Experiences in Using CORBA for a Multidatabase Implementation

Ebru Kilic, Gokhan Ozhan, Cevdet Dengi, Nihan Kesim*, Pinar Koksal
and Asuman Dogac

Software Research and Development Center
Scientific and Technical Research Council of Turkiye
Middle East Technical University
06531, Ankara Turkiye
e-mail: asuman@srdc.metu.edu.tr

Abstract

One way of achieving interoperability among heterogeneous, federated DBMSs is through a multidatabase system which supports a single common data model and a single global query language on top of different types of existing systems. In this paper we describe the use of a CORBA implementation in developing a multidatabase system. We present our design choices and experiences in making various databases CORBA compliant. We have so far registered Oracle7, Sybase and MOOD to CORBA and the current implementation makes it possible to access these databases simultaneously through a generic interface using a global query language based on SQL.

1 Introduction

In today's enterprises information is typically distributed among multiple database management systems. Therefore there is a need to access and share data across these systems. 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 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 (MDB) is a database system that drives other database and file systems and allows the users to simultaneously access independent databases and files using a single data definition and manipulation language. The primary objective of a MDB 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. This in turn requires the support of a single common data model and a single data definition and manipulation language. The users may formulate queries and updates against the global schema.

Several approaches have been proposed to address the issues of integrating hete rogeneous database systems. The IRO-DB [HFBK 94] project proposes a client-server system architecture based on three layers to achieve interoperability among local databases: a Local layer, a Communication layer and an Interoperability layer. The local layer on top of each local database maps

*Bilkent University, 06533, Ankara Turkiye

the individual data models into a common data model which serves as the canonical exchange data model. The Communication layer provides means to access these databases through a network and transfers queries and the results to and from a client cite to a local server. The interoperability layer contains all the components necessary to allow for integrated application access to local databases.

The local layer and communication layer give means to realize loosely coupled federations. We have implemented the infrastructure of the multidatabase system through CORBA(The Common Object Request Broker Architecture) which is developed by the Object Management Group (OMG) [OMG 91]. CORBA is a specification and an architecture that provides implementation independent access to objects on heterogeneous systems. 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 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.

Our ultimate aim is to build a multidatabase system, MIND (METU INteroperable Database System), complete with its global query manager, global transaction manager and schema integrator. In this paper we discuss our experiences in registering different database management systems to CORBA. We are currently using the CORBA implementation of Digital Equipment Corporation, namely ObjectBroker [DEC 94a, DEC 94b]. ObjectBroker runs on several platforms such as Windows, SunOS, AIX, Open VMS. We have so far 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 94a, 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 local 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 local 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. In Section 2 we describe the CORBA system in more detail. Section 3 discusses how CORBA fits into the area of distributed computing. In section 4 we present the design decisions and experiences in developing generic Database Object implementations for various DBMSs. Section 5 concludes the paper by a summary and presents the future work related with the MIND project.

2 What is CORBA?

CORBA is the core communication mechanism which enables distributed objects to operate on each other. There is another complementary standard developed by the OMG, called Common Object Services Specification (COSS), for integrating distributed objects. COSS [OMG 94] provides a set of standard functions to create objects, control access to objects keep track of objects and object references.

CORBA (the Common Object Request Broker Architecture) [OMG 91] is both an architecture and a specification for distributed object-oriented computing that has implementations currently available from several major software vendors (e.g. ObjectBroker-DEC, DistributedObjectsEverywhere-SunSoft, System Object Model-IBM, Distributed Smalltalk-HP, Orbix-Iona Technologies). It is the first of the next generation of software that gives its users plug-and-play software reusability by bringing object-oriented computing and distributed computing together. It should be noted that some CORBA implementations also contain COSS features such as Object Life Cycle Services, Name Services and Event Services [OMG 94].

CORBA uses an object-oriented model and defines the Object Request Broker (ORB) as an intermediary between clients and servers. In this model, clients send requests to the ORB asking for certain services to be performed by whatever servers can fulfill those needs. Only the ORB needs to know the locations of CORBA clients and servers on the network, that is, it is completely transparent to the client where the server is on the network and similarly, the location of the client is transparent to the server. A library of functions, which is called Basic Object Adapter, is used by the server program to locate and initialize the server implementa-

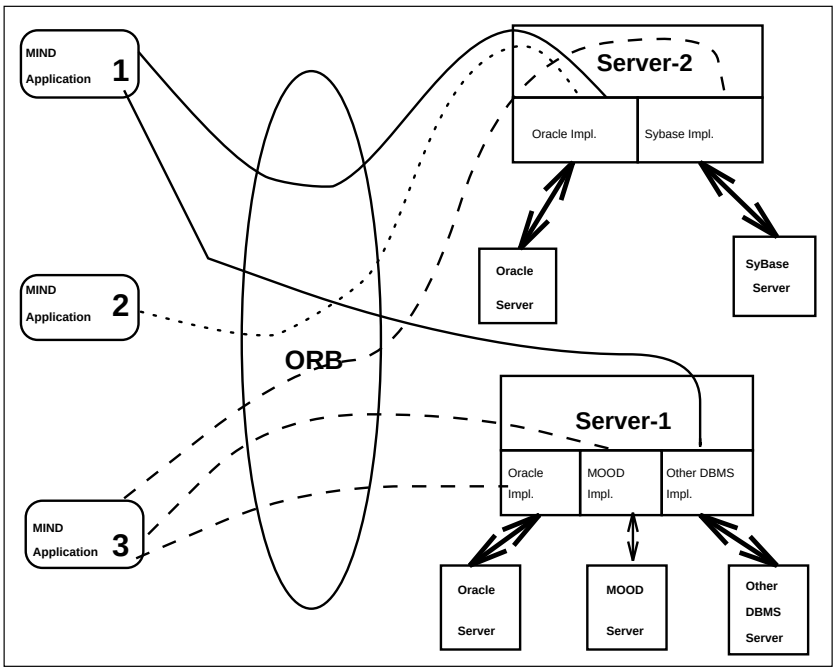

Figure 1: A general overview of the system

tion and to invoke the appropriate method to satisfy the client request.

In CORBA there is a formal separation between the client and the server. That is, a client using CORBA's object-oriented interface does not need to know how the server accomplishes its task - it only needs to know how to call the server that does the work. CORBA achieves this separation by restricting communication between the client and the server to a type of message called a *request*. Each request is sent from the client to the server and it contains an operation to be performed and a specific object on which the operation is to be performed. The objects and operations that a client can request and to which a server can respond are defined by the interface that both the client and the server support. In object-oriented terminology an interface is very similar to a class definition which defines the characteristics and behavior of a kind of object [DEC 94a].

In CORBA, interfaces are defined using the Interface Definition Language (IDL). The following example shows a portion of an interface defined using the IDL.

```
    interface Employee
{
    void promote ( in char NewJobClass );
    void dismiss ( in DismissCode reason,
                   in string description );
};
```

In this example two operations, **promote** and **dismiss**, are defined as a part of the employee interface. This means that a client which has this interface defined can request that a server promote or dismiss an occurrence of an employee object. For each such request there is a CORBA implementation which actually accomplishes the client's request. An implementation exists in a server and contains one or more methods for each request. When a client requests a promote operation on a specific employee, the ORB receives this request and finds an implementation of the employee object in a server application that can do the requested work. The implementation uses its methods to actually do the job of promoting that employee.

When an IDL code is compiled by the CORBA compiler, it generates client stubs and server skeletons. The server skeleton makes it possible for the ORB and an object adapter to translate the client request to a specific method on the server. A client stub maps IDL operation definitions for an object type into procedural routines that is called to invoke a request.

CORBA's object-oriented model provides several benefits that make it easier to integrate applications into a distributed system. First of all it allows the designers to take the advantage of many object-oriented design techniques, such as encapsulation, information-hiding

and inheritance.Object-oriented design techniques encourage the designer to define each object in the system as a black box that is capable of performing certain tasks, without the system knowing how that black box accomplishes those tasks.

Second, it provides more clearly defined interfaces between the parts of the system, which can be changed without affecting the entire system. This enhances software modularity so that software components can be worked on more independently. When a new capability is added to the system, it is simply defined as a set of new operations on one or more changed objects.

Third, the technique of inheritance promotes the reuse of software by allowing the properties and behaviors of one object to be subsumed by another. And finally, CORBA is not a pure object-oriented system with its own language, but a hybrid object-oriented system that allows one to create an object-oriented system using more familiar programming languages, such as C and C++.

3 CORBA and Distributed Computing

In simple terms, distributed computing is two or more pieces of software sharing information with each other. These two pieces of software could be running on the same machine or they could be running on different machines connected to the same network. Most of the current distributed computing systems are based on a client/server model. In this model there are two types of software. The software that requests the information or service is referred to as the client and the software that provides the information or service is referred to as the server.

CORBA extends the use of this traditional distributed computing in many different ways. The enhancements that CORBA brings can be summarized as follows:

In traditional client/server thinking, the server and the client are each thought of as a single process. This is not necessarily so in CORBA. In CORBA, although clients are typically a single process, servers may or may not be single processes. A CORBA server could be a single process, a server that itself calls on other servers to actually perform the tasks the client has requested or a shareable piece of code that is called by application processes. In addition, CORBA does not assume a one-to-one relationship between clients and servers. Multiple servers can work with a single client or a single server can work with multiple clients. As stated previously, servers and clients find each other through the ORB rather than knowing directly about each other.

CORBA allows both synchronous and asynchronous communication styles. Synchronous communication is when one piece of software sends a message to another piece of software and then waits for a reply. Asynchronous communication is when a piece of software sends a message to another piece of software and then continues working, expecting the reply to come at some later time. Which style is a better choice is something that depends on the application needs. Rather than support one or the other, CORBA provides both communication styles, which makes it more flexible.

Another flexibility that CORBA brings as a byproduct of its object-oriented approach is that it blurs the distinction between a client and a server. In traditional client/server terminology, a client and a server have a master/slave relationship. That is, the client is always the one to request a task (master), and the server is always the one to perform the requested task (slave). This relationship has been made more flexible by the object-oriented distribution model of CORBA. This model is centered on the use of objects and thus a piece of software can act as a client for one request and to act as a server for the next request. In this way, CORBA lets the user to think in terms of consumer and producer applications rather than client and server applications.

4 The Implementation Decisions and Experiences in Registering DBMSs to CORBA

The advantages of CORBA discussed in the previous sections proves it to be an effective tool for providing interoperability in heterogeneous environments. Therefore we have decided to use CORBA in the implementation of our multidatabase system.

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 interfaces of these systems [ORACLE 92, SYBASE 90, Dog 94b] support SQL data definition, data manipulation, query, and transaction control facilities. We have used C++ 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:

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 registering fine granularity objects to CORBA, there is a need for a powerful object repository to store and efficiently access the objects. However ObjectBroker does not support transaction management and query processing facilities for its object repository. When coarse objects are registered, the problem of maintaining a large object repository is avoided.

We have defined each database as an object and have left the type conversion, global query processing and global transaction management to the related components of our multidatabase system, that is, the schema integrator, global query processor and global transaction manager respectively.

Figure 2 illustrates invoking an operation to a database object instance.

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. Stubs provide code for invoking methods so that the method invocation is similar to a standard procedure call, thereby reducing the complexity of method invocation.

In *dynamic interface*, the client defines and builds requests as it runs. The dynamic interface, provides clients with more flexibility, allowing the use of deferred synchronous operations and new interfaces. It is best used when application needs to discover new types of objects at run time. 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. Furthermore the dynamic interface requires writing complex codes whereas it is possible to write simple codes for stub interfaces. Also with the stub-style invocation it is easier to limit the control a user has over any database server and implementation selection.

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. Although the simplicity of this approach is an advantage, it has the disadvantage of being fairly inflexible because each operation must be supported by a method in the implementation, and the operations and methods must match each other exactly. If a method is added to the impleme ntation, the corresponding interface definition that the implementation supports, must also be modified. Similarly, adding a new operation to the interface used by the client would require adding a new matching method to the implementation. This inflexibility would force us to write different interface definitions for each database management system.

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, however, one can have several implementations which might be on different servers. Allowing multiple implementations gives the flexibility to have different, but similar, implementations for the same in terface.

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, clients cannot create objects, only the servers have that capability. Therefore client applications generally get object references by invoking a request to the ORB; however, since each request requires an object reference as an argument in the first place, there is going to be a problem for the first object reference that the client needs.

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 Registry. Then clients can access the Advertisement Parti-

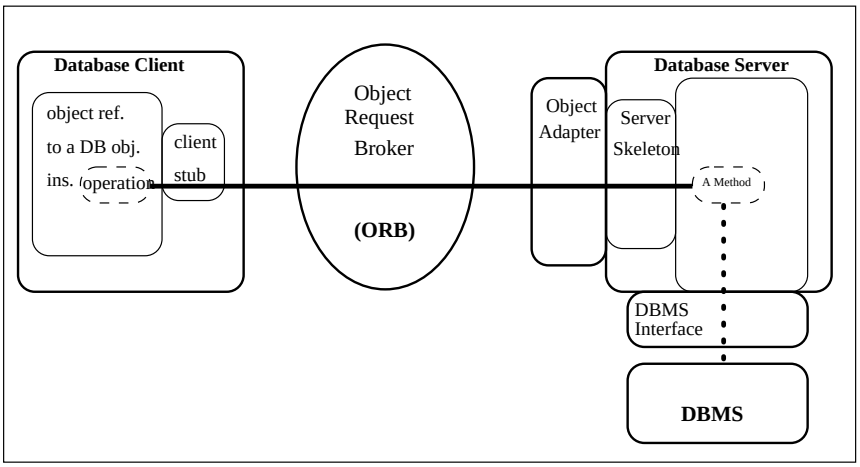

Figure 2: Invoking an operation to a Database Object instance through ORB

tion to obtain object references using the Naming Service of COSS.

The first approach makes the system inflexible since it requires that all client applications must know the statically bound object references to make any requests.

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. Every DBMS server puts an initial object in the Advertisement Partition of Registry of ORB. Object references selected from the Advertisement Partition are then used to get the object references of the corresponding DBMS objects and to connect the client to that specific database. In this way each client has its own object to communicate with the DBMS servers. This approach also allows users to discover new objects as they become available in the Advertisement Partition. Thus clients will be aware of new database servers connected to the network and can send requests to these DBMSs. Discovering new objects is something that can not be done in the first two approaches.

5) Activation Policy: A server code contains several implementations for the interface defined 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.

The shared activation policy is best suited 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 must support shared activation policy. This policy also allows the server to be more flexible.

6) ORB Binding policies: ORB has different binding policy options to resolve object references. Note that there could be more than one implementation for an interface. For example, in our case, there are different implementations of our generic Database Object for Oracle7, Sybase and MOOD. In binding a request to an implementation ORB provides the following policy options:

i. Static : All methods for an implementation must be executed by one specific server.

ii. Automatic: All methods for an implementation must be executed by the same server, but the server can be any server that can handle that implementation.

iii. Dynamic: Each method for an implementation can be executed by a different server.

The static binding is best used when only one, spe-

cific implementation can handle an interface's operations for the duration of the process. But in our implementation any server can support more than one implementation of the interface's operations. So static binding is also not suitable for our implementation.

In our implementation all the interface's operations are handled by each of the DBMS's implementation. So there is no need to use dynamic binding which is best when the interface's operations can be processed independently but worse on performance than the other bindings since method resolution is necessary for each method invocation.

We selected automatic binding so that the interface's operations are all handled by one implementation but also, more than one instance of an implementation can exist at a time. Performance of servers with automatic binding is generally better than for servers with dynamic binding because method resolution is bypassed after the initial method invocation. Automatic binding provides more than one instance of an implementation.

7) Event Notification: In COSS, there are some event notification routines for the management of objects and implementations. These routines allow the Basic Object Agent (BOA) to tell the application when events occur, so that the application can react in an event-driven way. ObjectBroker supports event notification. We use these routines when deactivating objects and implementations. For object or implementation deactivation, event notification routines are used to manage servers from the command line so that system managers can ask any implementation to shut itself down.

5 Conclusions

In this paper we describe our experiences in using CORBA for a multidatabase implementation. 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 DBMS).

The basic decisions involved in the process are as follows: In registering the different DBMSs to CORBA, we have chosen the object granularity to be a database. 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; thus there is no need for dynamic invocation. In associating a client request with a server method, we have chosen single server multiple implementation method since every database management system registered to CORBA provides methods for all of the operations that the interface definition specifies. In order to obtain the initial object reference, the naming service provided by the ORB is used. We have decided to support shared activation policy since it allows more than one client to request an object of a database implementation at the same time. We selected automatic binding so that the interface operations are all handled by one implementation but also, more than one instance of an implementation can exist at a time. And finally we use event notification routines when deactivating objects and implementations.

We have implemented all these design decisions and tested them on the three databases mentioned above. As a result of our experiences we can state that CORBA proved to be a highly effective architecture for the implementation of the communication layer of a multidatabase system. At the moment a global query language based on SQL is supported. A global SQL query is processed by the query manager of the MIND and the subqueries are sent to the corresponding databases through CORBA. For global transaction management a fully decentralized global concurrency control mechanism based on tickets has been implemented. This solves the global serializability problem. A global crash recovery system is yet to be designed.

The future work includes designing and implementing a schema integrator on top of the existing system. There is going to be a global schema which is defined as the integration of the schemas exported from the underlying databases. The global schema will support the ODMG-93 data model [Cat 94] with the extensions suggested in [GAR 95] and OQL [Cat 94] as the global query language. The global OQL queries will be translated into local SQL queries which are going to be sent to the local databases through CORBA. Also the design and implementation of a global query optimizer is under development [ED 95]

References

[GAR 95] M.J.Carey, et. al., "Towards Heterogeneous Multimedia Information Systems: The Garlic Approach", in Proc. of RIDE-ROM '95

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

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

[DEC 94b] ObjectBroker, System Integrator's Guide, Digital Equipment Coop., 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 94a] 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 94b] Dogac, A., "The MOOD User Manual", May 1994.

[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.

[ED 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.

[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.

[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.