METU Interoperable Database System

A. Dogac, C. Dengi, E. Kilic, G. Ozhan, F. Ozcan, S. Nural, C. Evrendilek,
U. Halici, B. Arpinar, P. Koksal, N. Kesim,* S. Mancuhan

Software Research and Development Center

Scientific and Technical Research Council of Turkiye

Middle East Technical University (METU)

06531 Ankara Turkiye

email: asuman@srdc.metu.edu.tr

Abstract

METU INteroperable Database System (MIND) is a multidatabase system that aims at achieving interoperability among heterogeneous, federated DBMSs. MIND architecture is based on OMG distributed object management model. It is implemented on top of a CORBA compliant ORB, namely, ObjectBroker. MIND provides users a single ODMG-93 compliant common data model, and a single global query language based on SQL. This makes it possible to incorporate both relational and object oriented databases into the system. Currently Oracle7, Sybase and METU OODBMS (MOOD) have been incorporated into MIND. The main components of MIND are a global query processor, a global transaction manager, a schema integrator, interfaces to supported database systems and a user graphical interface.

In MIND all local databases are encapsulated in a generic database object with a well defined single interface. This approach hides the differences between local databases from the rest of the system. The integration of export schemas is currently performed manually by using an object definition language (ODL) which is based on OMG's interface definition language. The DBA builds the integrated schema as a view over export schemas. The functionalities of ODL allow selection and restructuring of schema elements from existing local schemas.

MIND global query optimizer aims at maximizing the parallel execution of the intersite joins of the global subqueries. Through MIND global transaction manager, the serializable execution of the global transactions are provided.

1 Introduction

In today's enterprises information is typically distributed among multiple heterogeneous database management systems. 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 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 with limited capabilities. Thus, 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) is a database system that drives other database systems and allows the users to simultaneously access independent databases using a single data definition and manipulation language. The primary objective of a MDBS is to significantly enhance productivity in developing and executing applications that require simultaneous access against multiple independent databases. A multidatabase system provides a single global schema that represents an integration of the relevant portions of the underlying local databases. The users may formulate queries and updates against the global schema.

We have implemented a multidatabase system, namely, METU INteroperable Database system (MIND) on a CORBA compliant ORB implementation, namely, DEC's ObjectBroker. DEC's ObjectBroker runs on several platforms such as Windows, SunOS, AIX, Open VMS. Our implementation platform is SunOS. The source code of version 0.1 is available through anonymous ftp from ftp://ftp.srdc.metu.edu.tr/pub /mind/source/mind-V.0.1.tar.gz. In the near future we will provide a WWW gateway to MIND. Thus MIND will be accessible using a HTML browser such as Netscape or Mosaic.

CORBA(The Common Object Request Broker Architecture) which is developed by the Object Management Group (OMG) [OMG 91] is a specification and an architecture that provides implementation independent access to objects on heterogeneous systems. In other words CORBA handles the heterogeneity at the platform level. In CORBA, clients ask for work to be done and servers do that work, all in terms of tasks called operations that are performed on entities called objects. Applications interact with each other without knowing where the other applications are on

*Bilkent University, 06533, Ankara Turkiye

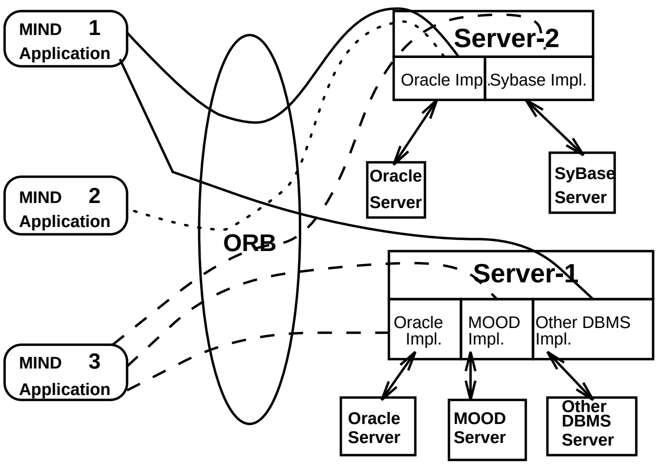

Figure 1: A General Overview of the System

the network or how they accomplish their tasks. By using CORBA's model, it is possible to encapsulate applications as sets of distributed objects and their associated operations so that one can plug and unplug those client and server capabilities as they need to be added or replaced in a distributed system. These properties provide the means to handle heterogeneity at the database level. Thus CORBA provides for an infrastructure for implementing a multidatabase system. Semantic interoperability remains to be solved at the application programming level.

We have defined an interface of a generic Database Object accessible through CORBA and developed multiple implementations of this interface for Oracle7, Sybase and MOOD (METU Object-Oriented Database System) [DEOO 94, Dog 94, Dog 95, DAOD 95]. The current implementation makes it possible to access any of these databases through CORBA using 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 a server site, the global subquery is executed by using the corresponding call level interface routines 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 by the client if necessary. A general overview of the system is presented in Figure 1.

The rest of the paper is organized as follows. The architecture of the MIND system is described in Section 2. Section 3 presents the infrastructure of the system. The design decisions and experiences in developing generic Database Object implementations for various DBMSs are discussed in this Section. Section 4 describes the schema integration in MIND. The global query manager of the system is briefly summarized in Section 5. Section 6 provides a short description of the global transaction manager.

2 MIND Architecture

MIND architecture is based on OMG distributed object management model. In other words, MIND allows clients (users and application programs) to access databases that reside anywhere in the environment transparently and without having knowledge of location or format of queries or operations involved. That is, what clients see are homogeneous objects acessible through a common interface. The common interface's data model (i.e., the canonical data model) is ODMG-93 [Cat 94] and the query language of the interface is ODMG-93 query language namely, OQL. Currently only a subset of OQL is operational.

An overall view of MIND Architecture is provided in Figure 2. The basic components of the system are as follows:

1. A collection of local database agents (LDA). Each LDA:
   - maintains export schemas provided by the local DBMSs represented in ODMG-93 data model,
   - translates the queries received in common query language (OQL) to the local query language and back.
2. A collection of global database agents (GDA). A GDA contains:
   - A global integrated schema,
   - a global query processor that is responsible from parsing, decomposing, and optimizing the queries [EDNO 95],
   - a global transaction manager that ensures serializability of multidatabase transactions without violating the autonomy of local databases.

A client accesses the MIND system through a Global Database Agent (GDA). For a client a GDA is a server. From GDA's point of view, LDAs are servers. Symmetrically from LDA's point of view a GDA is a client.

The infrastructure of the system is build on a CORBA implementation, namely, DEC's ObjectBroker [KOD 95]. The DBMSs currently registered to the system are Sybase, Oracle7 and MOOD[Dog 94, Dog 95, DOAD 95].

3 The Infrastructure of MIND

As an initial step in implementing the MIND system, we encapsulated Oracle7, Sybase, MOOD DBMSs in multiple implementations of a generic Database Object. The Database Object conveys requests from client to the underlying DBMSs by using the Call Level Interfaces of these DBMSs. The call level interfaces of these systems [ORACLE 92, SYBASE 90, Dog 95] support SQL data definition, data manipulation, query, and transaction control facilities. We have used C binding of call level interface to access these database servers. Results of the requests returned from the call level interfaces of underlying DBMSs are conveyed to the client through CORBA.

Our basic implementation decisions in registering different databases to CORBA are as follows:

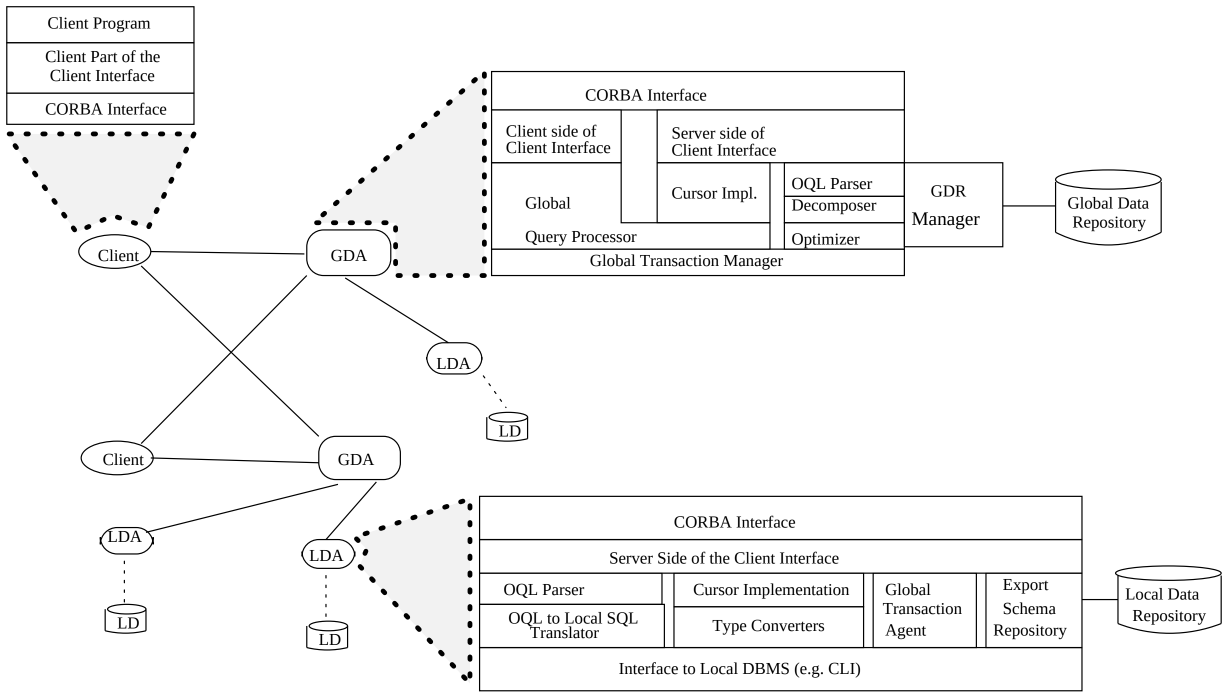

Figure 2: An Overview of the MIND Architecture

1) Object granularity: In CORBA, objects can be defined in different granularities. In registering a DBMS to CORBA an object can be a row in a relation or it can be a database itself. When fine granularity objects are registered, due to insertion and deletion of objects, it may be necessary to recompile the IDL code. Furthermore IDL code will be voluminous. Therefore, we have defined each database as an object.

2) Invocation style: CORBA allows both dynamic and stub-style invocation of the interfaces by the client.

In *stub-style* interface invocation, the client uses generated code templates (stubs) that cannot be changed at run time. In *dynamic interface*, the client defines and builds requests as it runs. We have chosen to use stub-style interface because in our current implementation all the objects are known and the interface that the clients use is not likely to change over time; there is no need for dynamic invocation. Figure 3 illustrates invoking an operation to a database object instance.

3) Mapping client requests to servers: In associating a client request with a server method, CORBA provides the following alternatives:

i. One interface to one implementation
ii. One interface to one of many implementations
iii. One interface to multiple implementations

When mapping one interface to one implementation, there is a direct one-to-one relationship between the operations in the interface and the methods in the implementation.

In the second alternative, for any one interface, it is possible to have several different implementations. Here again, there is a direct one-to-one relationship between the interface operations and the methods in the implementations.

The third alternative makes it possible to have multiple implementations associated with the same interface, with each implementation providing only a portion of the interface.

Since every database management system registered to CORBA provides methods for all of the operations that the interface definition specifies, the second alternative is sufficient for our purposes.

4) Object Life Cycle: A client application needs an object reference to make a request. In CORBA, the initial object reference can be obtained in one of the following three ways:

i. The application developer can build the object reference into the client. This requires the client application to be compiled together with the server application. During compilation a built-in object reference is generated in both codes. Thus the first request is statistically bound to this built-in object reference.

ii. The client can get the object reference from an external source. This external source could be a file or a database containing initial object references in string form.

iii. ObjectBroker allows servers to store initial object references in the Advertisement Partition of Registery. Then clients can access the Advertisement Partition to obtain object references.

The first approach makes the system inflexible since it is impossible to change a statically bound object reference

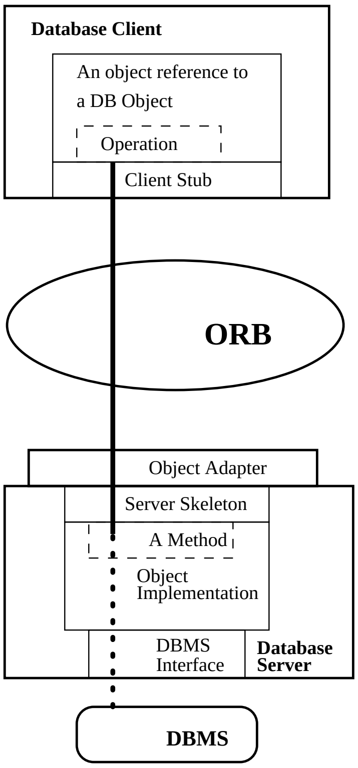

Figure 3: Invoking an operation to a Database Object instance through ORB

without recompiling the system.

The second approach is not suitable either, since it is not feasible to maintain or replicate a single file or a database as an external source in a distributed environment.

Therefore, we have chosen the third approach to get initial object references.We have implemented a server, namely DB_Factory which puts an initial object in the Advertisement Partition of Registry of ORB. An object reference selected from the Advertisement Partition is then used to get the object references of the corresponding DBMS objects. In other words this object acts as an object factory for database objects. In this way each client has its own database object to communicate with the DBMS servers.

5) Activation Policy: A server code contains one of several implementations for the interface defined one for each of the databases. So every site can have any of the supported database servers. There is a direct one-to-one relationship between the interface operations and the methods in the implementations.

When registering different databases to CORBA, one has to specify an activation policy for the implementation of each DBMS. This policy identifies how each implementation gets started. An implementation may support one of the following activation policies :

i. Shared : The server can support more than one object for the implementation.

ii. Unshared : The server can support only one object at a time for the implementation.

iii. Server_per_method : A new server is used for each method invocation.

iv. Persistent : The server is never started automatically. Once started, it is the same as the shared policy.

For the time being we have chosen the shared activation policy to our server implementations since it is used when multiple objects are related and handled by the implementation. In our system, more than one client may request an object of a database implementation. Therefore, the implementations of DBMSs support shared activation policy.

4 Schema Integration in MIND

MIND implements a four-level schema architecture that addresses the requirements of dealing with distribution, autonomy and heterogeneity in a multidatabase system. This schema architecture includes four different kinds of schemas:

1) Local Schema: A local schema is the conceptual schema of a export database system. A local schema is expressed in the native data model of the export database and hence different local schemas may be expressed in different data models.

2) Export Schema: A export schema is derived by translating local schemas into a canonical data model (which is ODMG-93 [Cat 94] in our case). The process of schema translation from a local schema to a export schema generates mappings between the local schema objects and the export schema objects.

3) Derived (Federated) Schema: A derived schema combines the independent export schemas to a (set of) integrated schema(s). A federated schema also includes the information on data distribution (mappings) that is generated when integrating export schemas. The global query processor transforms commands on the federated schema into a set of commands on one or more export schemas.

4) External Schema: In addition, it should be possible to store additional information that is not derived from export databases. An external schema defines a schema for a user or application. An external schema can be used to specify a subset of information in a federated schema that is relevant to the users of the external schema. Additional integrity constraints can also be specified in the external schema.

The classes in export and derived schemas behave like ordinary object classes. They consist of an interface and an implementation. But unlike ordinary classes, which store their objects directly, the implementation of the classes in these schemas derives their objects from the objects of other classes.

4.1 MIND Schema Integrator

In MIND, schema integration is a two phase process:

1) Investigation phase: First commonalities and discrepancies among the export schemas are determined. This phase is manual. That is the DBA examines export schemas

and defines the applicable set of inter-schema correspondences. The basic idea is to evaluate some degree of similarity between two or more descriptions, mainly based on matching names, structures and constraints. The identified correspondences are prompted according to the classification of schema conflicts. The classification of schema conflicts are not provided in this paper due to space limitation.

2) Integration phase: The integrated schema is built according to the inter-schema correspondences. The integration phase cannot be fully automated. Interaction with the DBA is required to solve conflicts among export schemas. In MIND, the integration of export schemas is currently performed manually by using an object definition language (ODL) which is based on OMG's interface definition language. The DBA builds the integrated schema as a view over export schemas. The functionalities of ODL allow selection and restructuring of schema elements from existing local schemas.

In ODL, a schema definition is a list of interface definitions whose general form looks as follows:

```
interface classname:superclass_list {
     extent          extentname;
     keys       attr1;
     attribute attr_type attr1;
     ...
     relationship OtherClass relname
   inverse OtherClass::invrel;
     ...
}
```

where classname is the name of the class whose interface is defined; superclass_list includes all superclasses of the class; extentname provides access to the set of all instances of the class; keys allows to define a set of attributes which uniquely identifies each object of the class.

In addition to its interface definition, each class needs information on how to determine the extent and how to map the attributes onto the local ones. The general syntax for this mapping definition is as follows [HFBK 94]:

```
mapping classname {
     origin          typename orig1, ...
     def_ext    extname  as
               select classname(orig1:i1,...)
               from ... in ...
               where ...;
     def_attr attname as query; ...
     def_rel pathname as [element (]
            select classname(orig1:i1, ...)
            from ... in ...
            where ... [)];
}
```

The keyword mapping marks the block as a mapping definition for the derived class classname. The origin clauses define a set of private attributes that store the back-references to those objects, from which an instance of this class has been derived. The extent derivation clause starting with def_ext defines a query that provides full instantiation of the derived class. A list of def_attr lines defines the mapping for each attribute of the class. And finally a set of def_rel lines express the relationships between derived classes as separate queries which actually represent the traversal of path definitions.

These are all specifications necessary to describe the federated schema(s). We are currently developing a graphical tool which will automatically generate these textual specification of class derivations. Our ultimate aim is to establish a semi-automated technique for deriving an integrated schema from a set of assertions that state the inter-schema correspondences. The assertions will be derived as a result of the investigation phase. To each type of assertion there will correspond an integration rule so that the system knows what to do to build the integrated schema.

5 Query Optimization in the MIND project

MIND query optimizer tries to maximize the parallel execution of intersite joins of the global subqueries [EDNO 95]. For this purpose, a two step heuristic algorithm has been developed. The first step produces a linear order from the query graph where the most profitable joins appear together in the linear sequence. In the second step, this order is exploited to maximize the parallelism in execution of intersite joins by also taking the appearance times, the communication costs and the conversion costs of global subqueries into account. The algorithm produces the schedule, that is, the assignment of join pairs to the sites.

Yet another problem considered in MIND query optimization scheme is data replication. We formulate the query decomposition in multidatabases in case of data replication as the following optimization problem [EDNO 95]:

- given an initial assignment of relations and fragments to the sites, distribute the modified and decomposed query to the sites such that the independent parallel execution time is optimized, i.e.,the load distribution is balanced.

For this NP-Complete assignment problem a heuristic algorithm has also been developed [EDNO 95].

6 MIND Transaction Management

In MIND, Global Transaction Manager (GTM) controls the execution of global transactions that access data across sites to preserve consistency (Figure 2). The correctness criterion for the execution of MIND transactions is chain conflicting serializability [ZE 93]. Transaction Management is performed in a distributed manner, therefore global concurrency control (gcc) is more resilient to site failures than a centralized gcc.

The problem in providing global serializability in multidatabases lies in the following: even though the local schedulers provide serializable executions and the execution order of the global transactions at all local sites are consistent, the global serializabilty may still be violated. Even tough global transactions are submitted and committed at all sites in the same order, their serialization order can change because of indirect conflicts caused by local transactions.

In [GRS 94] a practical solution namely the ticket method is suggested to enforce serializability of global transactions in an MDBS environment. In this method, artificial conflicts are introduced between multidatabase transactions at each site that they are executing by reading and writing a database item called a ticket. Ticket values determine the serialization order of multidatabase transactions at each site. When the local serialization orders of global transactions are consistent at all sites then the global serializability is ensured.

In MIND the original ticket idea is extended and implemented in a distributed manner. Global transactions can be submitted to any of the participating sites which is capable of coordinating their execution. The GTM to which a global transaction is submitted becomes the coordinator for that transaction. GTM employees an optimistic scheduling algorithm which assigns a global timestamp (or global ticket value) to each global transaction submitted to MDBS. Global ticket value is assigned according to submission order of transactions to the system and incremented monotonically. Global ticket values are maintained distributedly and it is not necessary to obtain tickets from a specific site. GTM determines sites involved in the global transaction in coordination with the global query optimizer and distributes the global subtransactions to Global Transaction Agent (GTA)s accordingly (Figure 2). GTAs at each site are responsible from processing global subtransactions and submitting them to Local DBMS (LDBS)s.

Global subtransactions are serialized in the timestamp order at all sites. To enforce the timestamp serialization order of global transactions at all sites, MDBS requires each subtransaction of a global transaction to perform additional data manipulation operations on a common data item called ticket stored in the local database. This technique introduces forced local conflicts between the subtransactions of global transactions at each LDBS. The ticket operations guarantee that the local serialization order is equivalent to the order of ticket operations, or subtransaction is aborted by the local system. Hence, global serializability is maintained. Currently both the global query optimizer and the global transaction manager is being integrated to the system.

As a final word, the goal of this first prototype is to provide clearer insights to the issues involved in a multidatabase implementation based on OMG's distributed object management model. And it turns out that the lessons being learned from the ongoing implementation efforts iteratively effect the design decisions.

References

[Cat 94] Cattell, R.G.G., The Object Database Standard: ODMG-93, Morgan Kaufmann, 1994.

[DEC 94a] The Guide to CORBA, Digital Equipment Corp., August 1994.

[DEC 94b] ObjectBroker, System Integrator's Guide, Digital Equipment Corp., August 1994.

[DEOO 94] Dogac, A., Evrendilek, C., Okay, T., Ozkan, C., "METU Object- Oriented DBMS", in Object-Oriented Database Systems, edited by Dogac, A., Ozsu, T., Biliris, A., Sellis, T., Springer-Verlag, 1994.

[Dog 94] Dogac, A., et. al., "METU Object-Oriented Database System", Demo Description, in the Proc. ACM SIGMOD Intl. Conf. on Management of Data, Minneapolis, May 1994.

[Dog 95] Dogac, A., Altinel, A., Ozkan, C., Durusoy, I., Altintas, 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).

[DAOD 95] Dogac, A., Altinel, M., Ozkan, C., Durusoy, I., "Implementation Aspects of an Object-Oriented DBMS", in ACM SIGMOD Record, Vol.24, No.1, March 1995.

[EDNO 95] Evrendilek, C., Dogac, A., Nural, S., Ozcan, F., "Query Optimization in Multidatabase Systems", in Proc. of the Next Generation Information Technologies and Systems, Israel, June 1995.

[GRS 94] D. Georgakopoulos, M. Rusinkiewicz and A. Sheth, "Using Tickets to Enforce the Serializability of Multidatabase Transactions", IEEE Trans. on Data and Knowledge Eng., Vol. 6, No.1, 1994.

[HFBK 94] Huck, G., Fankhauser, P., Busse, R., Klas, W., "IRO-DB : An Object-Oriented Approach towards Federated and Interoprable DBMSs", in Proc. of ADBIS '94, Moscow, May 1994.

[KOD 95] E. Kilic, G. Ozhan, C. Dengi, N. Kesim, P. Koksal and A. Dogac, "Experiences in Using CORBA in a Multidatabase Implementation", in Proc. of 6th Intl. Workshop on Database and Expert System Applications, London, Sept. 1995.

[OMG 91] Object Management Group, "The Common Object Request Broker: Architecture and Specification", OMG Document Number 91.12.1, December 1991.

[OMG 94] Object Management Group, "The Common Object Services Specification, Volume 1", OMG Document Number 94.1.1, January 1994.

[ORACLE 92] Programmer's Guide to the Oracle Call Interfaces, Oracle Corporation, December 1992.

[SYBASE 90] Open Client DB-Library /C Reference Manual, Sybase Inc., November 1990.

[ZE 93] A. Zhang and A. Elmagarmid, "A Theory of Global Concurrency Control in Multidatabase Systems", Journal of VLDB, Vol. 2, No. 3, 1993.