG-Yazilim 5, by Motorola (USA) and by Sevgi Holding (Turkey).

other systems using different communications protocols. The second level of heterogeneity is the database management system level. Data is managed by a variety of database management systems based on different data models and languages (e.g. file systems, relational database systems, object-oriented database systems etc.). Finally the third level of heterogeneity is that of semantics. Since different databases have been designed independently semantic conflicts are likely to be present. This includes schema conflicts and data conflicts.

Commercially available technology offers inadequate support both for integrated access to multiple databases and for integrating multiple applications into a comprehensive framework. Some products offer dedicated gateways to other DBMSs. This approach is quite restricted and provides only a partial solution. Some gateways are binary and only interconnect two DBMSs (e.g., Oracle 7's Dedicated Transparent Gateway), others do not provide transactions across multiple DBMSs (e.g., Sybase's OmniSQL Gateway). None of these systems properly deal with semantic or structural heterogeneity of the stored data. Also, they require a complete change of the organizational structure of existing databases to cope with heterogeneity.

Another way of achieving interoperability among heterogeneous databases is through a multidatabase system. A multidatabase system (MDBS) resides unabtrusively on top of existing database systems and presents a single database illusion to its users. In particular, a multidatabase system maintains a single global database schema against which its users issue queries and updates. It is suggested that this is the approach that should be preferred over gateways [1]. The primary objective of a MDB is to significantly enhance productivity in developing and executing applications that require simultaneous access against multiple independent databases.

One restriction of multidatabase systems has been that they are able to provide interoperability only among DBMSs and cannot handle repositories which do not have DBMS capabilities. The introduction of object-oriented technology into data management has lifted this restriction. Object-orientation, with its encapsulation and abstraction capabilities, enable the development of wrappers which encapsulate a particular repository and provide a common DBMS-like interface to the rest of the system. Also, since the systems interoperate in a distributed environment, capabilities and functionality of the distributed object management platform have to be taken into account in designing the architecture of multidatabase systems.

When a Distributed Object Management (DOM) architecture is used as the infrastructure of a multidatabase system, it easily becomes possible to provide interoperability of a multidatabase system with other repositories that do not have DBMS capabilities in a distributed environment. One important emerging distributed object computing platform is Object Management Group's (OMG)[1] Object Management Architecture (OMA). OMA is a family of industry standards that include Common Object request Broker Architecture (CORBA) [2] and Common Object Services Specification (COSS) [3]. The OMA defines a Reference Model identifying and characterizing the components, interfaces, and protocols that compose a distributed object architecture. CORBA is a middleware which enables distributed objects to operate on each other. COSS is a complementary standard for integrating distributed objects. There are currently a number of commercial Object Request Broker's (ORBs) that are CORBA-compliant (e.g., DEC's ObjectBroker, SunSoft's NEO and JOE, IBM's DSOM, HP's ORB Plus and Iona's Orbix) that can be used as an interoperability platform. CORBA and COSS together provide the basic infrastructure that can be used to provide database interoperability.

[1] OMG is a registered trademark, and CORBA, ORB, OMG IDL are trademarks of OMG.

2. MIND Architecture

MIND is a multidatabase system [4, 5, 6, 7] which aims at achieving interoperability among heterogeneous, federated DBMSs. Its architecture is based on OMG's Object Management Architecture (OMA), CORBA and COSS. In MIND, there is a generic Database Object defined in CORBA IDL and there are multiple implementations of this interface, one for each of the local DBMSs, namely Oracle7[2], Sybase[3], Adabas D[4], MOOD (METU Object-Oriented DBMS) [8, 9]. The current implementation makes unified access possible to any combination of these DBMSs through a global query language based on SQL. When a client application issues a global SQL query to access multiple databases, this global query is decomposed into global subqueries and these subqueries are sent to the ORB (CORBA's Object Request Broker) which transfers them to the relevant database servers on the network. On the server site, the global subquery is executed by using the corresponding call level interface routines of the local DBMSs and the result is returned back to the client again by the ORB. The results returned to the client from the related servers are processed if necessary. This approach hides the differences between local databases from the rest of the system. Thus, what the clients of this level see are homogeneous DBMS objects accessible through a common interface.

Figure 1 shows the major components of MIND architecture and their interactions.

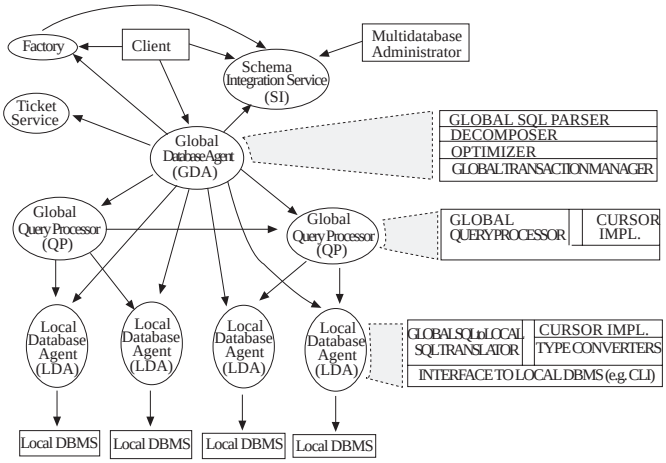

Figure 1. An Overview of MIND Architecture

The basic components of MIND are a Global Database Agent (GDA) class (Object Factory in CORBA terminology) and a Local Database Agent (LDA) class:

1. A LDA class objects are responsible from:

- maintaining export schemas provided by the local DBMSs represented in the canonical data model,
- translating the queries received in the global query language to the local query language,
- providing an interface to the LDBMSs.

2. A GDA class objects are responsible from:

[2] Oracle7 is a trademark of Oracle Corp.
[3] Sybase is a trademark of Sybase Corp.
[4] Adabas D is a trademark of Software AG Corp.

- parsing, decomposing, and optimizing the queries according to the information obtained from the Schema Information Service,

- global transaction management that ensures serializability of multidatabase transactions without violating the autonomy of local databases.

In addition to these, MIND has the following complementary components:

- *Object Factory Server*: This server, namely the FactoryServer is responsible from the creation of other MIND objects such as LDA Object (LDAO), Transaction Manager Object (TMO), Query Manager Object (QMO), and Query Processing Object (QPO). It provides the implementation of DBfactory Interface which has a single method, namely CreateObj. A client who needs an LDAO, TMO, or QPO just calls the CreateObj method of FactoryServer. Once the FactoryServer is started, it creates an object of its own implementation and writes its object reference to the Advertisement Partition of ObjectBroker's Registry.

- *Ticket Server*: This server provides a globally unique, monotonically increasing ticket number each time it receives a request. These unique ticket numbers are used in the transaction management process of MIND. This server is started at the initialization of MIND system and it continues to serve the whole MIND system continuously.

- *Schema Information Service*: This server, namely the SchemaServer provides and manages the global schema information necessary for the decomposition of global queries into subqueries [10]. It holds the implementation of a class called Schema Integration (SI) which contains attributes and methods necessary for the management and manipulation of schema. The integration of export schemas is currently performed by using an object definition language (ODL) which is based on OMG's inteface definition language.

  The multidatabase administrator (DBA) builds the integrated schema as a view over export schemas. MIND has a graphical tool for the integration of export schemas coming from the local DBMSs [11]. The functionalities of ODL allow selection and restructuring of schema elements from existing local schemas. Although SI can be replicated, current implementation of MIND has a single centralized SI. MIND provides its users a common data model and a single global query language based on SQL. Figure 2 provides a simple example of ODL definition which integrates export schemas from two different DBMSs. A simple global query example is also given in Figure 2. SchemaServer is started at the initialization phase of MIND system and stays alive during the life-time of the system. Query Manager is the client of this server.

- *Transaction Manager*: This server is started on demand by the ORB and is responsible from the execution and global serializability of both flat and nested transactions. It uses a technique called Nested Tickets Method for Nested Transactions (NTNT) [12] which we have developed in order to provide the correct execution of flat and nested transactions without violating the local autonomy of the participating DBMSs. The main idea of NTNT technique is to give ticket values of global transactions at all levels, that is, both the parent and the child transactions obtain tickets. Transaction Manager interacts with Ticket Server in order get unique ticket values. Then each global (sub)transaction is forced into conflict with its siblings through its parent's ticket at all related sites. Finally, the ticket values of the transactions are checked in order to provide the global serializability.

Implementation of the Transaction Manager is explained in more detail in [13]. Transaction Manager allows global nested sub-transactions only if the underlying DBMSs that will receive these sub-transactions support nested transactions. Among the DBMSs incorporated to MIND, only Sybase and Adabas D support nested transactions. Therefore the restrictions of a global transaction to Sybase and Adabas D servers can be nested transactions, the others are flat transactions. For example, since Oracle7 does not support nested transactions you cannot start a global transaction in MIND which in turn starts nested transactions on Oracle7 DBMS.

During the life-cycle of a global transaction, Transaction Manager keeps track of the states of each sub-transaction on each LDA object involved. It handles global commit and global abort using two-phase commit protocol (2PC) over LDA objects. LDA objects control submission of operations to the local DBMSs and communicate with Transaction Manager to achieve atomic commitment of global transactions. Note that the LDA objects execute the global sub-queries in parallel. Transaction Manager is infact a part of GDA and is embedded in GDA together with Query Manager.

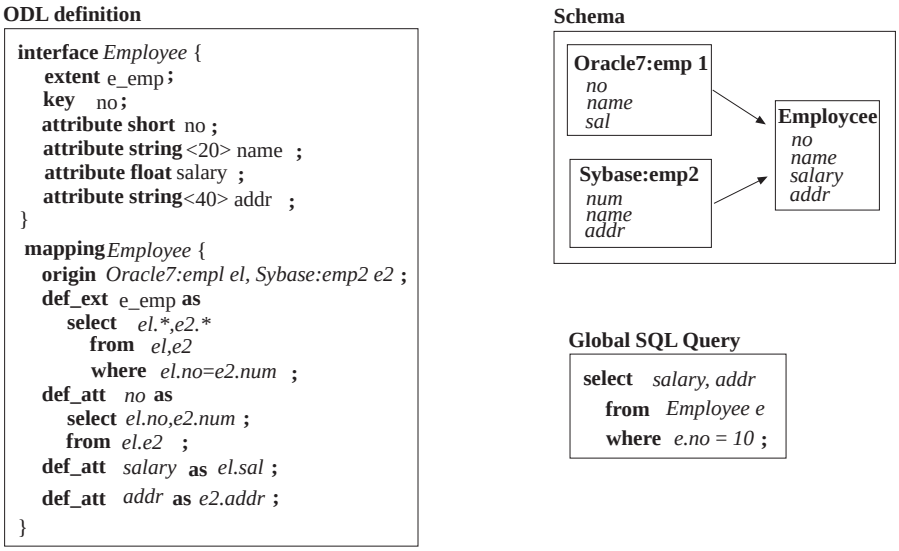

Figure 2. ODL and SQL example.

- *Query Manager*: This server is also started on demand by the ORB and is responsible from the decomposition of global queries into subqueries. It gets the schema information necessary for the query decomposition from the SchemaServer. In fact, Query Manager is also a part of GDA. Query and transaction management processes are combined within GDA. In addition to the decomposition of global queries, Query Manager also performs query optimization [14].

- *Query Processor*: This server is responsible for query processing. It minimizes the total query processing time by enabling parallel execution of subqueries. It performs the necessary operations (such as join, outer-join, and union) for processing partial results coming from the local DBMSs in order to get the final result of the global query. In other words, Query Processor objects perform intersite operations between two partial results that become available at run-time. A Query Processor is started by the ORB as a result of a request from the Query Manager indicating that two partial results to be processed together are ready at the LDAs. Query Processor provides the implementation of QPSrvc Interface which has methods such as ActivateQP and GetResult for query processing.

- *MIND GUI*: MIND system has a graphical user interface (GUI) implemented in the graphical programming environment of Tk/Tcl. In Figure 3. a screen snapshot of MIND graphical user interface is provided.

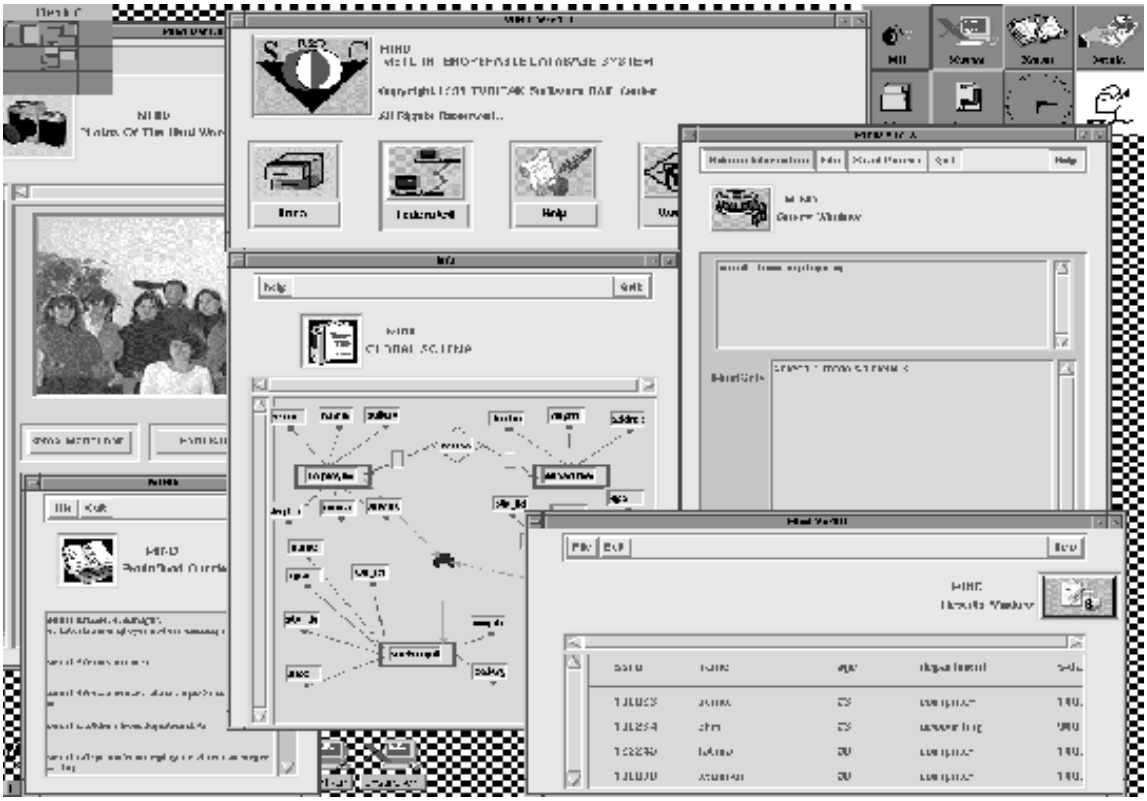

Figure 3. MIND GUI Screen Snapshot

3. Incorporating Local Databases Into MIND

Implementing a multidatabase system on top of a distributed object management architecture like CORBA requires the definition of objects in IDL and providing their implementations. A fundamental design question, therefore, is the granularity of these objects. In registering a DBMS to CORBA, a row in a relational DBMS, an object in an object-oriented DBMS (OODBMS), a group of objects or a whole database can be registered as an individual object. CORBA would accept all of these definitions. When fine granularity objects, like tables, are registered as objects, all the DBMS functionalities to process these tables, like querying, transactional control, etc., must be supported by the multidatabase system itself. However, when an entire relational DBMS, for example, is registered as an object, all the DBMS functionality needed to process these tables are left to that DBMS.

Another consideration regarding granularity has to do with the capabilities of the particular ORB that is used. In case of ORBs that only support BOA (CORBA's Basic Object Adaptor), each insertion and deletion of classes necessitates recompilation of the IDL code and rebuilding of the server. Thus, if the object granularity is fine, these ORBs incur significant overhead. When an OODBMS adapter becomes available in the future, exporting fine granularity objects of OODBMSs will be more convenient since such an adapter will work in cooperation with the OODBMS to handle the objects it owns.

A second issue that needs to be addressed is the definition of the interfaces for these objects. Most commercial DBMSs provide routines necessary for data definition, data manipulation, query execution, and transaction control facilities through their Call Level Interface (CLI) [15, 16, 17]. This property makes is possible to define a generic database object interface through CORBA IDL to represent all the underlying DBMSs. CORBA allows multiple implementations of an interface. Hence it is possible to provide a different implementation of the generic database object for each of the local DBMSs. Design decisions related to the issues mentioned up to this point are explained in more detail in [18].

In MIND, local DBMSs to be integrated are encapsulated in a generic database object following the reasoning mentioned above. In the implementation of generic database object a virtual C++ class definition, namely *DatabaseImpl*, is provided for LDBMSs [19]. LDBMSs inherent DatabaseImpl as a base class for their own implementation, e.g. Oracle7 has a C++ class definition, *OracleImpl*, which inherits DatabaseImpl. DatabaseImpl class provides method templates for the operations defined in DBservices interface. These method templates and their functionalities are as follows:

- *ConnectToDB*: This method gets username and password as input arguments and builds a connection to the related DBMS. Username and password are retrieved from the Schema Information Service.
- *DatabaseSendQuery*: This method is used to execute a local query on a LDBMS. The method returns a status value indicating whether the execution is completed successfully or not.
- *GetNext*: This method gets the results of the query executed by DatabaseSendQuery method into DBresult structure which defined in CORBA IDL. Its functionality is similar to a cursor implementation of a DBMS. GetNext method should be called in a loop until all the rows are fetched.
- *BeginTrans*: This method starts a new transaction on the LDBMS and returns a transaction identifier (TID) if CLI of the corresponding LDBMS provides such a facility. Note that, some of the LDBMSs, such as Oracle, do not provide TIDs through their CLIs. In order to overcome the problems due to the restricted transaction facilities of these CLIs, XA interface of X/Open DTP[5] (distributed transaction processing), which provides 2PC protocol and basic transaction primitives, is used.
- *PrepareToCommit*: This method is used to send PrepareToCommit message to the related DBMS if it supports 2PC protocol.
- *CommitTrans*: This method is responsible from committing the transaction with the given TID at the related LDBMS.
- *AbortTrans*: This method is responsible from aborting the transaction with the given TID at the related LDBMS.
- *CheckNestedTic*: This method is used by Transaction Manager of MIND system in order to implement NTNT technique. It checks the ticket values of the transactions at each site to provide the global serializability.

4. Conclusion

In this paper, we describe the infrastructure of a multidatabase system, namely MIND, based on OMG's object management architecture. In order to build the MIND system, a global query manager, a global

[5] X/Open DTP is a registered trademark of X/Open Company, Ltd.

transaction manager, a schema integrator, interfaces to supported database systems and a graphical user interface have been designed and implemented. All of the component objects (QMO, QPO, LDAO etc.) of MIND are created by the Object Factory Server as a standard.

We have defined an interface of a generic database object accessible through CORBA and developed multiple implementations of this interface for Oracle7, Sybase, Adabas D and MOOD. Through this common interface primitive methods necessary for data definition, data manipulation, query execution, and transaction control facilities are provided. CLI and XA (when necessary) interfaces of the corresponding DBMSs are used for the implementation of this common interface. In registering the different DBMSs to CORBA, we have chosen the object granularity to be a database in order to avoid the cost of building all the DBMS functionality such as querying, transaction control, etc. on the multidatabase itself.

Our experience have indicated that CORBA offers a very useful methodology and a middleware to design and implement distributed object systems. To clarify the advantage of using CORBA in a multidatabase implementation, assume that CORBA is not available and we have only a client-server architecture and TCP/IP as the infrastructure. In such a case, implementing a distributed object-oriented architecture would require implementing the ORB-like functionality first. Since this requires tremendous amount of design and coding effort, it would be difficult to justify the cost for one specific multidatabase system.

Furthermore, using CORBA as a middleware made it possible for MIND to become an integral part of a broad distributed object system that not only contains DBMSs but may also include many objects of different kinds such as file systems, spreadsheets, workflow systems, etc.

References

[1] W. Kim, *Modern Database Systems*, pp.551-572.

[2] Object Management Group, "The Common Object Request Broker: Architecture and Specification", OMG Document Number 91.12.1, December 1991.

[3] Object Management Group, "The Common Object Services Specification", Vol. 1, OMG Document Number 94.1.1, January 1994.

[4] Kılıç, E., Özhan, G., Dengi, C., Kesim, N., Köksal, P. and Doğaç, A., "Experiences in Using CORBA in a Multidatabase Implementation", in Proc. of 6th Intl. Workshop on Database and Expert System Applications, London, Sept. 1995.

[5] A. Doğaç, C. Dengi, E. Kılıç, G. Özhan, F. Özcan, S. Nural, C. Evrendilek, U. Halıcı, B. Arpınar, P. Köksal, N. Kesim and S. Mancuhan, "METU Interoperable Database System", ACM SIGMOD Record, 24 (3), September 1995.

[6] A. Doğaç, C. Dengi, E. Kılıç, G. Özhan, F. Özcan, S. Nural, C. Evrendilek, U. Halıcı, B. Arpınar, P. Köksal and S. Mancuhan, "A Multidatabase System Implementation on CORBA", 6th Intl. Workshop on Research Issues in Data Engineering (RIDE-NDS '96), New Orleans, February 1996.

[7] A. Doğaç, U. Halıcı, E. Kılıç, G. Özhan, F. Özcan, S. Nural, C. Dengi, S. Mancuhan, B. Arpınar, P. Köksal and C. Evrendilek, "METU Interoperable Database System", Demo Description, in Proc. of ACM Sigmod Intl. Conf. on Management of Data, Montreal, June 1996.

[8] Doğaç, A., et.al., "METU Object-Oriented Database System", Demo Description, in the Proc. ACM SIGMOD Intl. Conf. on Management of Data, Minneapolis, May 1994.

[9] Doğaç, A., Altınel, A., Özkan, C., Durusoy, I., Altınta, I., "METU Object-Oriented DBMS Kernel", in Proc. of Intl. Conf. on Database and Expert Systems Applications, London, September 1995 (Lecture Notes in Computer Science, Springer-Verlag 1995).

[10] Nural, S., Köksal, P., Özcan, F., Doğaç, A., "Query Decomposition and Processing in Multidatabase Systems", in Proc. of the Intl. Conf. on Eng. Systems Design and Analysis, Montpellier, July 1996.

[11] Mancuhan, S. "A Graphical Tool for Schema Integration in a Multidatabase System", MSc. Thesis, Dept. of Computer Engineering, METU, in Preparation.

[12] Halıcı, U., Arpınar, B., and Doğaç, A., "Serializability of Nested Transactions in Multidatabases", in Proc. of International Conference on Database Theory (ICDT'97), January 1997.

[13] Özhan, G., "Design and Implementation of a Transaction Manager for a Multidatabase System", MSc. Thesis, Dept. of Computer Engineering, METU, in Preparation.

[14] Özcan, F., "Dynamic Query Optimization on a Distributed Object Management Platform", MSc. Thesis, Dept. of Computer Engineering, METU, February 1996.

[15] Programmer's Guide to the Oracle Call Interfaces, Oracle Corporation, December 1992.

[16] Open Client DB-Library / C Reference Manual, Sybase Inc., November 1990.

[17] ENTIRE SQL-DB Server Call Interface, ESD-311-316, SOFTWARE AG, April 1993.

[18] Doğaç, A., Dengi, C., Tamer, M., "Building Interoperable Databases on Distributed Object Management Platforms", Communications of the ACM, to appear.

[19] Özhan, G., Doğaç, A., Kılıç, E., Özcan, F., Nural, S., Dengi, C., Halıcı, U., Arpınar, B., Köksal, P., Mancuhan, S., Evrendilek, C., "Making Oracle7, Sybase, Adabas D Interoperable Through CORBA: MIND Project", in Proc. of EOUG'96, Amsterdam, April 1996.