THE HYBRID OBJECT-RELATIONAL ARCHITECTURE (HORA)
An Integration of Object-Oriented and Relational Technology

Jeff Sutherland
ODB, Cambridge, MA

Matthew Pope
Marcam Canada, Burlington, Ontario

Ken Rugg
Object Design, Burlington, MA

Keywords: OODBMS RDBMS DATA STORAGE ARCHITECTURE

Abstract
Many organizations with investments in relational database management systems (RDBMS) want to build object-oriented applications supporting a graphical user interface without forcing programmers to deal with SQL and RDBMS limitations. This paper proposes a natural and relatively transparent coupling of object-oriented clients with relational database server technology for new applications. The specified architecture can concurrently deliver key features of both The Object-Oriented Database Manifesto[1] and the Third Generation Database System Manifesto[2]. The Hybrid Object-Relational Architecture (HORA) is designed to:

- support ANSI SQL III object-oriented functionality using currently available relational database systems
- provide good server performance to both relational <u>and</u> object applications
- provide full object storage support by the RDBMS
- allow the specification of high-level referential integrity rules and user-specified constraints without SQL coding.

The HORA approach allows read-only access from existing relational applications and is not designed for updates to legacy systems without using the object manager.

Introduction
The two approaches most often used to integrate object applications with relational database management systems (RDBMS) are:

- Direct connection of an object application to legacy relational datastores without modification of the relational schema.
- Connection of an object application to an object manager which translates persistent objects into relational tables eliminating the need for embedded SQL in the object application.

The first approach has been widely used in recent years and is part of many vendors language products, class libraries, and object database systems. It often requires hardwiring SQL code in the object client which negatively impacts flexibility and performance. Recent detailed data on a production system of this type for an aerospace application showed that 35-55% of the resources used to build an object-oriented application were wasted on rework generated by changes to the relational database schema during the development phase of the project.[3]

The second approach was initially implemented in the commercial object database product G-BASE/SQL which allows G-BASE developers to store objects transparently in an Oracle object server.[4] More recently, Hewlett Packard announced OpenODB which provides a layer of object services for an underlying RDBMS.[5] Both of these products allow object-oriented programmers to build new object applications independent of many of the limitations of the underlying RDBMS.

ACM-SAC '93/2/93/IN, USA
© 1993 ACM 0-89791-568-2/93/0002/0326...$1.50

In many environments, available object/relational database products may not be suitable. For those cases, this paper describes how to build object services from the bottom up that will allow persistent object storage in an RDBMS for new applications. Since the object services must manage the schema of the relational database, legacy relational applications may read the objects directly, but may not update the database without using the object manager.

The approach presented has been optimized since its initial implementation in GBASE/SQL.[6,7,8] A similar design is inherent in the emerging specification of ISO/ANSI SQL III scheduled for release in 1996.[9] At that time all compliant relational database systems will become object-oriented.

The remainder of this paper specifies the architecture for the Hybrid Object/Relational Architecture (HORA) approach. Details of implementation of objects in relational tables in the RDBMS are carefully explained. Functionality that must be supported by an object manager that uses the tables is described at a high level. Details of the Object Manager design are beyond the scope of this article. An Object SQL and C++ syntax are used to illustrate a simple example.

Hybrid Obect-Relational Architecture
The benefits of HORA are power, performance, and safety. There are many nieve approaches to storing objects in a relational database. Rumbaugh[10] provides a good overview of strategies for storing classes in relational tables. This paper presents new experience in building systems with all the power of a true object database, within the inherent limitations of RDBMS perofrmance. The power of HORA is that it is based on over a decade of object database research and product implementation[11]. The performance of HORA has been optimized due to repeated implementation in distributed enterprise environments over many years. Safety is based on experience which guarantees that HORA will work well in commercial environments.

HORA provides the following benefits:

- A rich object Meta-model can be stored directly in the RDBMS. This Meta-model information is much closer to ANSI SQL III object-oriented capabilities than the limited constraint definition and stored procedure capabilities of ANSI SQL II.
- Distributed client applications can locally run the object manager that performs dynamic mapping from objects to relational tables while enforcing user-defined integrity constraints. This allows "atomic" changes from one valid object state to another.
- Object applications can be insulated from RDBMS schema changes through minor modifications of the Meta-model.
- These distributed capabilities offload the RDBMS and improve relational server performance beyond server based SQL III implementations. Performance has been shown to be as efficient as typical relational applications.

HORA provides each class with a relational table. The records in the table will store object instances. In most cases, this approach will look like a normalized database to a relational application. However, since relational applications have no notion of object identity (which is maintained by HORA), they may not be allowed to bypass the object manager to update the database.

HORA will not provide the performance of a pure object database for complex objects because it will be constrained by the performance of the underlying relational database. It is designed for adding new object applications to currently available relational database systems which have become standard technology in most organizations.

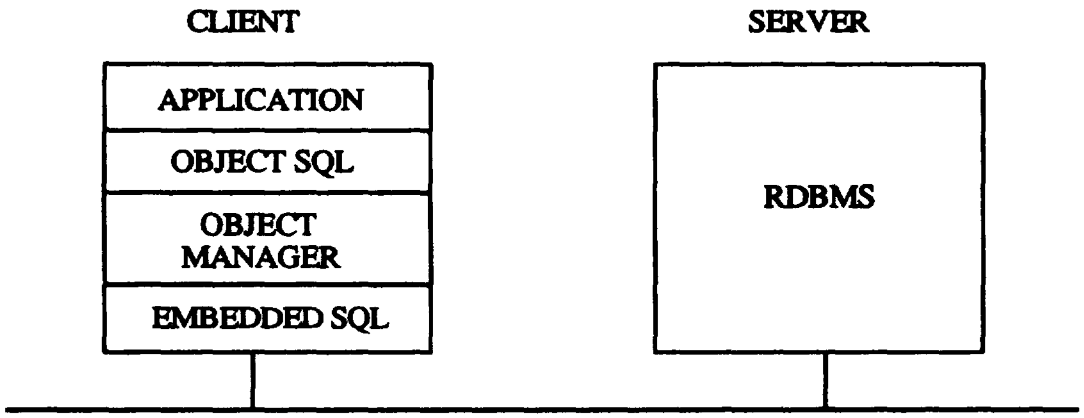


Figure 1

Hybrid Obect-Relational Architecture

In a commercial transaction processing environment, the client/server architecture in Figure 1 has been found useful.[7] An object-oriented client application communicates with the relational database through an object access layer using an Object SQL (OSQL) dialect. The Object Manager translates OSQL requests into one or more standard SQL commands to retrieve objects from the relational database.

The OSQL and Object Manager layers consist of class libraries with store and retrieve methods. OSQL provides a clear command structure and can be easily understood by SQL programmers. The Object Manager can provide a direct pass through of ANSI standard SQL commands[12] in order to provide some support conventional applications. In this case it must enforce Codd's Nonsubversion Rule[13] for relational systems by ensuring that standard relational statements do not bypass object integrity constraints.

Object Identity in the Relational Database

In order to reduce the work required to build an object-oriented application and retain the power of the object architecture, HORA gives every object a unique, system defined, object identifier.[14] In a relational system, objects are not guaranteed unique identity because they have keys which may be changed by a user. This creates a referential integrity problem which has increased development costs by over 50% on projects with published hard data on resource utilization.[3]

Functions of the Object Manager

For the purposes of this paper, we assume the Object Manager will be written in C++. It will accept OSQL messages and return C++ objects. The Object Manager must provide the following functionality:

- create, update, delete metadata, classes, and instances
- support OSQL and standard SQL queries
- enforce metarules

The Object Manager stores metadata in the database, is semantically complete, and has a full understanding of relationships between data. In contrast, a typical relational database stores only a portion of the schema in the database. The ability to maintain and evolve systems which store a complete set of metadata is enhanced over conventional systems. When an original developer moves on to other work, all the knowledge about the database does not depart with the person. Much of it remains in the database as a Meta-model, class schema, and *metarules* embodied in methods and the inheritance structure.

Building the HORA Meta-Model

The HORA Meta-model is created by building an entity-relationship semantic database model. The semantic model can be used to build an object template supporting methods and inheritance. Once the object template is in place, application classes can be created. The class schema can then be used as a template to populate the database with instances. Metaclasses, application classes, and class instances are all objects in the database. This recursive design enables powerful features not available in persistent object stores built without a semantic data model (dynamic schema evolution, for example).[15]

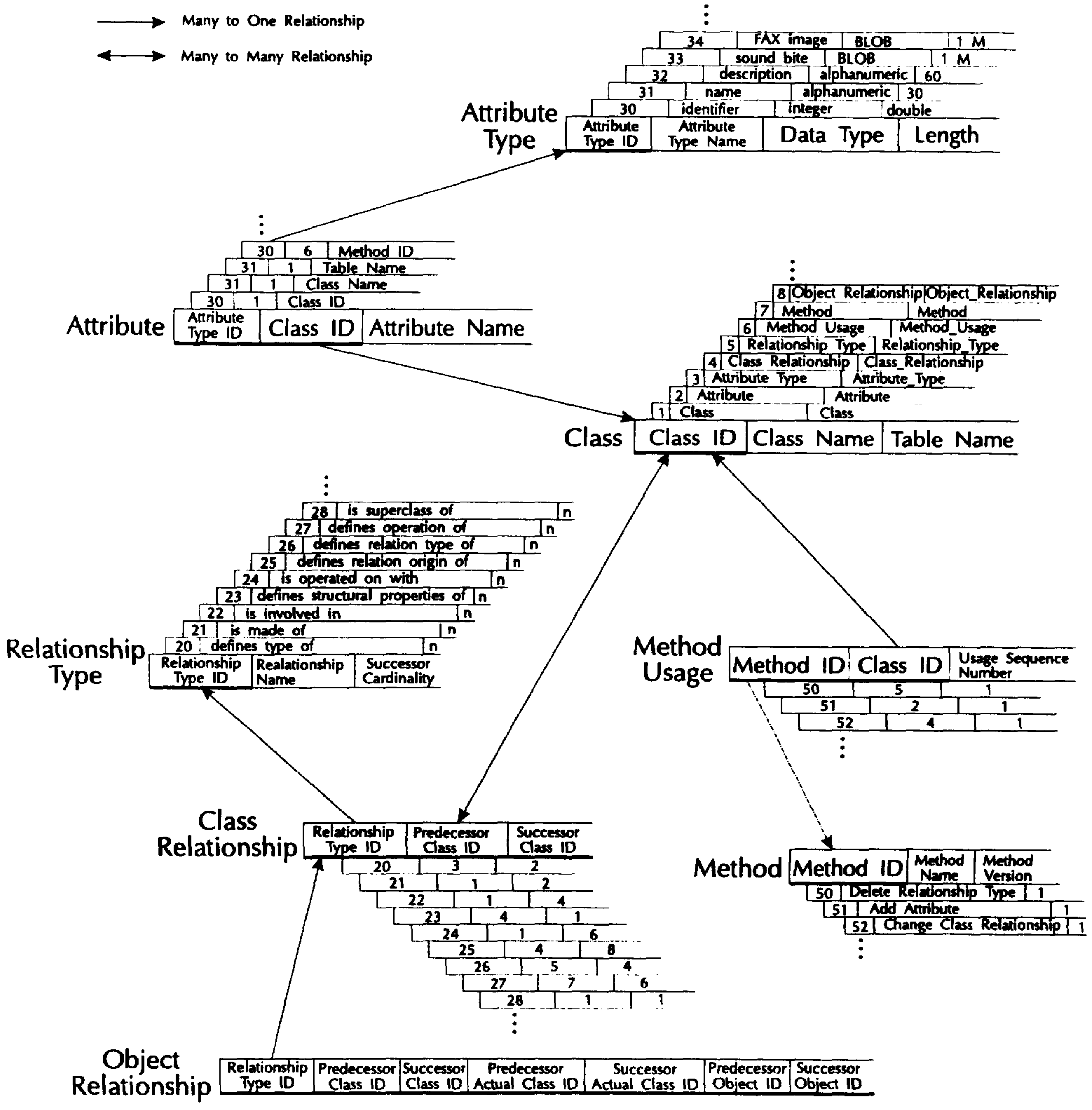
The Meta-model has been designed to be easily extensible to support distributed servers, versioning, and triggers. Space limitations make these topics beyond the scope of this paper.

The HORA Semantic Database Model

HORA is based on a semantic database model. Hull and King[16] provide a thorough review of the historical development of semantic models along with an extensive bibliography. All HORA objects are members of a class and all classes are constructed only from atomic attributes and relationships. This semantic data model is based on the work of Abrial[17] and directly supports the Entity Relationship Model of Chen.[18]

Atomic attributes can consist of any data type supported by the SQL Server or any data type which the Object Manager can support by using an algorithm to store data in the RDBMS. Strategies are available for storing Binary Large Objects (BLOBS) even in an RDBMS which does not directly support them. The term attribute will be used hereafter to mean atomic

Figure 2 - HORA Object Meta-Model Schema



Attribute Type

Attribute Type ID	Attribute Type Name	Data Type	Length
30	identifier	integer	double
31	name	alphanumeric	30
32	description	alphanumeric	60
33	sound bite	BLOB	1 M
34	FAX image	BLOB	1 M

Attribute

Attribute Type ID	Class ID	Attribute Name
30	1	Class ID
31	1	Class Name
31	1	Table Name
30	6	Method ID

Class

Class ID	Class Name	Table Name
1	Class	Class
2	Attribute	Attribute
3	Attribute Type	Attribute_Type
4	Class Relationship	Class_Relationship
5	Relationship Type	Relationship_Type
6	Method Usage	Method_Usage
7	Method	Method
8	Object Relationship	Object_Relationship

Relationship Type

Relationship Type ID	Realationship Name	Successor Cardinality
20	defines type of	n
21	is made of	n
22	is involved in	n
23	defines structural properties of	n
24	is operated on with	n
25	defines relation origin of	n
26	defines relation type of	n
27	defines operation of	n
28	is superclass of	n

Method Usage

Method ID	Class ID	Usage Sequence Number
50	5	1
51	2	1
52	4	1

Class Relationship

Relationship Type ID	Predecessor Class ID	Successor Class ID
20	3	2
21	1	2
22	1	4
23	4	1
24	1	6
25	4	8
26	5	4
27	7	6
28	1	1

Method

Method ID	Method Name	Method Version
50	Delete Relationship Type	1
51	Add Attribute	1
52	Change Class Relationship	1

Object Relationship

Relationship Type ID	Predecessor Class ID	Successor Class ID	Predecessor Actual Class ID	Successor Actual Class ID	Predecessor Object ID	Successor Object ID

The enthusiastic reader is encouraged to further populate the tables and complete the meta-model definition as an academic exercise.

attribute. Each class will have a single table which stores all attributes.

HORA links objects through the use of object identifiers in an Object Relationship Table (see Figures 2 and 4). Referential integrity is maintained by the Object Manager through enforcement of one simple rule. No object can be deleted until all its links to other objects are deleted from the Object Relationship Table.

The HORA Object Template

Reduced Instruction Object Semantics can be used to construct an object template by creating the following eight metadata tables (see Figure 2).

- Class Table
- Attribute Table
- Attribute Type Table
- Class Relationship Table
- Relationship Type Table
- Method Usage Table
- Method Table
- Object Relationship Table

Note that the notion of superclass is defined as a relationship type in Figure 2. Reuse is supported through this implementation of multiple inheritance.

The HORA Metaclass

When the Object Manager receives a request to open a new database, it open a database in the RDBMS and constructs the eight tables shown in Figure 2. Initially, there are no data in the rows of these tables. The Class Table is first populated with eight objects which represent the eight metatables (objects 1-8). In the Figures, object IDs are specified so as to aid understanding.

Attribute types are created in the Attribute Type Table (objects 30-34). Before the Object Manager can instantiate the HORA Metaclass, the "defines type of", "is made of", and "is superclass of" relationship types are needed. These are created in the relationship type table (objects 20, 21, 28).

The HORA Metaclass (the class named "Class") is made up of attributes and relationships by creating objects 21 and 22 in the Class Relationship table. Relationship 20 in the same table indicates that Attributes Types are related to Attribute Usage. The Object Manager must constrain Attribute Usage names to be unique to avoid name ambiguity in multiple inheritance specifications. Attribute Types, however, may be reused by the Object Manager.

The notion of superclasses is added by defining the Metaclass as a superclass of itself by adding relationship type 28 to the Class Relationship Table. The Object Manager supports standard specialization inheritance, i.e. subclasses consist of all attributes, relationships, and methods of all superclasses plus one or more additional attributes, relationships, or methods. Methods are added by putting relationship type 24, "is operated on with", in the Class Relationship Table. Methods for creating, updating, and deleting metadata can be added to the methods table. Polymorphism is supported by allowing a subclass to respecify a method previously defined in a superclass. In order to avoid ambiguity when searching the network of classes for methods, a Usage Sequence Number is defined in the Method Usage Table. When a message is sent to a class, if the method is not available in the class, the search for the method proceeds to superclasses. If multiple superclasses have the same method defined, the one with the lowest Usage Sequence Number is selected.

Instantiation of the HORA Metaclass bootstraps the system. The Object Manager can now dynamically determine the method of construction of all classes in the RDBMS by reading the metatables. With the instantiation of the appropriate methods, the Object Manager has the tools to easily change the Meta-model to support evolving standards such as those of the Object Management Group (OMG).[19,20,21] For example, a new Meta-model with additional features can be created by making it a subclass of the original Meta-model.

HORA Approach to Methods

From a theoretical standpoint, it is desirable to store method code in the database. In practice, distributed clients running in different languages on different platforms make this difficult. Also passing methods over the network can cause performance problems.

A workable solution is to store method names and version numbers in the database. When a client application connects to the database, it will read the metadata including method version information. The client software can then positively verify that the appropriate method versions have been previously linked into client code, prior to allowing a user to run the application. Method code under this scenario resides on application client workstations and not on the server.

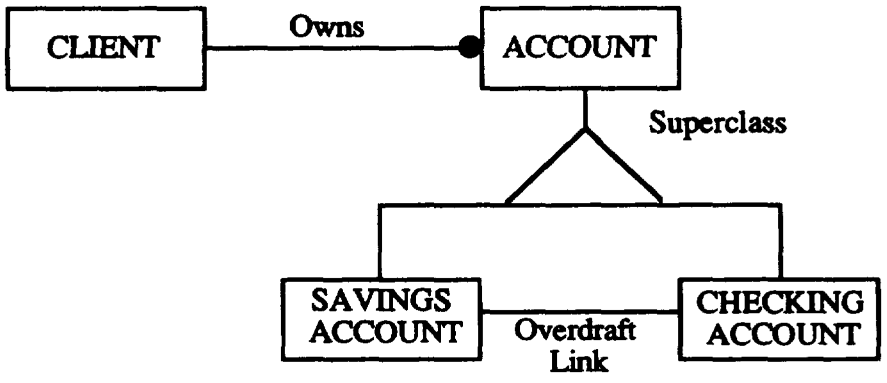
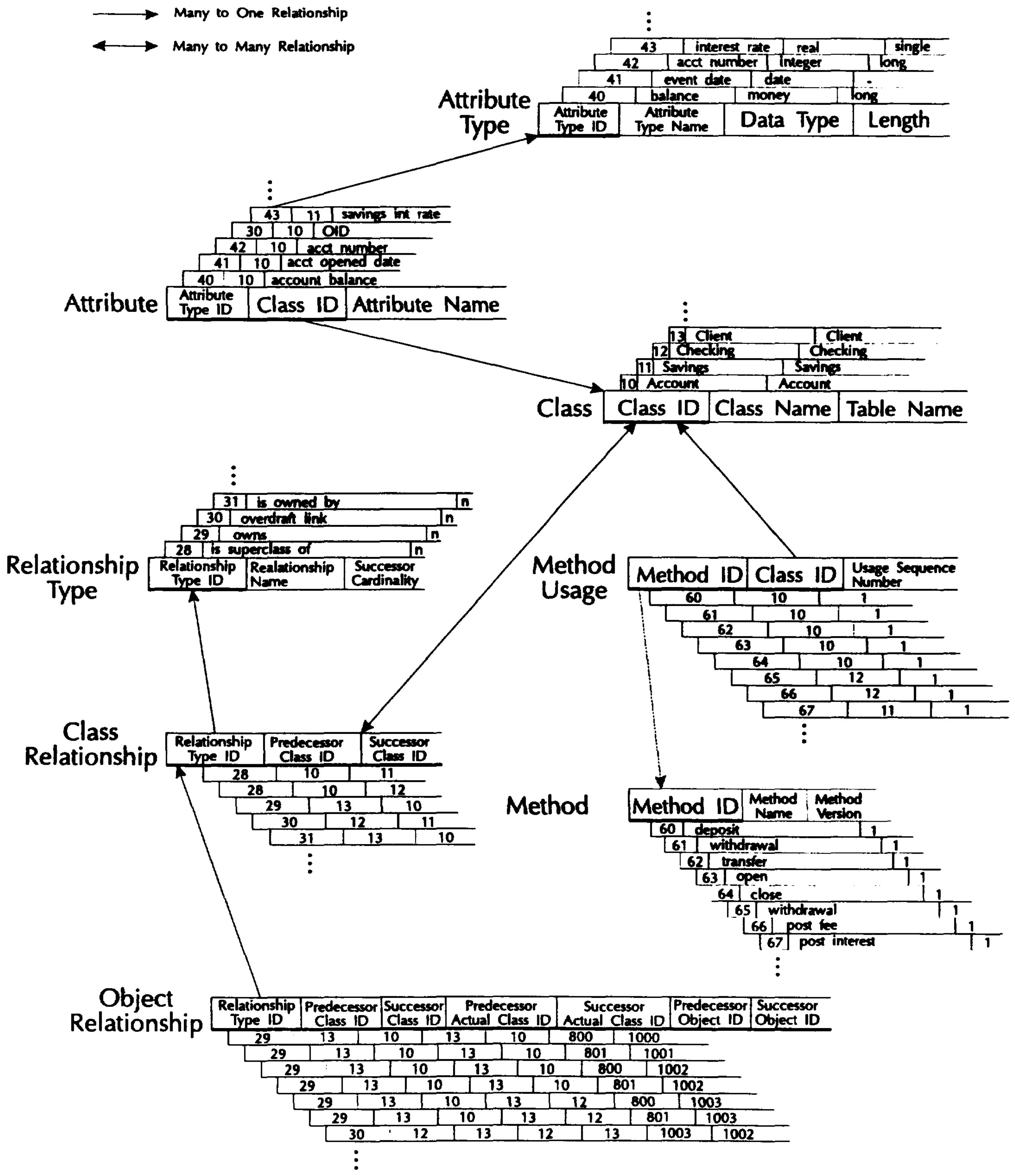


Figure 3

Figure 4 - Bank Accounts Class Schema



Attribute Type

Attribute Type ID	Attribute Type Name	Data Type	Length
40	balance	money	long
41	event date	date	-
42	acct number	integer	long
43	interest rate	real	single

Attribute

Attribute Type ID	Class ID	Attribute Name
40	10	account balance
41	10	acct opened date
42	10	acct number
30	10	OID
43	11	savings int rate

Class

Class ID	Class Name	Table Name
10	Account	Account
11	Savings	Savings
12	Checking	Checking
13	Client	Client

Relationship Type

Relationship Type ID	Realationship Name	Successor Cardinality
28	is superclass of	n
29	owns	n
30	overdraft link	n
31	is owned by	n

Method Usage

Method ID	Class ID	Usage Sequence Number
60	10	1
61	10	1
62	10	1
63	10	1
64	10	1
65	12	1
66	12	1
67	11	1

Class Relationship

Relationship Type ID	Predecessor Class ID	Successor Class ID
28	10	11
28	10	12
29	13	10
30	12	11
31	13	10

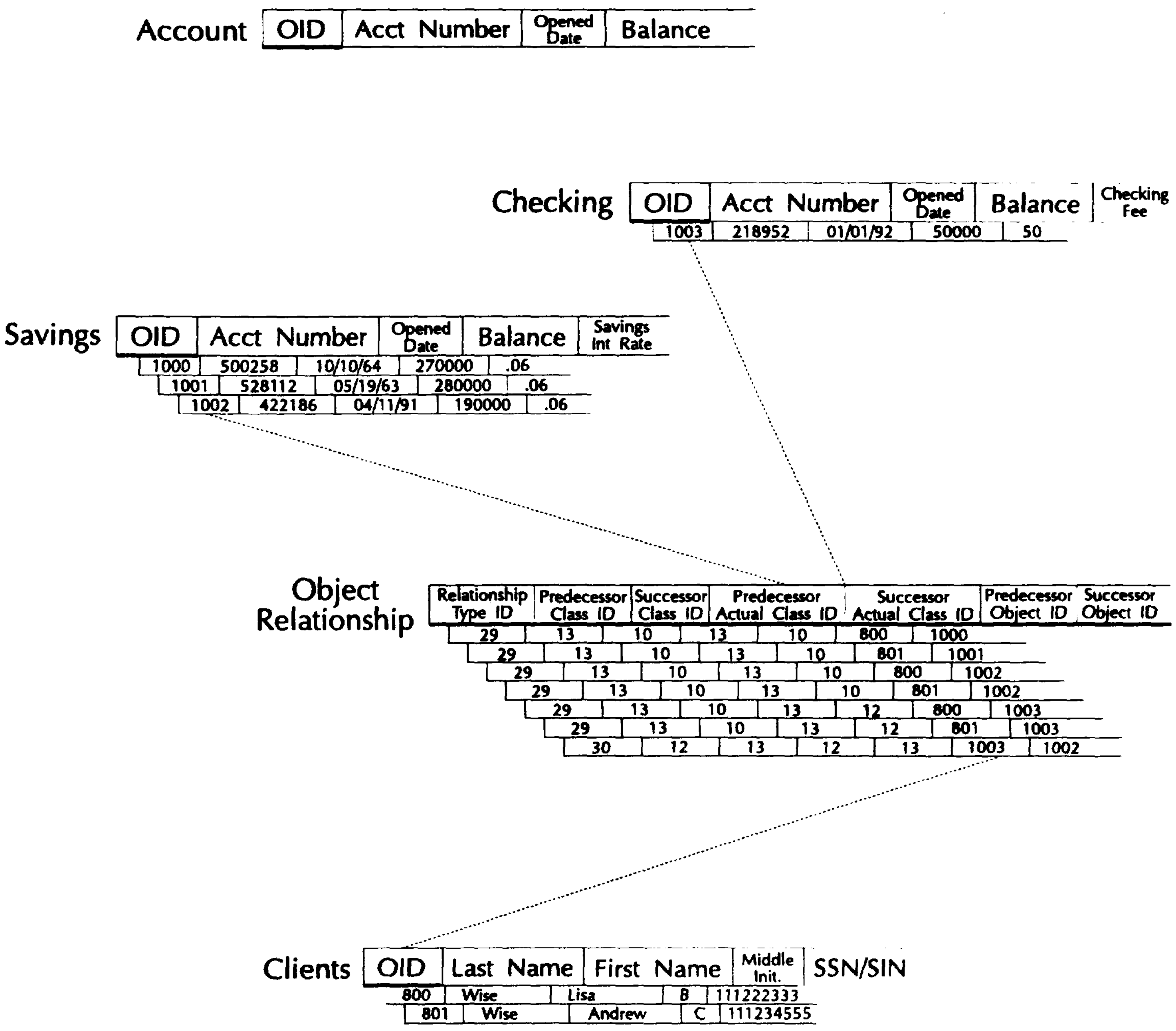
Method

Method ID	Method Name	Method Version
60	deposit	1
61	withdrawal	1
62	transfer	1
63	open	1
64	close	1
65	withdrawal	1
66	post fee	1
67	post interest	1

Object Relationship

Relationship Type ID	Predecessor Class ID	Successor Class ID	Predecessor Actual Class ID	Successor Actual Class ID	Predecessor Object ID	Successor Object ID
29	13	10	13	10	800	1000
29	13	10	13	10	801	1001
29	13	10	13	10	800	1002
29	13	10	13	10	801	1002
29	13	10	13	12	800	1003
29	13	10	13	12	801	1003
30	12	13	12	13	1003	1002

Figure 5 - Bank Account Instances



Account

OID	Acct Number	Opened Date	Balance

Checking

OID	Acct Number	Opened Date	Balance	Checking Fee
1003	218952	01/01/92	50000	50

Savings

OID	Acct Number	Opened Date	Balance	Savings Int Rate
1000	500258	10/10/64	270000	.06
1001	528112	05/19/63	280000	.06
1002	422186	04/11/91	190000	.06

Object Relationship

Relationship Type ID	Predecessor Class ID	Successor Class ID	Predecessor Actual Class ID	Successor Actual Class ID	Predecessor Object ID	Successor Object ID
29	13	10	13	10	800	1000
29	13	10	13	10	801	1001
29	13	10	13	10	800	1002
29	13	10	13	10	801	1002
29	13	10	13	12	800	1003
29	13	10	13	12	801	1003
30	12	13	12	13	1003	1002

Clients

OID	Last Name	First Name	Middle Init.	SSN/SIN
800	Wise	Lisa	B	111222333
801	Wise	Andrew	C	111234555

Bank Accounts - A Simple Example of the HORA in Action

Consider the object diagram in Figure 3 (Rumbaugh notation).

The example consists of two object types, Clients and Accounts. Account will be used as a superclass of Savings Account and Checking Account. Clients will own accounts and Checking Accounts will be related to Savings Accounts through an Overdraft Link. This will allow the system to withdraw money from a Savings Account in the case of a Checking Account overdraft. Attributes of these objects are as specified below:

```
CLASS Client
        (Last_Name, First_Name, Middle_Initial,
         SSN-SIN,
        RELATIONSHIPS (Owns Account))

CLASS Account
        (Account_Number, Opened_Date, Balance,
        METHODS (Open, Close, Deposit, Withdraw,
                 Transfer))

CLASS Savings_Account
        (Interest_Rate,
        SUPERCLASSES (Account),
        METHODS (Post_Interest))

CLASS Checking Account
        (Checking_Fee,
        RELATIONSHIPS (Overdraft_Link
                Savings_Account),
        SUPERCLASSES (Account),
        METHODS (Withdraw, Post_Fee))
```

This simple example illustrates the power of object technology. The class CHECKING ACCOUNT has a Withdraw method that overrides the method of the same name in the superclass ACCOUNT. This method may be written to transfer money from a savings account in the event of overdraft using the Overdraft_Link relationship. The Transfer method in ACCOUNT may use Withdraw (from itself) and Deposit (to another account).

Consider a transfer from a checking account to another account. CHECKING ACCOUNT does not have a Transfer method, so the Transfer method of superclass ACCOUNT is executed. This sends a Withdraw message to the checking account. If the checking account has insufficient funds, the Withdraw method sends a Transfer message to the attached savings account, and so forth. A complex series of events will occur automatically using only ACCOUNT Withdraw, Deposit, and Transfer methods and the modified Withdraw method in CHECKING ACCOUNT.

Development and maintenance of the code for these account transactions is significantly reduced through reuse of methods and through the built in thread of execution of these methods supported by inheritance. In a real banking system, it will be necessary to consider security, transaction logs and other factors. Reduction in code required will be much greater than in this simple example.

Opening A Database

To open a new application database, the application must send the Object Manager an OSQL command to create a database. The Object Manager will respond by opening a database on the SQL server and creating the metadata tables described previously.

Using a proposed OSQL standard which will eventually be superceded by ANSI SQL III syntax, the call would look like as follows with the application code specifying the database name and mode of operation (read/write). The Object Manager returns a cursor and connection ID for use in the next call to the database.

```
Object_Manager::open(cursor, conn_ID, database,
                     mode);
```

Creating The Class Schema

Once the metatables are created, the application must send OSQL commands to create the class schema.

```
Object_Manager::osql(cursor, OSQL_statement,
                     length, conn_ID);
```

where,

```
OSQL_statement= CREATE CLASS Account
        (Account_Number integer 12 INDEX
                REQUIRED,
        Opened_Date date,
        Balance money 15.2,
        METHODS (Open 1, Close 1, Deposit 1,
                Withdraw 1, Transfer 1))

OSQL_statement= CREATE CLASS Client
        (Last_Name string 30 INDEX   REQUIRED,
                First_Name string 30,
        Middle_Initial string 1, SSN_SIN integer
                9,
        RELATIONSHIPS (Owns Account))

OSQL_statement = CREATE CLASS Savings_Account
        (Interest_Rate 4.2,
        METHODS (Post_Interest 1),
        SUPERCLASSES (Account))

OSQL_statement = CREATE CLASS
        Checking_Account (Checking_Fee 6.2,
        RELATIONSHIPS (Overdraft_Link
                      Savings_Account),
        METHODS (Post_Fee 1, Withdraw 1),
        SUPERCLASSES (Account))
```

Upon receipt of these commands, the Object Manager will set up the class schema in Figure 4 and generate the object tables of Figure 5 without instances. All that remains is to instruct the Object Manager to create the instances.

Creating Instances

To enter the first account,

```
OSQL_statement = CREATE OBJECT OF CLASS
        Savings_Account (Account_Number 500258,
        Opened_Date 10-10-64, Balance 2200.00,
        Interest_Rate 0.06)
```

A cursor will be returned with Account_OID, a pointer to the object just created. This is used to connect a Client instance to the Savings Account instance.

```
OSQL_statement = CREATE OBJECT OF CLASS Client
        (Last_Name "Wise", First_Name "Lisa",
        Middle_Initial "T",   SSN_SIN 111222333,
        RELATIONSHIPS (Owns 1000, 1002, 1003))
```

The creation of the remaining instances in Figure 5 proceeds in a similar fashion. Classes and instances can be modified and deleted with similar OSQL commands.

Optimizing the Database for Performance

When building a database application and bringing it into production, it is sometimes necessary to improve performance of the system by denormalizing the database. This invariably reduces the flexibility of the system and generates additional maintenance overhead in a conventional relational application. In the object-oriented environment, it can also destroy the integrity of the object system.

There is one extension to the object system which may improve performance for some applications. For one to many relationships, a foreign key OID may be inserted as an attribute in the instance tables. This will allow relational joins to be accomplished without going through the Object Relationship Table. The Object Relationship Table must still be maintained to enable reverse pointers which support referential integrity. This process can be automated through proper design of the Object Manager.

In our example above, this would result in an addition column in the Checking Account table which would contain the OID's of the related Savings Account. This would yield performance gains on transactions that require simultaneous access to both checking and savings accounts. For more complex object hierarchies, improvements could be significant.

Conclusion

The HORA methodology for connecting new object-oriented applications to relational datastores has been described. It supports full object-oriented functionality while allowing traditional applications read access to the database and is easily extensible to distributed RDBMS servers and versioned databases. The object schema, including the Meta-model, can dynamically evolve to support complex object structures and emerging ISO, ANSI, and OMG object standards.

Bibliography

1 Atkinson, M. et al. The Object-Oriented Database Systems Manifesto. In *Deductive and Object-Oriented Databases*. Elsevier Science Publishers, 1990.

2 Committee for Advanced DBMS Function (Stonebraker, M., et al.). Third Generation Database System Manifesto. *ACM SIGMOD Record*, Sep 1990.

3 Gardner, J., Sutherland, J.V. *Report on Buyer Furnished Equipment Development History and Level of Effort*. Object Databases, Cambridge, Mass., November, 1990.

4 *G-BASE/SQL Product Brief*. Object Databases, Cambridge, 1990.

5 Lyngback, Peter. *OSQL: A Language for Object Databases*. Hewlett Packard Technical Report HPL-DTD-91-4, Jan 15, 1991.

6 Sutherland, J.V. *Graphael-Boeing Working Paper: Boeing Aerospace APF Project*. Graphael, Inc., Waltham, 1989.

7 Rymer, J.R. Guiness P.A.'s Buyer Furnished Equipment: An Object-Oriented Application Case Study. *First Class: The Object Management Group Newsletter*, Nov/Dec 1991, pp. 20-21.

8 Rymer, J.R. *Object Orientation 1991: Toward Commercial Reality*. Patricia Seybold's Office Computing Group Special Research Report, 1991, pp. 280-286.

9 Melton, Jim. (ISO/ANSI) Working Draft Database Language SQL (SQL3). *ANSI X3H2-92-155 DBL CNB-003*, July 1992.

10 Rumbaugh, J. et al. *Object Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, 1991.

11 Barthes, John-Paul A., Vayssade, M., Znamierovska, M. Property Driven Databases. In *Proceedings of 6th IJCAI*, Tokyo. AAAI, 1979.

12 Date, C.J. *A Guide to the SQL Standard, Second Edition*. Addison-Wesley, Reading, Mass., 1989.

13 Codd, E. F. *The Relational Model for Database Management, Version 2*. Reading: Addison-Wesley, 1990.

14 Khoshafian, Setrag N., Copeland, George P. Object Identity. In *Readings In Object-Oriented Database Systems*. Zdonik, S.B., Maier, D. (Eds.) San Mateo, Morgan Kaufmann, 1990.

15 Sutherland, J.V. Is a Persistent Object Store a Database? In *OOPSLA '89 Proceedings*, New Orleans, October 1-6. ACM, New York, 1989, pp. 499--500.

16 Hull, Richard, King, Roger. Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys* 19:3:201-260.

17 Abrial, J.R. Data Semantics. In *Data Base Management*. Klimbie JW, Koffeman KL (Eds.) North-Holland, 1974, pp. 1-59.

18 Chen, P.P. The Entity-Relationship Model--Toward a Unified View of Data. In *Readings in Database Systems*. Stonebraker, M. (Ed) Morgan Kaufmann, San Mateo, 1988, pp. 374-387.

19 Osher, H.M. Software Without Walls. *BYTE* 17:3 (Mar), 1992, pp. 122-128.

20 Object Model Task Force (OMTF). The *OMG Object Model*. Framingham: Object Management Group (OMG), draft 1 Mar 1992.

21 Digital Equipment Corp. et al. *The Common Object Request Broker*. Framingham: Object Management Group (OMG), 1992.