Object-Oriented Technology for Integrating Distributed Heterogeneous Database Systems

Dr. Marion G. Ceruti, NCCOSC RDT&E DIV 4221
Dr. Magdi N. Kamel, Naval Postgraduate School
Dr. Bhavani M. Thuraisingham, The MITRE Corporation.

ABSTRACT

In this paper, we describe the application of object technology for heterogeneous database integration. We first discuss the issues of integrating heterogeneous database systems and present a collection of some heterogeneous databases as they occur in the Joint Maritime Command Information System (JMCIS) as an example of a system that could benefit from this technology. Next we describe the need for object technology and its application to handle schema heterogeneity and platform heterogeneity. In particular, we give an overview of the main features of an object-oriented model and why this model is a good candidate as an intermediate model for integrating heterogeneous databases. We discuss the use of Object Management Group's Common Object Request Broker Architecture (CORBA) for resolving platform heterogeneity. Finally, we explore the directions for future research.

Index terms—Object-oriented technology, object data model, relational data model, federated database systems, multidatabase systems, heterogeneous database integration, schema integration, database heterogeneity, semantic heterogeneity, data transformation, command and control systems, object request broker, CORBA

I. INTRODUCTION

The rapid growth of database technologies and networking have had a major impact on the information processing requirements and methods in organizations. Not surprisingly, this topic has received considerable attention in the literature (See for example [2, 3, 7, 8]). In recent years, most large organizations, including the Department of Defense, have seen a dramatic proliferation of incompatible databases and their associated database management systems (DBMS). Sooner or later, these organizations discover the need to integrate the data in these incompatible databases to satisfy their increasing and changing information requirements.


79
Various types of heterogeneity have been discussed in the literature (See for example [7, 8]). These include the following:

(i) Schema (or data model) Heterogeneity: Databases in a heterogeneous architecture frequently are represented by different data model with different conceptual schemas.

(ii) Transaction Processing Heterogeneity: Different algorithms for transaction processing are used in different DBMSs. For example, some approaches use locking, others use time stamping, whereas still others use validation mechanisms for concurrency control.

(iii) Query Processing Heterogeneity: Different query processing and optimization strategies may be used in different DBMSs.

(iv) Query Language Heterogeneity: Different DBMSs use different query languages. Even in relational DBMSs, dialects of SQL are used.

(v) Constraint Heterogeneity: Different integrity constraints are enforced in different DBMSs which often lead to inconsistencies. For example, one DBMS could enforce a constraint that all ships must have an overall readiness rating of at least "2" to be deployed, whereas another DBMS may not enforce such a constraint.

(vi) Semantic Heterogeneity: The meaning of data may differ in the different component databases. For example, the port of a ship could mean the home port of origin in one component, whereas in another component, it could mean the port of current location.

(vii) Platform Heterogeneity: Different vendors have produced different hardware platforms, operating systems, network protocols, etc., without considering interoperability issues.

To facilitate interoperability and integration among these heterogeneous databases systems, two approaches, the tightly coupled and loosely coupled approaches [8], have been proposed and developed. In the tightly coupled approach, a unified, global schema is constructed from the underlying individual schemas of the component databases to be integrated. Users and applications need not be concerned with the component database schemas or the integration process. They treat the federated database (FDB) as a single database and issue all query and data-manipulation operations on that schema. In the loosely coupled approach, the component database schemas are not integrated into a global schema. Rather, a method of performing queries on multiple databases is defined and developed to allow access to several or all the component databases simultaneously while hiding the heterogeneity of the underlying databases. A user would be presented with each of the sub-schemas and a powerful data manipulation language or set of tools that enables queries on several or all the component databases. In this approach, the user
needs to know about the structure of the sub-schemas to perform queries and data manipulation successfully. Regardless of the chosen approach, an intermediate representation schema in a semantically rich model is needed to facilitate the transformation of the schemas between the different database systems.

The organization of the paper is as follows. Section II describes an example of an application from a DOD program to illustrate the fact that heterogeneous database integration is a real-world problem. Section II also discusses the need for integrating the component databases in the example. Section III presents an overview of the object-oriented approach and model and discusses why it is a desirable model for facilitating the integration across many heterogeneous database systems as well as administering them. Section IV shows how the object-oriented model can be used as an intermediate data model to address some aspects of heterogeneity, such as schema heterogeneity and platform heterogeneity. It also describes the use of distributed object management system approach to interconnect heterogeneous database systems. Finally, the paper concludes in Section V with a summary and a discussion of directions for future research.

II. A HETEROGENEOUS, COMMAND AND CONTROL DATABASE SYSTEM

The Joint Maritime Command Information System (JMCIS) is an example of a Command and Control system for the Navy, Marine Corps, and Coast Guard resulting from the integration of many legacy systems, including but not limited to the Operations Support System (OSS) and the Navy Tactical Command System-Afloat (NTCS-A). JMCIS was chosen as an example here because many of the major system components and segments were developed separately by different engineers at a variety of agencies, organizations, and commands. Therefore, the aggregate of databases that support these components exhibits considerable heterogeneity, including heterogeneity with respect to platform, query and transaction processing, DBMS, data model, as well as schema heterogeneity and semantic heterogeneity discussed in Section I. For example, the Track Database Manager (TDBM) uses a flat-file data format for real-time applications, and a relational DBMS for analyses and applications with less critical-time constraints. Relational DBMS also are used commonly to manage data in both the afloat (NTCS-A) and the ashore (OSS) variants of JMCIS. The Naval Warfare Tactical Database components, the Naval Intelligence Database (NID) and the Radar Parametric Data Set (RAPADS) are specific examples of data sets managed using RDBMS.

Even within the relational model, DBMS heterogeneity occurs because software from various DBMS vendors has been chosen to support JMCIS database segments of different origins. For example, the JMCIS ashore community has been using Oracle DBMS to manage data from the NID, RAPADS, casualty reports (CASREPS), movement reports (MOVREPS), Naval Status of Forces (NSOF), Status of Readiness and Training (SORTS), employment schedules (EMPSKD), and the
Defense Mapping Agency's (DMA) ports and airfields data sets. These data sets originated in the OSS Integrated Database [4], and have been chosen for migration into the JMCIS ashore variant.

By contrast, JMCIS afloat data are managed using Sybase DBMS for data in the Naval Intelligence Processing System/Services (NIPS) Central Database Server (CDBS). Examples of CDBS component databases are electronic warfare data sets, a message database and a reference database. Moreover, other systems that depend on the JMCIS afloat variant for track and order-of-battle data, also use Sybase DBMS to manage internally generated data for their applications, as well as data derived from JMCIS database components. The DBMS-vendor heterogeneity is not limited to Oracle and Sybase. Informix DBMS software also is included in the Navy's Tactical Advanced Computer-4 (TAC-4) contract and in a recent Coast Guard upgrade.

Some object-oriented technology is being introduced into JMCIS gradually and more object-oriented migration is in the planning stages. New applications are being developed increasingly with a view toward modularity and the open-systems architecture. Similarly, older applications are becoming targets for re-engineering. An effort will be underway to redesign the structure of the Modernized Intelligence Database (MIDB) using an object-oriented data model. MIDB, which originally was developed using a hierarchical data model, is a DOD standard database included in the CDBS of JMCIS afloat. The JMCIS FDB is an example of a federated database system with loose coupling between the components. A goal of the JMCIS integration is to evolve toward tighter coupling. Object-oriented technology can facilitate this tighter coupling.

Not only do the components of the JMCIS FDB exhibit every kind of heterogeneity [3], but any one of many subsets of these components can be present on platform and/or the local area network (LAN) at a given JMCIS site. Some data, such as tracks and casualty reports are dynamic whereas other data sets, such as the DMA ports, are relatively static. Keeping each node current and consistent with all other nodes has been the subject of much effort. Presenting these data in a uniform manner to applications and users also is a challenge because of the wide diversity in data sources. The agency responsible for the production of each data set "owns" these data, and generally offers these data sets to data engineers and developers in a format that is most convenient for the originating agency, rather than the for users. The diversity in the JMCIS FDB is depicted in Figure 1, which suggests the application of an Object-Request Broker (ORB) to address the database heterogeneity problem. The ORB concept is explained in section IV.B.1.
Figure 1. Application of object-oriented architecture to the JMCIS database heterogeneity problem.
III. THE OBJECT-ORIENTED DATA MODEL AS AN APPROACH TO INTEGRATING HETEROGENEOUS DATABASES

In this section we present an overview of the object-oriented data model [1] and a discussion of why an object-oriented data model is particularly useful for integrating distributed heterogeneous databases.

III.A. Object-Oriented Data Model Overview

A fundamental concept in an object-oriented model is the object class, which describes a collection of similar objects, called object instances. An object class has a name, a set of properties that describes its state, and a set of methods that describes its behavior. Each instance of a given class has a value for each property of the class and can invoke the methods associated with the class to perform operations on the properties. Object classes are arranged in a hierarchy known as a class hierarchy. This class hierarchy allows a class, called a subclass, to be defined starting with the definition of its parent class, called a superclass. The subclass inherits the superclass properties and methods, in addition to having its own specific properties and methods.

Whereas a widely accepted definition of what constitute an object-oriented data model has yet to be developed, there is a general consensus on certain core concepts, which are discussed below (See for example [6, 9]). The object-oriented data model is illustrated in Figure 2.

(i) Encapsulation: This refers to the fact that an object has a public interface part and an implementation part that is encapsulated or hidden from public view. The interface part specifies a set of operations that other objects or applications are allowed to perform on the object. The implementation part describes how each operation is used. Encapsulation provides a form of "logical data independence" because one can change the implementation of a method without changing the programs using that method, thus providing for a greater degree of modularity than some other data models.

(ii) Message Communication: Objects communicate with each other through messages. A message consists of the name of an object followed by the name of a method to be executed and optionally any parameters that the method requires for its execution.

(iii) Polymorphism: Polymorphism is the ability of different kind of objects to respond differently to the same operation depending on the type of object.

(iv) Inheritance: A subclass inherits the properties and methods of the superclass, which has two advantages. It allows for a more concise and precise way to model the world and it facilitates the identification of shared specifications and implementations in applications.
Figure 2. The object-oriented data model
(v) Data Abstraction: The ability to define new high-level data types by combining existing data types to achieve a better representation of the semantics of a new application is called "data abstraction."

In Section IV, the object-oriented concepts above will be related to those of the relational model with a view toward facilitating a data representation transformation between these models.

III.B. Why use an Object-Oriented Data Model for Integrating Heterogeneous Databases?

The object-oriented data model is a particularly powerful model for integrating heterogeneous database systems for several reasons (See for example [2, 10]). First, the extensible data typing feature of an object-oriented model makes the model semantically rich and therefore, a good candidate for an intermediate data model to represent the semantics of any existing data model of the component databases. The use of the object-oriented model as an intermediate data model is illustrated in Section IV.A. Secondly, an object-oriented data model supports both operational mapping and structural mapping. Operational mapping allows the database integration engineer to define the correspondence between operations at different levels of the system that is being integrated. This feature is particularly useful for integrating non-traditional databases that have no formal schema. Third, the inheritance characteristic of object-oriented models simplifies the definition of tailored integrated views, since it allows a view to use existing mappings defined for other views. In addition the object-oriented model could be used to represent the entire distributed heterogeneous system. With this approach, the different components of the distributed heterogeneous system are represented as objects that can interact with each other by exchanging messages. The distributed object management approach is illustrated in Section IV.B.

IV. OBJECT TECHNOLOGY FOR HETEROGENEOUS DATABASE INTEGRATION

In section IV.A we discuss the application of object technology to handle schema heterogeneity (See for example [10]). In section IV.B we discuss how object technology could be used to handle platform heterogeneity (See for example [11]).

IV.A. Object Technology for Schema Integration

Most users find increasing difficult in learning a variety of data representation schemes on systems that they must access remotely. A much more desirable alternative is for the users to know at most one or two schemes to be effective in their work. For example, a user who needs to access a relational database at one site, a hierarchical database at another site, and an object-oriented database at still another site, will be able to access these databases more efficiently using his or her preferred
data model and language. Thus, schema integration will be reduced to transforming the constructs of one data model into those of the others.

In this method, translators can be installed to perform the necessary transformations between the various data representations. This is advantageous because it relieves the users of the task of learning the details of more than one data representation scheme. Unfortunately, this method suffers from a disadvantage; if N different data models are utilized, \( (N^2 - N) \) translators will be required. Thus, the task of producing the required translators becomes costly and cumbersome, especially if the number of data models in the system continues to grow.

A more efficient manner in which to interconnect data models is to use an intermediate representation scheme that can act as a central point of translation. The individual data representation schemes are all translated into the intermediate representation scheme. The intermediate representation schemes then are translated back into the individual data representation schemes to allow for a two-way conversion between the individual and the intermediate representation schemes. Thus, for N different data models, only 2N translators are required because in this method, each intermediate representation scheme communicates only with the intermediate representation scheme.

An object-oriented representation scheme was proposed to serve as the intermediate representation scheme [10]. In the present work, we show how an expanded version of this scheme can be applied to an example derived from the SHIP_MASTER_LIST table in the JMCIS FDB. This object-oriented representation is sufficiently powerful to model structural in addition to behavioral properties of entities. Moreover, the majority of constructs of other data models, such as the relational model, can be transformed into constructs of object-oriented data models. For example, schema integration can be accomplished using an object-oriented approach in heterogeneous environment that consists of relational DBMS and a flat-file data server.

Because relational DBMSs are in such widespread use throughout the DOD and elsewhere, we examine the relationship between a relational model and an object-oriented model. In the strict theoretical sense, several steps are required to generate relations from an object-oriented data model. The theory associated with these steps is described in more detail in Appendix A. An intermediate relational data model is generated to be a generic relational data model. The end result is that the constructs of the generic relational data model are transformed into those of a specific relational data model such as one used in commercial DBMSs. The essential concepts are illustrated below with command and control examples.

Object classes are mapped into one or more domains, which can contribute to one or more relation variables (See Appendix A). Relations are then constructed using the relation variables as structures which are filled with values from the appropriate domains. Consider the class SHIP illustrated in Figure 3. This class has
attributes HULL_NBR, CALL_SIGN_INTL, CATEGORY, PRIMARY_MISSION, and COUNTRY. It has an instance object whose HULL_NBR is "599315FC," CALL_SIGN_INTL is "5FC," CATEGORY is "Surface Ship," PRIMARY_MISSION is "ASW," and COUNTRY is "US." This ship class is transformed into a complex domain, $D_A$, consisting of the following domains, or data types:

- **HULL_NBR**: set of all alphanumeric vessel identifiers
- **CALL_SIGN_INTL**: set of all alphanumeric international call signs
- **CATEGORY**: set of characters indicating surface ship, submarine, amphibious craft, etc.
PRIMARY_MISSION set of three-alphabetical characters designating the primary warfare mission area
COUNTRY set of two-alphabetical characters designating the country that is using the ship

From this complex domain, a SHIP relation can be formed. The object identifiers of the instances can be used as primary keys or a mapping could be provided between these object identifiers and the user-generated primary keys if users need to access the relations directly.

This methodology provides a way to transform constructs such as associations and inheritance. The association between object classes, "SHIP" and "PORT" is "SAILS TO." Associations between object classes are similar to message communications between objects. For example, a particular ship, an object instance of class "SHIP," communicates with its destination (another object instance of class, "PORT," by sailing to it. The association also has attributes PORT_ID, PORT_NAME, HARBOR_TYPE, PILOTAGE, LATITUDE, LONGITUDE, and COUNTRY [12]. A relation can be formed to describe this association by drawing from the domains that constitute the essential object classes. The attributes of this relation consist of the attributes of the association and the identifiers of the classes involved in the association.

Object classes can be arranged into a class hierarchy. That is, VEHICLE class has SHIP as its subclass. Similarly, SHIP has SURFACE SHIP and SUBMARINE as object subclasses. A domain exists for each class and subclass. Depending on the degree of normalization, the relation variable generated for the subclass may contain only the attributes that distinguish the subclass from other subclasses or it could include additional attributes also found in the superclass. Any instance of the subclass SHIP will be represented by two tuples; one in the relation VEHICLE and one in the relation SHIP.

IV.B Distributed Object Management Approach

This section describes how distributed object management systems such as the Common Object Request broker Architecture could be used for heterogeneous database integration.

IV.B.1 The Common Object Request Broker Architecture. The information on CORBA discussed here is from [5]. CORBA consists primarily of the object model, the Object Request Broker (ORB) and object adapters, and the Interface Definition Language (CORBA-IDL). Each component is discussed below.

Object semantics and object implementation are described by the object model. Object semantics include the semantics of an object, type, requests, object creation and destruction, interfaces, operations, and attributes. Object implementation includes
the execution model and the construction model. In general, the essential constructs of most object models can be found in the object model of CORBA.

An essential feature of the ORB is that it enables communication between a client and a server object. A client invokes an operation on the object and the object implementation consists of the code and data needed to implement the object [5]. The ORB provides the required mechanisms to identify the object implementation for a particular request and enables the object implementation to receive the request. The ORB also provides the communication mechanisms needed to deliver the request. Furthermore, the ORB supports the activation and deactivation of objects and their implementations. The ORB generates and interprets object references. To summarize, the ORB provides the mechanisms to locate the object and communicate the client's request to the object. The client does not need to know the exact location of the object or the details of its implementation. Objects use object adapters to access the services that the ORB provides.

IDL is a declarative language that describes the interfaces that the object implementations provide and that the client objects call. It should be noted that the clients and object implementations are not written in IDL. The IDL grammar is a subset of ANSI C++ with additional constructs to support the operation invocation mechanism. An IDL binding to the C language has been specified, whereas other language bindings are in progress. IDL is used to communicate between a client and a server in the following manner. Two types of modules, the IDL stub and the IDL skeleton, are connected to the ORB core. The client's request is passed to the ORB core via an IDL stub, and an IDL skeleton delivers the request to the server object from the ORB core.

IV.B.2. The Use of CORBA for Integrating Heterogeneous Database Systems. Section I presented an overview of various types of heterogeneity and section IV.B.1 explained OMG's CORBA. This section provides a description of some initial directions on using CORBA for integrating heterogeneous databases.

A major motivation for adopting a CORBA-like approach to the integration of heterogeneous databases is the complexity of migrating legacy databases to new-generation architectures. Whereas the migration of such databases and applications to the client-server architectures is desirable, the costs of such migration can be enormous. Therefore, a better approach is to keep the legacy databases and applications and develop mechanisms to integrate them with new systems. These mechanisms include the distributed object management system approach in general and the CORBA approach in particular.

The major advantage of the CORBA approach is the ability to encapsulate legacy database systems and databases as objects while eliminating the need for major modifications as is depicted in Figure 4. However, the techniques to handle the various types of heterogeneity are still needed. This is because the CORBA approach does not handle some problems like transaction heterogeneity and semantic
Figure 4. Encapsulating legacy and new-generation database systems

Figure 5. A client-server architecture for JMCIS
heterogeneity. However, the procedures for handling the various types of heterogeneity can be encapsulated in the CORBA environment and invoked appropriately. These concepts are illustrated below with some examples.

Consider the need for clients to communicate with a group of database servers. This is shown in Figure 5, which uses some components of JMCIS as an example. One way is to encapsulate the database servers as objects and have the clients issue appropriate requests and access the servers through an ORB. If the SQL-based servers are used, the entire SQL query or update request could be embedded in the message. When the method associated with the server object gets the message, the method can extract the SQL request and pass it to the server for execution. The results from the server objects are encoded as a message and passed back to the client via the ORB. This approach is illustrated in Figure 6, which contains a generalization of some features of Figure 1 as both include an ORB as the intermediate between clients and servers.

Next, consider the issue of how to deal with a particular type of heterogeneity. Suppose a SQL-based client is present with a server is some legacy database system based on the network model. Then the client's SQL query will need to be transformed into an appropriate language that the server can understand. In Section IV.A, the issues of transforming one representation scheme into another were discussed. The client's request is sent to the module responsible for performing the transformations. This module, called the "transformer," could be encapsulated as an object. The client's SQL request is sent to the transformer, which converts the request into a format that the server can understand. The transformed request is sent to the server object. Note that the transformer could transform the SQL representation directly into a network representation or it could use an intermediate representation to complete the transformation.

The Distributed Processor, which is a module that can perform the distributed data management functions, is responsible for handling functions such as global query optimization, and global transaction management. This module also is encapsulated as an object and processes the global requests and responses. The server assembles the response sent to the transformer to convert into a representation that the client can understand.

If semantic heterogeneity is an issue that requires attention, one may need to maintain a data repository to store the different names given to a single object or the different objects represented by a single name. The repository could be encapsulated as an object which would resolve semantic heterogeneity. For example, referring to Figure 1, a JMCIS client application could request an object to be retrieved from multiple servers. The request is first sent to the repository which will issue multiple requests to the appropriate servers depending on the names used to specify the object. The response also could be sent to the repository so that it can be presented to the client in an appropriate manner. Note that the repository could be an extension of the transformer. All the communications are carried out through the ORB.
V. SUMMARY AND FUTURE CONSIDERATIONS

In this paper, we first described the issues involved in integrating heterogeneous database systems, including some real-world examples. We also described the reasons for an object-oriented approach to integrating heterogeneous database systems. We introduced a multi-phased approach that involves several steps between the object-oriented data model, and the relational model. Two types of applications we discussed include the use of the object model for schema integration to handle schema heterogeneity, and the use of a distributed object management system to handle platform heterogeneity.
Several issues need further consideration with respect to the use of CORBA for integrating heterogeneous databases. For example, what is the best level of granularity at which to apply encapsulation? Should a server be encapsulated as an object as described in this paper? How can databases be encapsulated? Should an entire database be encapsulated as an object or should it consist of multiple objects? Should stored procedures be encapsulated also? Although much work still needs to be done, the various approaches that are being proposed to handle these issues are showing substantial promise. Furthermore, until efficient approaches are developed to migrate the legacy databases and applications to client-server based architectures, approaches like CORBA and other distributed object management systems for integrating heterogeneous databases and systems are needed.

REFERENCES


APPENDIX A

OBJECT CLASSES AND DOMAINS: THE INTERMEDIATE STEPS BETWEEN THE OBJECT-ORIENTED AND RELATIONAL DATA MODELS

Object classes in the object-oriented data model map to domains in the relational model. By contrast, object classes do not map directly to either relations or relation variables. Domains, in addition to being conceptual pools of values from which various attributes draw their actual values, are really scalar data types [A3], sometimes known as "atomic," "basic," or "primitive" data types, which also allows for the possibility of strong data typing. These atomic data types can be aggregated into structured data types that are no longer scalar, by means of type constructors. Examples of type constructors include lists, arrays, and relations, in which each tuple has a composite data type that results from the contribution from the scalar data type, or domain, of each attribute in the relation [A1, A2].

Relations, which contain tuples, should not be confused with relation variables, which have no tuples but serve as headers for tuples. A relation variable is a set of ordered pairs \((A, V)\) where \(A\) is an attribute and \(V\) is its domain. This set serves as the specified heading, \(H\), whose permitted values are relations [A2]. In essence, a relation variable or relvar, \(R\), is a created table structure or heading, whereas a relation, \(r\), constitutes the data fill that is arranged according to that structure [A1, A2]. More explicitly, consider the following expressions, in which relvar, \(R\) is a set of ordered pairs:

\[
R = \{ (A_1, V_1), (A_2, V_2) \ldots (A_n, V_n) \}
\]

and relation, \(r\) is a set of ordered triples:

\[
r = \{ (A_1, V_1, v_1), (A_2, V_2, v_2) \ldots (A_n, V_n, v_n) \}
\]

where each \(A\) is an attribute and \(V\) is the domain or data type that applies to that attribute, and \(v\) is the specific value from domain \(V\) that is assigned to attribute \(A\) in
relation r. Note that relvar, R does not have specific data value assignments, v. The relations, r are simply values of variable, R.

The main focus of this phase of the approach emphasizes importance of domains and conceptually distinguishes between relations (with fill) and relation variables (with structure only) [A1-A3]. However, a mapping procedure that results in the construction of relations is still needed. This is complicated by the fact that the object-oriented model and the relational model are really very different paradigms that do not always have a clear one-to-one mapping. Further complications arise because information contained in the implementation part of an object, which is hidden from public view, may need to be expressed explicitly in the relational model, thereby necessitating a deeper analysis than an examination of the public interface part.

In some cases, an object class can map to a domain, and in others, to a relvar-like entity but not to a relation. If the object class represents basic, atomic values, it can map directly to a single domain. However, by virtue of the principle of encapsulation, nothing in the object-oriented model prevents an object class from being quite complex, in which case the object class could map to a composite data type, consisting of two or more atomic data types.

Although this composite type lacks some characteristics of a relvar, it can be treated as a relvar precursor in which the domains, but not the attributes have been defined. Consider the following definition of complex domain D, which is a composite data type consisting of a collection of single, atomic domains, V:

\[
D = \{ V_1, V_2 \ldots V_n \}
\]

Note that D is analogous to relvar, R without attributes, A. In fact, the scalar data types that compose D can suggest attribute names without D actually being a relvar. The concept of a composite data type or complex domain plays a key role that leads to the generic relational data model which is the intermediate step in the proposed mapping scheme. Note that there is nothing to prevent "n" from being equal to 1, in which case D would degenerate into a scalar, atomic domain. D fits the description of the relvar-like entity mentioned above.

Although the domain of the relational world that corresponds to the object class of the object-oriented world, it is necessary to allow for the aggregation of data types to form complex domains. This is accomplished with the concept of data abstraction. Object classes that represent scalar data types map to atomic domains in the relational model, can be used in the construction of relations. Similarly, object classes that represent a composite data type map to complex domains. Relations can be constructed from these complex data types by choosing appropriate attribute name and data element values associated with each component domain. This is not the only way to form relations from complex domains because a relation also can take into account the information contained in message communications between
objects. Certain aspects of the message communication between objects in these classes may be expressed as relationships that also can be captured in relational format, if needed by the user. Messages can represent relationships between objects that may not be explicit in the object classes themselves, but that may prove useful if included in a relational design. Thus, message communications and also can be used to influence relvar structure. It is possible to express all aspects of the object-oriented model in terms of the relational model, either explicitly or implicitly.

APPENDIX A REFERENCES


AUTHOR BIOGRAPHIES

Dr. Marion G. Ceruti

NCCOSC RDT&E Division, Code 4221
53140 Gatchell Rd., San Diego, CA 92152-7464
Tel. (619) 553-4068, DSN 553-4068, Fax (619) 553-5136
INTERNET: ceruti@nosc.mil

Dr. Ceruti is a scientist in the C4I Systems Engineering and Integration Group of the Command and Intelligence Systems Division at the Naval Command, Control and Ocean Surveillance Center, Research, Development, Test, and Evaluation Division. She received her Ph.D. in physical chemistry, with emphasis on data acquisition systems, from the University of California at Los Angeles (UCLA) in 1979. While at UCLA, she was awarded a research fellowship from the International Business Machine Corp. Dr. Ceruti’s present professional activities include database development and integration for C4I decision-support systems, including the Joint Maritime Command Information System. She has served on the program committee and as Government Point of Contact for all annual Database Colloquia since 1987. An active member of AFCEA and several other scientific and professional organizations, Dr. Ceruti is the author of numerous publications on various topics in science and engineering, including information management.