BUSINESS OBJECTS AND THE CORPORATE METADATA REPOSITORY

Gregory J. Lestina, Jr., Daniel W. Gillman
Gregory J. Lestina, Jr., U.S. Bureau of the Census, Washington, DC 20233

Key Words: metadata, object-oriented, Java, ODBMS, database, CORBA, object model

Introduction

As the year 2000 approaches, organizations have been preparing for possible effects on their computer operations. Organizations also need to look beyond 2000 and perhaps understand what changes will take place in software development that will affect information processing for statistical and scientific systems. In 2005, for example, we must ask ourselves what new technologies will help us collect and display statistical data. How can we improve our existing software and computer systems to provide statistical information more easily and less costly? Certain leaders in the technology field believe that organizations will concentrate on developing network capacity and connections, accommodate the communications needs of mobile workers, and design more object-oriented applications to improve operations (Burden, 1999). If these areas prove successful in 2005 we may see more data collection being done using smaller hand-held computers and transmitted by wireless technology. Statistical processing software may be object-oriented, that is, software will need to be written once and then reused on a number of other applications. This paper discusses our experiences and possible impacts that object modeling and design have on the Census Bureau corporate metadata repository.

The Corporate Metadata Repository

The Census Bureau Corporate Metadata Repository (CMR) was conceived in 1994 and was implemented in 1997. It is a relational database that contains statistical metadata for Census Bureau surveys, censuses, and projects. It is a repository of statistical metadata, pointers to metadata (such as documents or images), and pointers to data. The CMR contains statistical metadata that describes survey designs, processing, analyses, and data sets for all the surveys the Census Bureau conducts. Using detailed models to organize the metadata allows users to find specific types of information or to compare similar kinds of information about surveys, documents, products, variables, and data sets. Generally, metadata is loosely defined as "data about data" or "information about information". Such a definition is somewhat imprecise and leads to confusion. The Census Bureau has therefore explicitly defined statistical metadata to distinguish it from statistical data (Gillman, Appel, 1997):

Statistical Metadata is descriptive information or documentation about statistical data, i.e. microdata and macrodata. Statistical metadata facilitates sharing, querying, and understanding of statistical data over the lifetime of the data.

The two types of statistical data (electronic or otherwise) are described as follows (see Lenz, 1994):

- **Microdata** - data on the characteristics of units of a population, such as individuals, households, or establishments, collected by a census, survey, or experiment.
- **Macrodata** - data derived from microdata by statistics on groups or aggregates, such as counts, means, or frequencies.

The extensive nature of statistical metadata lends itself to categorization (see Sumpter, 1994) into three components or levels:

- **Systems** - the information about the physical characteristics of the application's data set(s), such as location, record layout, database schemas, media, size, etc;
- **Applications** - the information about the application's products and procedures, such as sample designs, questionnaires, software, variable definitions, edit specifications, etc;
- **Administrative** - the management information, such as budgets, costs, schedules, etc.

Object Analysis

Object analysis is a way of modeling an organization or describing its business processes. Like relational modeling, a systems designer needs to first identify the business processes in the organization. Then the designer identifies what real-life or business objects exist in these processes. For example, the Census Bureau collects, analyzes, and disseminates data. The business objects that carry out these processes include surveys, data elements, questionnaires, datasets, and documents. A business object, such as a survey, may be created or deleted or updated. The act of creating a survey is an operation or method of a survey object. Below are some more precise definitions of terms used in object and relational design (Lestina, Gillman, Appel, 1998):

- **Object** – An object is "an identifiable, self-contained unit, which is distinguishable from its surroundings". It can be described by a set of *attributes* (properties) "and a set of operations which defines its behavior" (Dabrowski, Fong, Yang, 1990).

- **Business Object** – An object that models a real-world entity (Orfali, Harkey, Edwards, 1996).
- **Instance** – An occurrence of an object (Dabrowski, Fong, Yang, 1990).
- **Method** – A function, operation, or procedure within an object that performs specified actions. Methods send and receive *messages* (e.g., the name of the object and the name of the arguments being passed) which allow objects to communicate with one another.
- **Class** – A generic description of an object consisting of instance variables and methods (Dabrowski, Fong, Yang, 1990).

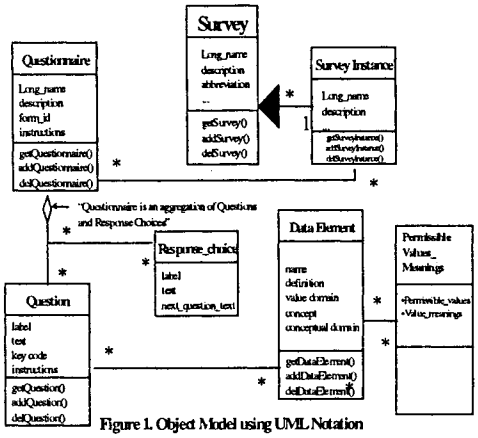

Figure 1. Object Model using UML Notation

- **Inheritance** – The ability for objects to acquire methods and attributes from one or more classes.
- **Table** – In a relational database, a table is a two-dimensional data structure that holds data. It is comprised of *rows* and *columns* (Cheu, Linden, 1990).
- **Row** – In a relational database, a row represents one occurrence of an entity (e.g., employee) represented by the table (Cheu, Linden, 1990).
- **Column** – In a relational database, a column represents one attribute (e.g., salary) of an entity.
- **Referential Integrity** – The property that ensures specific relationships between tables are maintained.

Object-oriented computer languages such as C++ or Java are designed so that a user can write code that implements the objects and classes in an object model. Object-oriented languages work in such a way that once an object is defined or created, the code can be reused in applications around the organization. That is, the code used to define a survey object used in one application can be used in another application or extended without extensive modification. By reusing objects, developers will find it much easier to develop applications that design tables and charts, produce datasets, and access detailed information.

Object-oriented design can be extended to include the idea of having business objects (e.g. a questionnaire object) being transmitted from system to system to interact with, say, a document object. These distributed objects allow computer applications to be accessed across operating systems and perform their assigned tasks no matter what platform or system the object is on. Because Web applications typically access many different platforms and many different databases, distributed objects would theoretically make Web development and access much easier. Advancements in this technology have been made possible by emerging standards such as CORBA (Common Object Request Broker Architecture), Microsoft's OLE/COM (Object Linking and Embedding/Common Object Model) and OpenDoc (Orfali, Harkey, Edwards, 1996).

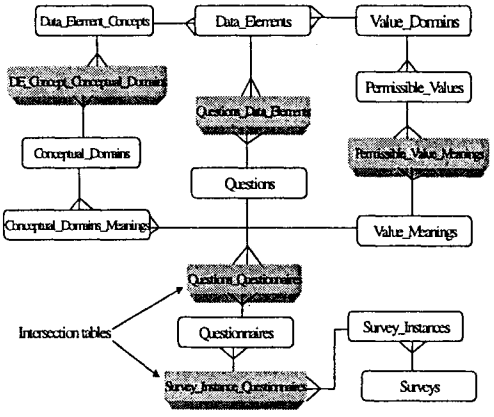

Figure 2. Relational Model Using Entity/Relationship Notation

Census Bureau Objects and Classes

The Census Bureau has redesigned a section of its Corporate Metadata Repository model from a relational design to an object design. The purpose of this redesign was to put object modeling into practice, write software based on the model, and then build a prototype application to demonstrate the utility of object-oriented methods. This prototype application creates survey, questionnaire, question, data element, and response choice objects and allows a user to browse through a Web application to obtain useful information pertaining to the relationships of these objects. For example, a user would be able to browse all the questions for a particular questionnaire and the data elements for a particular question on a questionnaire.

Our experience from creating a relational model for the CMR helped us in defining the classes, objects, and attributes in the object model (see Figure 1). For example, a survey is an entity in the relational model (Figure 2) and it is defined as a class in the object model.

By comparing the two models, there is a similarity in meaning between a table in a relational schema and a class in an object schema. Likewise, a column in the relational model is analogous to an object's attributes.

But this is where the similarities end. The relational model describes the world as two dimensional, that is, the attributes of real-world entities such as a Questionnaire are stored as rows and columns. The object model, however, describes real world entities such as Questionnaires as entities that have responsibilities. That is, not only are the attributes of a Questionnaire stored in an object model, but the methods describing the responsibilities of a Questionnaire are also stored. For example, in Figure 1, a Questionnaire object has responsibility for Question objects. Responsibility in this context means that a Questionnaire object defines what Question objects it accesses and therefore has control over the methods that access Question objects. Conversely, a Question object does not have responsibility for what Questionnaires it is associated with, but does have responsibility for the Data Elements it accesses. A separate relationship may need to be created for Question objects to access Questionnaire objects. The object model implements these relationships by using pointers directly from the object-oriented programming language. The relational model implements these types of relationships by the use of foreign keys, intersection tables, and table constraints through SQL.

There are other implications that can be derived from modeling the CMR as business objects. A document object, for example, can be thought of as a composition of word objects and heading objects. These components can be used to perhaps create new documents electronically or to classify existing documents by just accessing the components. By using components, a questionnaire can be designed electronically by obtaining the correct question and response choice objects and adding them to a template. Similarly, publication tables and datasets can be created by implementing data element objects.

Obvious differences between the models include the fact that the object model has no intersection tables and therefore allows many-to-many relationships between classes. In object modeling, many-to-many relationships are handled through the programming language rather than in a database. Object-oriented languages allow the developer to put instantiated objects from the same class in a "collection" or a logical grouping of objects (similar to arrays in C language). Java or C++ objects use collection classes such as Vector, Hashtable, and Dictionary to store and index a collection of objects. Storing collections for future use may be tricky if using a relational database. The developer would have to add SQL code to his or her application to store the objects. The attributes of each object would need to be mapped to the rows and columns of the database.

The easiest way for objects to be stored in a database is through an object database management system (ODBMS). Instead of having to take objects apart to store them, objects are stored using a traditional object-oriented language. There is no intermediary language needed such as SQL. Objects are made "persistent", that is, they reside on the database server until someone wishes to delete them. To access objects, developers use the methods in the collection classes in the traditional object-oriented language. Such programming eliminates the extra programming effort of accessing data through SQL using complicated table joins. An ODBMS provides tools for managing and viewing objects in a database. They also provide query classes for accessing objects' attributes. ODBMS's provide many of the administrative features of relational databases such as transaction protection, performance issues, referential integrity, and backup and restore capabilities. The Object Database Management Group (ODMG) is the standards organization that administers ODBMS's. The ODMG-93 standard suggests ways that ODBMS vendors can define their databases to have common functionality.

Distributed Objects

One of the intriguing things about business objects in an object-oriented environment is the promise that business objects can access other objects across networks and participate in applications on different platforms. This means that a computer application doesn't need to know what operating system it is working with or what language an object was written in. Users can theoretically access applications without seeing what is going on in the background. Therefore, it is possible that data from non-object-oriented language such as COBOL could access business objects from a metadata repository application written in Java. COBOL applications that provide data entry could communicate with C++ programs that produce reports and store datasets. CMR applications could access large amounts of legacy metadata on separate computer systems without asking people to rewrite or convert code.

To achieve such interoperability, a standards organization called the Object Management Group (OMG) has been working on the problem. The OMG has over 500 members (Orfali, Harkey, Edwards, 1996) and is the largest standards organization in the world. It has been working on the Common Object Request Broker Architecture (CORBA) since 1989 to allow business objects gain access to legacy information on different systems in different programming languages. CORBA defines at least 16 computing services that provide a common way for applications and objects to communicate (Orfali, Harkey, Edwards, 1996). Products that provide CORBA-compliant middleware (additional software that allows the client and server to communicate) must follow strict rules about how applications are prepared and how objects are transmitted in a CORBA environment.

Currently, there is commercially available CORBA-compliant middleware for several languages including Java, C++, and COBOL. That is, by using CORBA middleware, we would be able to access

applications written in Java, C++ or COBOL without being concerned about what platform the applications are on.

For example, suppose we are interested in obtaining information from a dataset created by an application written in COBOL. We could of course log on to the system that has this information and run the application that creates this dataset. But more often a user or researcher who needs this information does not know where it is or how to get it. The user is more concerned about getting the information rather than knowing how or where it was obtained. To access information from an existing COBOL application residing on a remote server, CORBA middleware can be used to make calls from an application (written in, for example Java) on the client computer which has the ability to access the COBOL application on the remote server. Such interoperability allows existing applications to communicate with one another, making data access, in theory, much simpler. This section of the paper gives a general description of the components of CORBA middleware and how it may work in a statistical application.

CORBA middleware consists of an Object Request Broker (ORB), Common Object Services, and Common Facilities that provide the interoperability of your business objects. Below are definitions of some basic elements of the CORBA architecture (Orfali, Harkey, Edwards, 1996):

- **Client** – a computer that makes requests to or consumes a service on another computer.
- **Server** – an application or computer that shares resources or provides services to other computers.
- **Interface Definition Language (IDL)** – a subset of C++ that defines objects, attributes, methods, and parameters in a client/server application. It is a specification language only and only references compiled code.
- **Stub** – an application written in IDL that defines how the client sends requests and invokes methods on the server application.
- **Skeleton** – an IDL application on the server that processes and returns requests from the client application.
- **Object Request Broker (ORB)** – software or middleware that receives messages from a stub application, determines a server application that can process the messages, and sends the messages to the corresponding skeleton application.
- **Common Object Services** – IDL applications that assist and complement the functions of the ORB.
- **Common Facilities** – IDL applications that assist and complement the business objects.
- **Interface Repository** – a built-in database that contains metadata that describe stubs, skeletons, and implementations.

Probably the most important aspect of the CORBA architecture is the Interface Definition Language (IDL). It is a computer language that looks very similar to C++ and specifies the interfaces used in a CORBA application. To implement CORBA services, the programmer creates interface files that describe the methods and objects used in the entire application. The programmer then compiles the IDL descriptions to create the client stubs and server skeletons. Stub code allows a client application to send calls or arguments to CORBA objects on the server. The skeleton code creates CORBA objects that allow the client and server applications to communicate. IDL, then, is the standard language used to allow applications written in different languages to communicate on remote servers.

To build a CORBA application that accesses data elements from a dataset produced from a COBOL application, the programmer would first need to create an interface definition file. The interface definition file only gives a description of the classes and methods used in the CORBA application. Below is a sample interface definition file that may be used for accessing certain data elements from a dataset:

```
module metadata {
        interface data_element {
                String getName();
        };
        interface dataset {
                String getDatafile();
        };
};
```

The programmer then needs to create the code for the client application and the server application. The server application contains the code for executing the original the need of the user, accessing datasets in this case. The server application also needs to implement the ORB and begin communication with the client computer. Therefore, a call that instantiates the skeleton classes is required in the server application. The client application contains code for sending calls to the server object (calling getName() or getDatafile() in the above example). It also instantiates the stub classes which initializes the ORB and establishes the communication network between the client and the server. Figure 3 shows the elements involved in compiling and executing a CORBA application (Fowler, 1999 and Orbix 3 White Paper, 1999).

Probably the most important part of the development process occurs when the client and server applications are compiled. The compilation process creates the stub and skeleton files for the

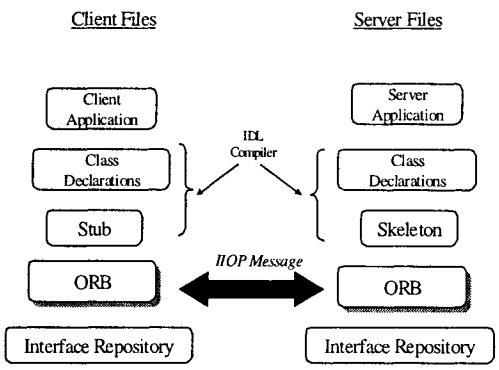

Figure 3. Sending Messages Using CORBA

client and server applications. Stubs and skeletons are created on the client machine. The skeleton files are then copied to the remote server. Once all the applications are created on both the client and server, all the applications need to be compiled and linked together. If they are linked successfully, the ORB service needs to be started on both the client and server. The client then should access the server application. This allows the data element and dataset objects to be transmitted from the metadata application (client) to the COBOL application (server).

Conclusion

Many statistical organizations have spent much time and resources designing databases and repositories that hold their data and metadata. These databases communicate with applications that have been developed years ago and are written in languages that are rarely used to develop in today. It can be very costly for a department to convert software written in COBOL to Java or C++. Not necessarily because of the man-hours to do the work, but the risk involved in not converting the code to precisely to meet computing requirements or the risk of not having the experience to make all the changes. For very large production systems, it is probably very difficult to convert a relational system to an object-oriented system.

But in large organizations, it is imperative that legacy applications communicate with applications currently being developed. OMG's CORBA standard offers an opportunity to communicate with legacy applications written in COBOL or C++ or Java using object-oriented technology. Organizations have an opportunity to explore an alternative to converting their legacy applications. It may be a few years before CORBA products mature and the technology penetrates the marketplace. But in 2005, our need for object-oriented applications will increase due to our need to move business objects across the Web.

References

- Burden, Kevin, "Tomorrow's IT: The Technology", Computerworld, April 12, 1999, pp. 68-69.
- Cheu, D., Linden, B., SQL Language Reference Manual, Version 6.0, Oracle Corporation, 1990.
- Dabrowski, C., Fong, E., Yang, D., Object Database Management Systems: Concepts and Features, U.S. Department of Commerce, National Institute of Standards and Technology, Special Publication 500-179, April 1990.
- Fowler, Kim, Java for Enterprise Systems Development: Hands-On (Course Notes), Learning Tree International, Inc., 1999, pp. 472-5-1 to 472-5-40.
- Fowler, Martin, UML Distilled, Addison Wesley Longman, Inc., Reading, Massachusetts, 1997
- Gillman, D., Appel, M., Building a Statistical Metadata Repository, Second IEEE Computer Society Metadata Conference, Silver Spring, Maryland, September 16-17, 1997.
- Lenz, H. J. (1994), "The Conceptual Schema and External Schemata of Metadatabases", Proceedings of SSDBM-7, pp160-165, Charlottesville, VA, September 28-30, 1994.
- Lestina, G., Gillman, D., Appel, M., The Role of Object Databases in Accessing Metadata at the Census Bureau, Joint Statistical Meetings, Dallas, Texas, August 10-14, 1998.
- Martin, J., Odell, J., Object-Oriented Methods: A Foundation, PTR Prentice Hall, Englewood Cliffs, New Jersey, 1995.
- Orbix 3 White Paper, Iona Technologies, PLC, April 1999.
- Orfali, R., Harkey, D., Edwards, J., The Essential Distributed Objects Survival Guide, John Wiley & Sons, Inc., 1996, pp. 47-105.
- Sumpter, R. M. (1994), "White Paper on Data Management", Lawrence Livermore National Laboratory document, 1994.