An Approach to Object Sharing in Distributed Database Systems[1]

Peter Lyngbaek
Dennis McLeod

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0782

ABSTRACT

This paper describes DODM, a simple model for object sharing in distributed database systems. The model provides a small set of operations for object definition, manipulation, and retrieval in a distributed environment. Relationships among objects can be established across database boundaries, objects are relocatable within the distributed environment, and mechanisms are provided for object sharing among individual databases. An object naming convention supports location transparent object references; that is, objects can be referenced by user-defined names rather than by address. The primitive operations introduced can be used as the basis for the specification and stepwise development of database models and database systems of increasing complexity. An example is provided to illustrate the use of DODM in the design of a distributed database system supporting a semantically expressive database model.

1. Introduction

Distributed computing systems are becoming increasingly common. This trend is largely caused by the decreasing cost of hardware: not only are powerful personal computers becoming so inexpensive today that individuals can afford them for personal use, but the cost of computer networks that enable computer systems to exchange information at a very high rate is decreasing drastically. Decentralization overcomes many of the limitations and deficiencies of centralized systems. A network of computers simply provides a higher level of performance, availability, reliability, fault tolerance, and security than a centralized computer system. In addition to the technical advantages that make decentralized systems feasible, social attitudes tend to indicate that a collection of smaller, autonomous computer systems are preferable to large central systems.

The growing popularity of distributed computing establishes a need for mechanisms that allow individual users to communicate with each other and share both hardware and software resources. Individual users also need access to the growing number of "public" databases, which contain a variety of information such as grocery prices, the values of stocks, and the histories of bank accounts. Of course, sharing mechanisms decrease the autonomy of the components of the distributed environment, and affect the performance, availability, reliability, fault tolerance, and security of the total system. Sharing and communication mechanisms alsc introduce data transmission and naming problems.

Most current approaches to distributed database management system design fail to adequately address issues concerning location transparency (the ability to reference data by name rather than by address), logical decentralization, catalog management, and the uniform handling of meta-data and user-data. Logically centralized database systems [Rothnie 80, Stonebraker 77, Andler 82] provide the users with a single integrated database schema describing all the data in the physically centralized or distributed environment. Recent research has also resulted in approaches to support the integration of heterogeneous as well as homogeneous (pre-existing) databases [Motro 81, Smith 81, Litwin 81, Kimbleton 79]. However, a critical remaining problem is accommodating information sharing among individual, autonomous databases. Finally, existing distributed database system architectures that emphasize the autonomy of the individual databases [Heimbigner 82, Williams 81, Tsichritzis 82] require centralized or complex catalog management.

The aim of the research described in this paper is to define a simple model for object sharing in distributed database systems. This is done by stepwise development of a series of object-oriented models. First, a simple model called *ODM* (for *object-oriented database model*) is defined. ODM provides a

[1]This research was supported, in part, by the Joint Services Electronics Program through the Air Force Office of Scientific Research under contract F49620-81-C-0070.

user with the basic primitives for object definition, manipulation, and retrieval; it is straightforward to implement, but it lacks semantic expressiveness as well as mechanisms for integrity control and protection. Next, ODM is extended to provide object definition, manipulation, and retrieval facilities for a distributed environment. The distributed version of ODM, called *DODM*, supports object sharing among individual databases, and allows relationships to be established between objects in different databases. Moreover, DODM provides location transparency, and can be implemented without introducing any central data structures or authorities.

The next logical step is the introduction of semantic database models for centralized and distributed databases. These high-level models, based on the primitives of ODM and DODM, support types/subtypes for object classification [Smith 77] (generalization hierarchies) and inter-object mappings (attributes).

The framework for the research described in this paper is a personal information management environment called INFOBASE [McLeod 83a], currently being developed in the Computer Science Department of the University of Southern California. INFOBASE is intended to provide information management facilities to support a wide range of personal workstation applications, including data management activities of a professional, manager, or home computer user. INFOBASE is also intended to support the information management needs of engineering applications, including software engineering, and CAD/VLSI design [McLeod 83b].

A collection of INFOBASE workstations is modelled as a logical network of components called stations [Lyngbaek 82]. Each station has a unique station name, and is operated and owned by a single user. This convention is adopted to allow all resource sharing to be defined at the same level. A user is simply identified with the station he/she owns. In this way, resource sharing between two users can be treated as sharing between their two stations. Several INFOBASE stations may be grouped together at the same physical node in a computer network, but that is not reflected in this model, which describes the distributed environment as a logical network of identical stations. The stations in the network need not implement the same database model; they are only required to be identical from a network point of view, i.e., they must all provide the same network interface. Thus, existing databases and information systems may be part of the network, if they are accommodated as virtual INFOBASE stations.

This paper focuses exclusively on ODM and DODM. Current research is in progress to explore object sharing in distributed databases that are modelled by higher-level semantic database models. Section 2 defines ODM by describing its primitive operations and its implementation. Section 3 extends ODM to cover distributed database modelling. A naming convention is introduced for DODM, its operations are described, and the implementation is discussed. Section 4 contains a distributed database example; a simple distributed database system based on a semantic database model is implemented on top of DODM. Finally, in Section 5 some concluding remarks are provided.

2. ODM: An Object-Oriented Database Model

ODM is a very simple object-oriented database model with a straightforward implementation. The main purpose of ODM is to provide a basic framework for object-oriented database models and database systems. ODM is based on a small number of simple concepts, that can be used as a tool for stepwise development of database models of increasing complexity and levels of abstraction.

2.1. The Modelling Elements of ODM

A database is modelled in ODM as a collection of *objects* and *relationships*. Objects correspond to concepts that have an associated meaning, e.g., the space shuttle Columbia, the Queen of Denmark, and the name "John Smith". A relationship is an association among three objects (say {x, y, z}). Relationships are used to define a mapping between two objects; that is, object y in the relationship {x, y, z} is a mapping from object x to object z. Since relationships can be established between any three objects in a database, it is possible to model relations that are one-to-one, one-to-many, many-to-one, and many-to-many.

Objects are divided into the three categories:

- *Descriptor objects* are atomic strings of characters. They are displayable and serve as symbolic identifiers in the database. The character string that constitutes a given descriptor object is called the *object identifier*, which is unique (since there is a single instance of each descriptor object in ODM).

- *Behavioral objects* embody database operations [Brodie 81, King 82] and are executable. ODM includes behavioral objects to support data definition, manipulation, and retrieval; user-defined behavioral objects are also supported. Behavioral objects are uniquely identified by object identifiers (atomic character strings), that are displayable and serve to reference and invoke the objects.

- *Abstract objects* are objects that are not descriptor objects or behavioral objects. They are neither displayable nor executable, but can only be described in terms of their relationships with other objects.

Abstract objects are introduced to support objects with no single meaningful name. For example, a given abstract object may represent the person Bill Connors. This abstract object does not itself contain any descriptive information about the person it denotes, Rather, such information is modelled by descriptor objects, e.g., the person name 'Bill Connors' and the social security number '234-54-2397', which are related to the abstract object via appropriate mappings ('has name' and 'has social security number'). When referring to an abstract object in this paper, an unquoted mnemonic name is used, e.g., Queen of Denmark; descriptor and behavioral objects are referenced by their object identifier (e.g., 'Ridge Zinfandel').

2.2. The ODM Data Definition and Manipulation Language

The ODM data definition and manipulation language is described here as a set of primitive operations that are embedded into a host programming language. The purpose here is not to propose a specific approach to host language embedding of data manipulation operations, but rather to define a set of primitive building blocks to support (among other things) a high-level, interactive user interface through which unsophisticated users can communicate with the database.

The host language must support the data types *objectid*, *objectref*, and *set of objectref*, and the usual set operations. Variables can then be declared (in programs written in the host language) to be of these types. Values of variables of type objectid are object identifiers. Values of variables of type objectref are references to objects; variables of this type are used as "handles" on objects in the database. At any given time, several variables (in the same or different programs) can denote the same object.

ODM contains eight primitive operations. These allow a user to add new descriptor, behavioral, and abstract objects to a database, to remove existing objects from a database, to test whether or not a given object reference denotes an existing database object, to create and delete relationships between existing database objects, to retrieve objects from the database, and to return (print) their unique object identifiers. A detailed description of the operations together with examples of their usage is given below. A Pascal-like language is used here to illustrate the use of the operations.

2.2.1. CREATE([id: objectid]): objectref

The CREATE operation creates a new object, adds it to the database, and returns a reference to it. If no identifier is specified, the new object is an abstract object, which can only be accessed via the reference returned. If an identifier is given that is not the object identifier of an existing database object, the new object is a descriptor object with the specified object identifier. An execution error occurs if the specified identifier denotes an existing database object.

Suppose that john, mary, mail, memo, person1, and person2 are all variables of type objectref; then the following operations create seven new objects:

```
john := CREATE('John Smith');
mary := CREATE('Mary Brown');
mail := CREATE('Incoming mail');
memo :=
  CREATE('The committee is meeting at 3 p.m.')
has-name := CREATE('Has name')
person1 := CREATE();
person2 := CREATE();
```

The objects denoted by person1 and person2 are abstract objects, while the other objects are descriptors.

2.2.2. ISOBJECT(o: objectref): boolean

The ISOBJECT operation returns the value true if the specified object reference denotes an existing object; otherwise it returns the value false. After the execution of the CREATE operations listed above, the operations ISOBJECT('Has name') and ISOBJECT(person1) would return the value true, but the operation ISOBJECT('7') would return the value false.

2.2.3. DELETE(o: objectref)

The DELETE operation simply removes a given object from the database. If the specified object participates in relationships, those relationships are deleted (see the DETACH operation below). After the execution of the CREATE operations listed above, the operation DELETE(john) would remove the object 'John Smith' from the database.

2.2.4. RELATE(d, m, r: objectref)

The RELATE operation relates the three objects specified. The first parameter, d, is called the domain object; the second parameter, m, is called the map object; and the third parameter, r, is called the range object. Here, d is related to r via m, d is in the domain of m, and r is in the range of m. In ODM, any three objects can be related in this way. Thus, it is the user's responsibility to avoid the creation of meaningless relationships (this problem is further discussed below).

The following example illustrates the use of the RELATE operation (assuming that the CREATE operations listed above have been executed):

```
RELATE(person1, has-name, john)
RELATE(person2, has-name, mary)
RELATE(person1, mail, memo);
RELATE(person2, mail, memo);
```

The expression $x(y) = z$ is a boolean expression associated with three objects denoted by x, y, and z. If y is related to z via x, then $x(y) = z$ is true; otherwise $x(y) = z$ is false. Note that the RELATE operation sets the value of $m(d) = r$ to true.

2.2.5. DETACH(d, m, r: objectref)

If the object denoted by d is related to the object denoted by r via the object denoted by m, the DETACH operation deletes that relationship; i.e., after the execution of the DETACH operation, the object denoted by d is no longer related to the object denoted by r via the object denoted by m. The DETACH operation has no effect if the specified objects are not related. Note that the DETACH operation sets the value of $m(d) = r$ to false, but it does not effect the following expressions: $m(r) = d$, $d(m) = r$, $d(r) = m$, $r(d) = m$, and $r(m) = d$.

After the example CREATE and RELATE operations above have been executed, the operation DETACH(person1, mail, memo) would cause the abstract object modelling John Smith not to be related to the object 'The committee is meeting at 3 p.m.' via the (mapping) object 'Incoming mail'. The operation DETACH(memo, person1, mail) would have no effect, since the object 'The committee is meeting at 3 p.m.' is not related to the object 'Incoming mail' via John Smith.

2.2.6. FIND(d, m, r: objectref): set of objectref

The parameters to the FIND operation specifies a query. The FIND operation returns the set of objects satisfying that query. Each parameter is either a question mark "?" or an object reference. The question mark denotes the objects in question. The first parameter corresponds to domain objects, the second to map objects, and the third to range objects. Thus, if the first parameter is a question mark and the two other parameters are object references, the query asks for all the domain objects that have been related to the specified range object via the specified map object.

The don't care symbol "*" is a special object reference. When used as a parameter to the FIND operation, it means that the corresponding object is unspecified. In other words, it may be replaced by any object in the database. Therefore, if the first parameter is a question mark, the second a don't care symbol, and the third an object reference, the query asks for all the domain objects that have been related to the specified range object via some map object. In the following examples describing the FIND operation, OBJECTS denotes the set of all objects in an ODM database:

- FIND(?, m, r) returns all the objects that have been related to r via m: FIND(?, m, r) = {d in OBJECTS| m(d) = r}.

- FIND(d, *, ?) returns all the objects that d has been related to via some map object: FIND(d, *, ?) = {r in OBJECTS| m(d) = r for some m in OBJECTS}.

- FIND(*, *, ?) returns all the range objects in the database: FIND(*, *, ?) = {r in OBJECTS| m(d) = r for some d and m in OBJECTS}.

- FIND(?, m, ?) returns all the objects in the domain of m and all the objects in the range of m: FIND(?, m, ?) = UNION(FIND(?, m, *), FIND(*, m, ?)).

- FIND(?, ?, ?) returns all the objects that participate in a relationship, that is, all the domain objects, map objects, and range objects in the database: FIND(?, ?, ?) = UNION(FIND(?, *, *), FIND(*, ?, *), FIND(*, *, ?)).

Note that at least one question mark must be specified in an invocation of the FIND operation.

After the execution of the CREATE and RELATE operations listed above, the operation FIND(person1, *, ?) would return the set {'John Smith', 'The committee is meeting at 3 p.m.'}, viz., the set of all objects to which the object modelling John Smith is related via some map object.

2.2.7. PRINT(o: objectref)

The PRINT operation prints the unique object identifier for the specified object, which must be either a descriptor or a behavioral object. An execution error occurs if the specified object is an abstract object. For example, the operation PRINT(memo) outputs the string 'The Committee is meeting at 3 p.m.', and the operation PRINT('John') outputs the string 'John'.

2.2.8. DEFINE(operation definition)

The DEFINE operation creates a new behavioral object. The new operation is defined in the host programming language in terms of previously defined operations, as a procedure or function is defined in a Pascal program. The name of the new operation is the object identifier of the behavioral object and must therefore be unique. While the specification of behavior is a very important issue [Brodie 81, King 83], it is beyond the scope of this paper to directly address it. However, figure 2-1 illustrates the definition of a new operation called RENAME, the purpose of which is to change the object identifier of a given object.

```
DEFINE(

RENAME(oldname: objectref; newname: objectid)
var d, n, m, r: objectref;
    domains, maps, ranges: set of objectref
begin
   if ISOBJECT(oldname) and
      not ISOBJECT(newname) then
   begin
      n := CREATE(newname);
      maps := FIND(oldname, ?, *);
      for each m in maps do
         ranges := FIND(oldname, m, ?);
         for each r in ranges do
            RELATE(newname, m, r)
         end
      end;
      domains := FIND(?, oldname, *);
      for each d in domains do
         ranges := FIND(d, oldname, ?);
         for each r in ranges do
            RELATE(d, newname, r)
         end
      end;
      domains := FIND(?, *, oldname);
      for each d in domains do
         maps := FIND(d, ?, oldname);
         for each m in maps do
            RELATE(d, m, newname)
         end
      end
      DELETE(oldname)
   end
end

)
```

Figure 2-1: Definition of the RENAME operation

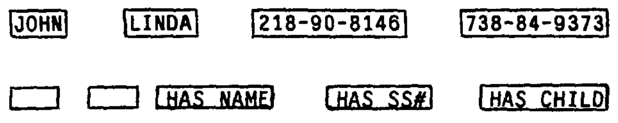
2.3. Graphical Representation

Objects and their relationships can be illustrated in a graph. A descriptor or behavioral object is shown as a box labeled with its unique object identifier, and an abstract object is shown as an empty box. A relationship is shown as a labeled edge connecting two objects. Suppose object x is related to object z via object y. Then an edge labeled y originates in the box labeled x and terminates in the box labeled z. Figure 2-2 specifies a sequence of operations and the corresponding graph.

```
var john, daughter, linda, ss1, ss2, person1,
    person2: objectref

john      := CREATE('JOHN');
linda     := CREATE('LINDA');
ss1       := CREATE('218-90-8146');
ss2       := CREATE('738-84-9373');
person1   := CREATE();
person2   := CREATE();
has-name  := CREATE('HAS NAME');
has-ss    := CREATE('HAS SS#');
has-child := CREATE('HAS CHILD');
```

These operations create the objects shown below:



```
RELATE(person1, has-name, john);
RELATE(person1, has-ss, ss1);
RELATE(person2, has-name, linda);
RELATE(person2, has-ss, ss2);
RELATE(person1, has-child, person2);
```

These operations create the relationships shown below:

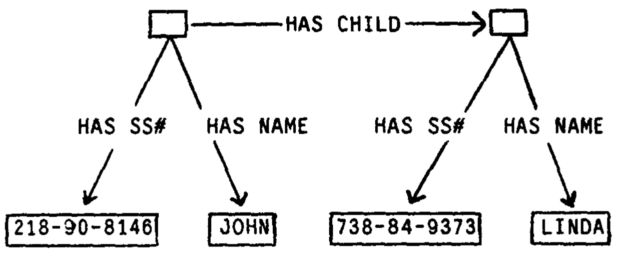


Figure 2-2: Graphical Representation

2.4. Implementation of ODM

A straightforward implementation of ODM is possible using existing database technology. This section describes a simple prototype that has been built using the INGRES relational database management system [Stonebraker 76], running the UNIX[2] operating system.

Associated with each object is a system-dependent unique *object key*, which is implicitly assigned when the object is created. Object keys are internal references to the objects in the database; they are neither displayable nor modifiable. Object identifiers of descriptor and behavioral objects are already unique, but using them as keys directly would cause a problem, since they can be of arbitrary length. One approach to solve this problem is to convert the object identifiers to object keys via a table (possibly implemented via a hash function, B-tree, etc.).

The prototype uses an OBJECTS relation and a RELATIONSHIPS relation. Each object in the database is described by a tuple in the OBJECTS relation. An object is described by the following information:

- The unique object key, if the object is an abstract object.
- The unique object key together with the unique object identifier, if the object is a descriptor object.
- The unique object key together with the unique object identifier and the executable code, if the object is a behavioral object.

The OBJECTS relation has the three attributes: KEY, IDENTIFIER, and OPERATION. Note that the object identifier of a given object is only stored once.

Each relationship in the database is described by a tuple in the RELATIONSHIPS relation. The RELATIONSHIPS relation has three attributes: one attribute (DOMAIN) for the domain object keys, one (MAP) for the map object keys, and one (RANGE) for the range object keys.

The ODM operations are implemented as a collection of separately compiled EQUEL/C programs. A user-friendly interface to the ODM system is provided by a command interpreter that guides the user through the process of selecting, specifying, and invoking ODM operations. In the implementation of ODM, the CREATE and RELATE operations become tuple insertions in the OBJECTS and RELATIONSHIPS relations, respectively. The DELETE and DETACH operations become tuple deletions from the OBJECTS and RELATIONSHIPS relations, respectively. The FIND and ISOBJECT operations become tuple selections followed by projections.

The DEFINE operation is not implemented by the current prototype. It is a major task to integrate the proposed primitives with a host programming language on top of a relational database system. Furthermore, due to limitations of UNIX, run-time invocations of the user-defined ODM operations is a nontrivial feature to implement.

If a relational database management system were not available, a simple ad-hoc implementation can be built on top of a file system. Such an implementation would typically maintain two files, an OBJECTS file describing the objects in the database and a RELATIONSHIPS file describing the relationships in the database. In order to provide fast access to the information in the OBJECTS and RELATIONSHIPS files, four B-tree indices would suffice. One of the B-trees maintains an index, based on object keys, to the OBJECTS file. The other three B-trees describe the RELATIONSHIPS file by providing indices on the key values of the domain objects, the map objects, and range objects, respectively.

[2]UNIX is a trademark of Bell Laboratories.

2.5. Limitations of ODM

ODM is based on the two essential concepts of objects and relationships. All data in an ODM database is treated uniformly as objects, and relationships between these objects allow semantic properties to be modelled[3]. In addition, the implementation of ODM is straightforward. However, such a simple database model is not appropriate for a non-expert database user. It is too easy to create meaningless relationships and operations, and no built-in mechanisms for data protection and integrity control are provided. However, the primitive operations of ODM can be used as the basic building blocks in the design and implementation of higher-level database systems based on semantic database models.

3. DODM: An Object-Oriented Database Model for Distributed Databases

DODM is a simple extension of ODM, which provides object definition, manipulation, and retrieval capabilities in an environment of distributed databases. The distributed databases can be thought of as a logical network of communicating databases. In DODM, relationships among objects can be established across database boundaries, objects may be copied or moved from one database to another, and mechanisms are provided for object sharing among individual databases.

3.1. Objects in a Distributed Environment

The single object instance rule of DODM states that in a network of databases, there is exactly one instance of a given object. However, objects stored in different databases may be identical; that is, they may have the same content even though they are considered to be instances of different objects. Such objects are said to be *equivalent*. When an object in a given database is copied to another database, the copy will be a completely new object owned by the database to which it is copied, but the copy and the original object are equivalent.

In the distributed environment, a distinction is made between *local objects* and *global objects*. An object is said to be local to the database containing the object, and an object is said to be global to those databases that may access the object. Since a given database may access all its local objects, an object is always global to the database to which it is local.

Objects may be relocated from one database to another. Therefore, it is important to distinguish between the *creator* of the object and the *owner* of the object. The creator of a given object is the database creating the object; the owner of an object is the database currently containing the object. At object creation time the owner and the creator of the object are identical. The creator of an object remains unchanged throughout the life-time of the object, whereas the owner of the object change every time the object is relocated.

[3]In the current implementation of ODM, relationships are not objects in the sense that they cannot be related to other objects.

3.2. Database and Object Naming

Each database in the distributed environment is uniquely identified by its *database identifier*, which is an atomic string of characters. A database can be referenced either by its database identifier or by a don't care symbol "*". A don't care symbol, when used as a database reference, denotes every database in the system.

Descriptor and behavioral objects can be denoted by three different kinds of *object identifiers*: local, global, and transparent object identifiers. As noted above, the object identifier of a descriptor object is the value of the string that constitutes the object, and the object identifier of a behavioral object is the atomic string denoting the object[4]. A *local object identifier* is an object identifier as introduced in ODM. It uniquely identifies an object within the database that is local to the reference. Each database in the network may contain an object denoted by the same local object identifier, but there can at most be one such object per database. Objects with identical local object identifiers are equivalent. A *global object identifier* uniquely identifies an object within the entire network of databases. It is composed of a local object identifier, an "@", and a database identifier:

<object identifier>@<database identifier>

Note that the global object identifier of a given object depends on the owner of the object and not the creator (as it is the case in R* [Lindsay 80]). If an object is relocated from one database to another, its global object identifier is changed accordingly.

The presence of distribution-dependent information in a global object identifier may seem inconvenient, but location transparency can by attained by using transparent object identifiers. A *transparent object identifier* denotes every object in the entire database network that has the same local object identifier (at most one per database). A transparent object identifier is composed of a local object identifier, an "@", and a don't care symbol ("*"):

<object identifier>@*

If there is only one object in the global system with a given local object identifier, that object may be uniquely referenced with its transparent object identifier, i.e., the object reference is completely location transparent.

Suppose that a database network consists of the three databases (DB1, DB2, and DB3), and that each database contains the two objects 'Employees' and 'Has-instances'. Then the operation FIND('Employees', 'Has-instances', ?) returns references to every object in the network that has been related locally to the local objects 'Employees' and 'Has-instances'. The operation FIND@DB2('Employees', 'Has-instances', ?) returns references to every object in the network that has been related in DB2 to the two local (with respect to DB2) objects 'Employees' and 'Has-instances'. The operation FIND@DB3('Employees'@DB1, 'Has-instances', ?) returns references to every object in the network that has been related in DB3 to the object 'Employees' in DB1 and the local (with

[4]As stated above, abstract objects do not have object identifiers; they can be referenced only by their relationships with other objects.

respect to DB3) object 'Has-instances'. The operation FIND@*('Employees', 'Has-instances'@*, ?) returns references to every object in the network that has been related in some database to the local object 'Employees' and the object 'Has-instances' in some database.

3.3. The Primitive Operations of DODM

DODM supports the primitive operations as ODM. The CREATE operation creates a local object, and the DELETE operation deletes a local object. The ISOBJECT operation tests if an object reference denotes an existing global object. The RELATE operation creates a local relationship between three global objects, and the DETACH operation deletes a local relationship. The FIND operation returns a set of global object references denoting objects that have been related locally. The PRINT operation prints the object identifier of a global object. Finally, the DEFINE operation creates a new local behavioral object.

In addition to the primitive operations of ODM, DODM has primitive operations for object sharing, and for copying and moving objects from database to database:

3.3.1. EXPORT(o: objectref, d: dbref [, oi: objectid])

Object sharing among individual databases is specified by EXPORT operations. After the execution of an EXPORT operation, the referenced object is known and accessible to the specified database(s). If an object identifier is given, the exported object becomes known to the importing database only by that name. Unless a given object explicitly has been exported to a certain database, the object is not known to that database and cannot be accessed by that database. The renaming facility allows the same object to be exported to different databases under different names. If an object is renamed upon export, the new identifier need only be unique with respect to other objects exported from the same database to the same importing database. The EXPORT operation results in an execution error if the referenced object is not a local object.

The following examples illustrate the use of the EXPORT operation. The operation: EXPORT('Ron', Payroll, '258-17-8513') causes the local object 'Ron' to be known to the Payroll database as the object '258-17-8513'. The operation EXPORT('FIND', IRS) makes the FIND operation known to the IRS database. The operation EXPORT('EXPORT', Smith) allows the Smith database hence forward to perform EXPORT operations in the exporting database.

3.3.2. REVOKE(o: objectref, d: dbref)

The REVOKE operation causes the referenced object no longer to be known to the specified database(s). For example, REVOKE('EXPORT', Smith) revokes the export right granted above to the Smith database.

3.3.3. EQUIVALENCE(os: set of objectref): boolean

The EQUIVALENCE operation returns the value true if the set of objects specified are equivalent; otherwise it returns the value false. Suppose the two databases DB1 and DB2 each contain the two objects 'employee' and 'salary'; then, the operation EQUIVALENCE({'employee', 'employee'@DB2}) returns the value true, whereas the operation EQUIVALENCE({'employee', 'salary'}) returns the value false.

3.3.4. COPY(o: objectref): objectref

The COPY operation creates a copy of the referenced object in the local database where it is considered a new object. The original object and the copy have the same content and are therefore equivalent. The COPY operation returns a reference to the new local object. The COPY operation has no effect if the referenced object is a local object. The examples shown below illustrate the use of the COPY operation. First, the book entitled "Rabbit, Run" by John Updike is copied from the Library database to the local database. Then, the telephone number (213) 743-5501 is copied from the Payroll database to the Carpool database:

```
var book, mybook: objectref;
    books: set of objectref

books := FIND@Library('John Updike',
    'Rabbit, Run', ?);
for each book in books do
   mybook := COPY(book);

COPY@Carpool('(213) 743-5501'@Payroll)
```

Notice that the Library database is not responsible for the new copy of the book "Rabbit, Run" once it has been created in the importing database; the importer is the owner of the book copy and may modify it as desired.

3.3.5. MOVE(o: objectref)

The MOVE operation moves the referenced object to the local database. If the specified reference denotes a local object, the operation has no effect. The object reference remains the same after the object has been moved. The section describing the implementation of DODM explains how location transparency in object references can be achieved. The operation MOVE('Chivas Regal'@Import) moves the object 'Chivas Regal' from the Import database to the local database. The operation MOVE@IRS('238038280'@BankX) moves the account number 238038280 from the BankX database to the IRS database.

3.3.6. ISLOCAL(o: objectref): boolean

The ISLOCAL operation returns the value true if the specified object reference denotes a local object; otherwise it returns the value false. Suppose the two databases DB1 and DB2 each have an object with the local object identifier 'red'. Then the operation ISLOCAL('red'@DB2) returns the value false if performed in DB1. The operation ISLOCAL('red') returns the value true if performed in either DB1 or DB2.

3.4. Implementation of DODM

Each database in a DODM database network contains its own objects and relationships in a way similar to an ODM database. As in ODM, objects are referenced by their unique object keys; but in order to be able to distinguish between objects from different databases, object keys must be unique within the entire network. This is achieved by using object keys that have two parts: a key that is unique within a given database (like the ODM object key), and the database identifier of that database. This key format is similar to the format of a global object identifier. There is a difference, however. An object key will never change during the lifetime of an object, not even if the object is relocated from one database to another. This is not the case for a global object identifier which changes every time the object is relocated in the network.

Since object keys are unique within the entire database network, a relationship can be described by the keys of the three objects in the relationship. Thus, relationships may span database boundaries. Furthermore, a relationship is not affected by objects being relocated to other databases after the relationship has been established.

In the experimental implementation of DODM currently under development, each node in the database network consists of a *database*, a *catalog manager*, a *communication subsystem*, and a *database operation interpreter* (see Fig. 3-1). These components allow users and application programs at a given node to communicate, cooperate, and share objects with users and programs at other nodes in the network. The database, of course, stores all the objects and relationships. Objects are described in the catalogs by their object keys and object identifiers. The catalog manager maintains two kinds of catalogs that provide access to the objects in the database:

- The *local catalog* describes every object in the database. It is used for resolution of local object references.

- *Export catalogs* are used to describe object sharing between individual databases. The catalog manager maintains an export catalog for every remote database in the network. An export catalog describes all the objects in the database that are known to a specific remote database, and references from that remote database are resolved from the export catalog.

The database operation processor interprets database operations. If it is necessary to access remote objects in order to process a given operation, the databases containing the remote objects are activated via the communication subsystem.

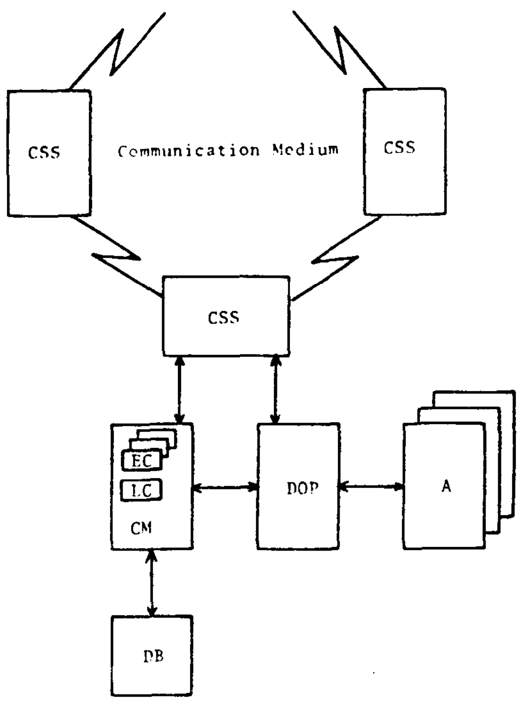
The communication subsystem provides the following three primitives:

- send(receiver-station-name, message)

- broadcast(message)

- receive(sender-station-name, message, type)

The send primitive sends the message from the sender station to the receiver station, where it is queued. The broadcast primitive simply broadcasts the message to all the stations in the network. The receive primitive obtains from the queue of incoming messages the next message, the address of the station that sent it, and the type of the message. The type indicates whether the message was sent by a send operation or a broadcast operation.



CSS	Communication Subsystem	CM	Catalog Manager
DB	Database	LC	Local Catalog
A	Application Program	EC	Export Catalog
DOP	Database Operation Processor		

Figure 3-1: System Structure

Broadcast communication is used to implement location transparency. An object reference in the form of an object key, does not provide sufficient information to locate the corresponding object in the network. In order to resolve such a reference the database operation processor broadcasts a request to every database in the network via the communication subsystem. Every remote database then attempts to resolve the object reference from the export catalog corresponding to the requesting database. The requesting database tries to resolve the object reference from the local catalog. Finally, the requesting database is notified of the outcome of the catalog lookups.

In order to provide location transparent object references, the communication subsystem must support broadcast communication. Ethernet [Metcalfe 76] is a commercially available local area network that uses broadcasting as the basic communication technique. In fact, the Ethernet hardware broadcasts a message, whether it is intended for every node in the network or just a single node. Therefore, the communication subsystem can perform a send and a broadcast operation as explained above for the same cost. If broadcasting is not supported directly by the network hardware, it can be implemented on networks of both the star and ring configurations.

3.5. Limitations of DODM

DODM is a very simple database model for the modelling of objects and relationships in a logical network of databases. Mechanisms are provided to allow relationships to be established across database boundaries, objects are allowed to be copied and moved from database to database, and object sharing among individual databases is accommodated. Like ODM, DODM is not a high-level model appropriate for unsophisticated database users; it lacks semantic expressiveness, mechanisms for integrity control, high-level operations for database integration, etc. However, the main purpose of DODM is not to define such a high-level database model, but on the contrary, to define a small set of fundamental concepts to be used as a vehicle in the design and implementation of distributed database systems providing more expressive models.

4. An Example DODM Database

In this section, an example application of the use of DODM is presented; DODM is used here as a tool in the development of a distributed database system based on a semantic database model. The example application environment is a university, wherein students enroll in classes and faculty members instruct classes. Each department of the university maintains a local database describing its students and faculty members as well as the classes offered by the department. Students are allowed to enroll in classes offered by different departments, but faculty members are only allowed to instruct classes offered by their own department.

4.1. The Example Database Model

The example semantic database model used here is a simplified version of the Event Database Model (EDM) [King 82]. In EDM, a database is modelled by a database conceptual schema, which is a collection of *objects*. As is ODM, there are three kinds of objects: *descriptor objects*, *abstract objects*, and *behavioral objects*. Objects are classified into *types*, based on common properties. Relationships among objects are modelled by *attributes*. An attribute is a mapping from one object type (the *domain* type) to another (the *range* type). Thus, the attribute value of a given object in the domain type is a subset of the objects in the range type.

A type may be specified to be a *subtype* of another (parent) type. A subtype contains a subset of the objects in the parent type, it inherits all the attributes of the parent type, and in addition it may have attributes that the parent type does not have. A type may also be specified to be a *public* type[5]. A public type is known and accessible to every database in the distributed environment, together with all its instances and attributes.

By default, an attribute is a mapping between objects in the same local (with respect to the attribute) database. However, if an attribute is specified to be *global*, the attribute maps objects from the local database to local objects or to objects anywhere in the network that are instances of a public type by the same name as the range type of the attribute (name equivalence).

Figure 4-1 contains a portion of the database schema for the university application. It consists of four types, two of which are subtypes. The type Person has attributes Name and Id. Both attributes are descriptor objects. The types Student and Faculty are both subtypes of Person. Every student is a person and every faculty member is a person. In addition to the attributes defined on the Person type, the Student type has the attribute Enrollment and the Faculty type has the attribute Teaching. The Class type is a public type. It has the attributes Cname and Classno. The schema of each departmental database contains the type definitions specified in Figure 4-1.

```
Type Person
   attributes(Name: String,
              Id: String)

Subtype Student of Person
   attributes(Enrollment: Class is global)

Subtype Faculty of Person
   attributes(Teaching: Class)

Type Class is public
   attributes(Cname: String,
              Classno: String)
```

Figure 4-1: Example EDM Database Schema (partial)

Figure 4-2 shows how the schema in Figure 4-1 can be represented in DODM and an example EDM database is illustrated by its corresponding DODM representation in Figure 4-3. Each type and attribute defined in EDM is represented by an object in DODM. The DODM database has a root object related to every type object via the object 'types', and related to every attribute object via the object 'attributes'. This structure allows a user to request all the types defined in the EDM schema by the simple query FIND('root', 'types', ?), and similarly for attributes. Supertype/subtype hierarchies are modelled as relationships between the respective type objects via the object 'supertype'. A public type is represented by a relationship from the corresponding type object to itself via the object 'public'. Instances of a given type are objects that are related to the corresponding type object via the object 'instances'. An attribute is related to the type object of its domain type via the object 'domain', and it is related to the type object of its range type via the object 'range'. A global attribute is represented by a relationship from the corresponding attribute object to itself via the object 'global'.

[5]This feature is not part of the event database model per se [King 82], but has been added to handle the distributed case.

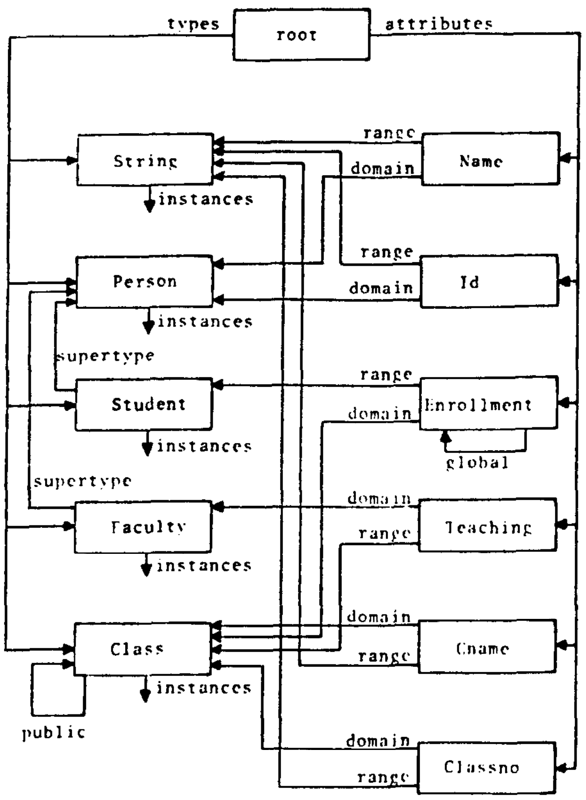


Figure 4-2: DODM Representation of the EDM Schema

All the DODM objects introduced to represent the EDM schema ('root', 'types', 'attributes', 'instances', 'supertype', 'domain', 'range', 'public', 'global', and the type and attribute objects) are meta objects; their sole purpose is to describe the user data in the EDM database. It is important to note that no distinction is made between meta data and user data at the DODM level.

4.2. The Example Database Operators

The EDM database model supports operators for data retrieval and data manipulation. There are two data retrieval operators:

- open(t: type, d: database): cursor
- apply(a: attribute, o: object): cursor

The actions of these operators are defined in terms of *cursors*[6] [Gray 78]. The *open* operator returns a cursor containing all the objects in the specified type, and the *apply* operator returns a cursor that contains the objects in the attribute value of the specified object with respect to the specified attribute.

There are four data manipulation operators:

[6] In the above specifications, type, attribute, and object are objectref's in DODM terminology. Similarly, cursor is a set of objectref, and database is a database identifier.

- add-instance(t: type [,s: string]): object
- remove-instance(o: object)
- add-attribute(d: object, a: attribute, r: object)
- remove-attribute(d: object, a: attribute, r: object)

The *add-instance* operator creates a new instance of the specified type and returns a reference to the new object. A string value must be given if the specified type is a descriptor type. The *remove-instance* operator removes the specified instance from the database. The *add-attribute* operator assigns the specified range object to the attribute value of the specified domain object with respect to the specified attribute, and the *remove-attribute* operator removes the specified range object from the attribute value of the specified domain object with respect to the specified attribute.

5. Concluding Remarks

This paper has described a simple model for object sharing in distributed database systems. The model provides a small set of operations for object definition, manipulation, and retrieval. First, the primitive operations and the implementation of ODM, a simple object-oriented database model for a single centralized database system, were described. A prototype implementation of this model was discussed. Then, the operations and the implementation of the DODM model, an extended version of ODM for the modelling of distributed databases, was described. An experimental implementation of DODM (currently under development) and the associated support facilities required were discussed. Finally, an example was provided to illustrate the use of DODM to design a semantically-expressive distributed database system.

Many of the ideas described in this paper are related to similar ideas in programming languages and operating systems. A database is responsible for data encapsulations very much like a module in a programming language, and the concepts of data importing and exporting are also common to the two research areas. Moreover, the primitive operations for object relocation and copying are similar to primitives in existing operating systems. However, it is beyond the scope of this paper to extensively compare the results obtained with significant work in other research areas.

Analysis, testing, and extensions of the research described in this paper are currently under study. In particular, the prototype DODM implementation will be used to further assess the adequacy and completeness of the primitives of the model. Another important area concerns the use of broadcast communication. Finally, the concurrency and multiple copy control issues have been avoided in this paper, by assuming a single copy of each object and single-user systems at each node in the network. Results of current research in these areas will be utilized as this research progresses to address these limitations. As noted above, the work described in this paper is part of a current research effort at the University of Southern California to design and develop a "personal information management environment" and experimental tool called INFOBASE.

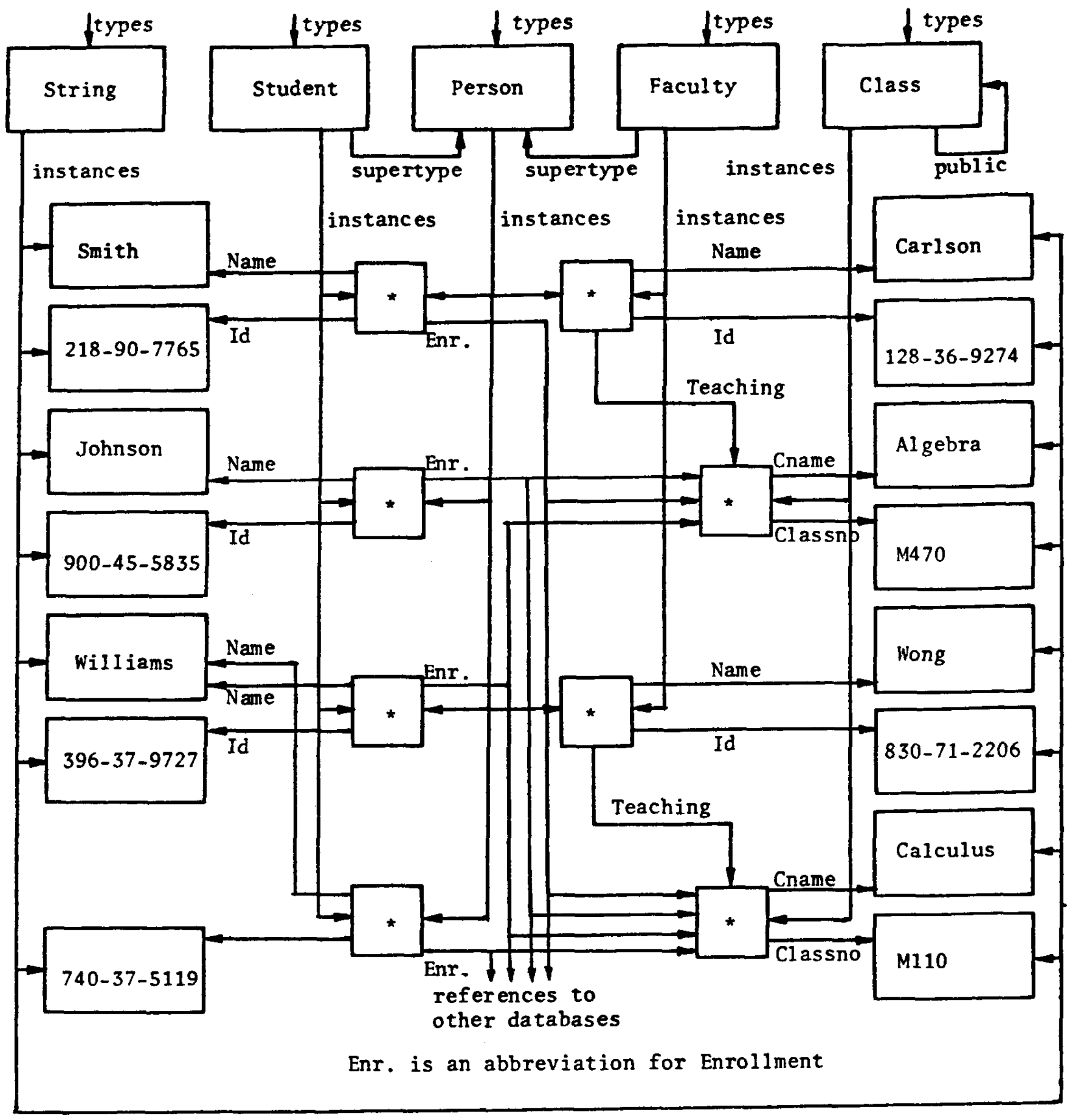


Figure 4-3: DODM Representation of the EDM Database

References

[Andler 82] S. Andler, I. Ding, K. Eswaran, C. Hauser, W. Kim, J. Mehl, and R. Williams. System D: A Distributed System for Availability. In *Proceedings of International Conference on Very Large Databases.* Mexico City, Mexico, September, 1982.

[Brodie 81] M. L. Brodie. On Modelling Behavioural Semantics of Data. In *Proceedings of International Conference on Very Large Databases.* Cannes, France, September, 1981.

[Gray 78] J. N. Gray. Notes on Data Base Operating Systems. In *Lecture Notes in Computer Science*, pages 393-481. Springer Verlag, 1978.

[Heimbigner 82] D. Heimbigner. *A Federated Architecture for Database Systems.* USC Technical Report TR-114, University of Southern California, August, 1982.

[Kimbleton 79] S. R. Kimbleton, P. S. C. Wang, and E. Fong. XNDM: An Experimental Network Data Manager. In *Proceedings of Berkeley Workshop on Distributed Data Management Systems.* Berkeley, Ca., August, 1979.

[King 82] R. King and D. McLeod. The Event Database Specification Model. In *Proceedings of International Conference on Improving Database Usability and Responsiveness*, pages 299-322. Jerusalem, Israel, June, 1982.

[King 83] R. King and D. McLeod. A Unified Model and Methodology for Conceptual Database Design. In M. Brodie, J. Mylopoulos, and J. Smith (editors), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Database, and Programming Languages.* Springer-Verlag, 1983. (to appear).

[Lindsay 80] B. Lindsay. *Object Naming and Catalog Management for a Distributed Database Manager.* IBM Research Report RJ2914, IBM Research Laboratory, San Jose, Ca., August, 1980.

[Litwin 81] W. Litwin. *Logical Design of Distributed Databases.* Technical Report MOD-I-043, INRIA, July, 1981.

[Lyngbaek 82] P. Lyngbaek and D. McLeod. *A Distributed Name Server for Information Objects.* USC Technical Report TR-200, University of Southern California, December, 1982.

[McLeod 83a] D. McLeod. *INFOBASE: An Environment for Personal Information Management.* USC Technical Report, Computer Science Department. University of Southern California, Los Angeles, Ca., September, 1983.

[McLeod 83b] D. McLeod, K. V. Bapa Rao and K. Narayanaswamy. Information Modelling for CAD/VLSI. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* San Jose, California, May, 1983.

[Metcalfe 76] R. M. Metcalfe and D. R. Boggs. Ethernet: Distributed Packet Switching for Local Computer Networks. *Communications of the ACM* 19(7):395-404, July, 1976.

[Motro 81] A. Motro and P. Buneman. Constructing Superviews. In *Proceedings of ACM SIGMOD International Conference on Management of Data.* Ann Arbor, Michigan, April-May, 1981.

[Rothnie 80] J. B. Rothnie, Jr., P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. Reeve, D. Shipman, and E. Wong. Introduction to System for Distributed Databases (SDD-1). *ACM Transactions on Database Systems* 5(1), March, 1980.

[Smith 77] J. M. Smith and D. C. P. Smith. Database Abstractions: Aggregation and Generalization. *ACM Transaction on Database Systems* 2(2):105-133, June, 1977.

[Smith 81] J. M. Smith, P. A. Bernstein, D. Umeshwar, N. Goodman, T. Landers, K. W. T. Lin, and E. Wong. Multibase - Integrating Heterogeneous Distributed Database Systems. In *Proceedings of National Computer Conference*, pages 487-499. June, 1981.

[Stonebraker 76] M. Stonebraker, G. D. Held, and P. Kreps. The Design and Implementation of INGRES. *ACM Transactions on Database Systems* 1(3), 1976.

[Stonebraker 77] M. Stonebraker and E. Neuhold. A Distributed Database Version of INGRES. In *Proceedings of Berkeley Workshop on Distributed Data Management Systems.* Berkeley, Ca., May, 1977.

[Tsichritzis 82] D. Tsichritzis, F. A. Rabitti, S. Gibbs, O. Nierstrasz, and J. Hogg. A System for Managing Structured Messages. *IEEE Transactions on Communications* COM-30, January, 1982.

[Williams 81] R. Williams, D. Daniels, L. Haass, G. Lapis, B. Lindsay, P. Ng, R. Obermarck, P. Selinger, A. Walker, P. Wilms, and R. Yost. *R*: An Overview of the Architecture.* IBM Research Report RJ3325, IBM Research Laboratory, San Jose, Ca., February, 1981.