Modeling Your Business With Objects

Executive Summary

Enterprise Objects Framework builds on NeXT's leadership in object-oriented software and redefines how custom business applications are constructed, setting a new benchmark for the capability and scalability of object-oriented systems.

Using the Enterprise Objects Framework, programmers can build business-centric information systems by defining custom objects that tightly integrate business information and the logic associated with managing or processing that information. The Framework seamlessly extends the capabilities of NEXTSTEP and NeXT's Portable Distributed Objects (PDO), enabling developers to create highly reusable business objects that store data in relational databases.

Enterprise Objects simplify the process of building complex business information systems because it eliminates the need to duplicate data processing code in multiple application modules. The Framework enables programmers to create business objects that incorporate business process logic once and use those objects to manage enterprise-wide operations, even when the data is represented differently in different divisions of your organization.

Applications created using Enterprise Objects Framework can also incorporate customizable, vertical industry business objects supplied by industry specialists. By simply mapping the data elements of Enterprise Objects to the data structures used to store information, programmers can take advantage of pre-defined business processes designed to properly process business information. Programmers customize only those processes that require modification to meet special requirements.

This white paper first defines business objects and provides an overview of the technology used to incorporate these objects in enterprise information systems.

I. Defining Business Objects

Developing information systems that meet complex business requirements has never been easy. Numerous methodologies and modeling techniques have been introduced for capturing requirements, analyzing the business and building business applications. In theory, all of these methodologies and modeling techniques have one common goal: simply and accurately model real-world systems without complexity in order to develop applications that mirror an organization's business.

However, modeling large, complex systems in practice is an extremely challenging proposition. Many methodologies partition the entire system into domains, or subsystems, similar to the divisions of a large organization. Each subsystem is further decomposed into a data model, a process model and other functional models. While developing separate functional models for a complex system can be useful in implementing software systems, the approach does not deliver a simple, high-level representation of the system or the organization as a whole.

An alternative approach is object-oriented analysis of real-world systems which results in a model that integrates both data and process. Object-oriented analysis involves defining objects as entities of importance that can be described in terms of *attributes*, or characteristics of importance and *processes* which they may respond to or initiate.

Object-oriented analysis and design techniques have been successfully applied to build complex, real-time applications such as navigation and guidance systems and telecommunications switching systems. The delivery of these complex applications has been attributed in part to the use of objects, both in modeling the system and in the implementation of discrete system components.

Modeling a Customer Service Operation

Object-oriented analysis and design techniques can also be successfully used to develop business information systems. Business objects, which tightly couple business information and the processes that are required for properly managing that information, can be used to model the business and its operation.

The resulting business model is especially useful for understanding and communicating how the business operates. It will also be an important design specification for building business applications.

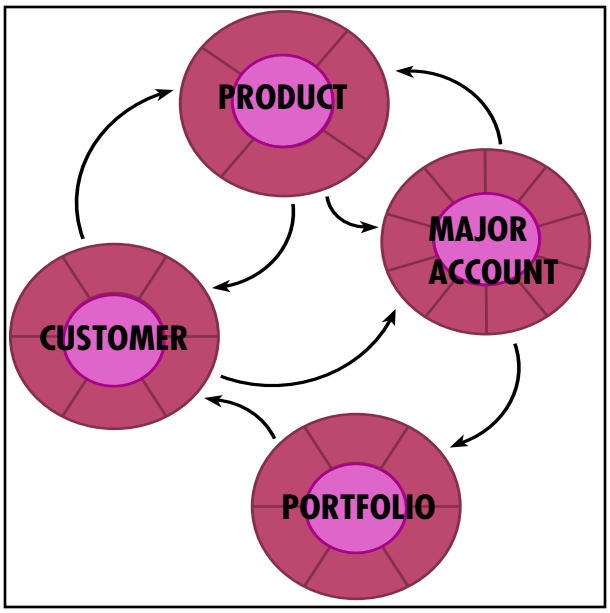

Figure 1

Just as objects simplify the complexities of the real-world, they simplify the process of implementing the development of business applications. The unification of the business model and its physical implementation through business objects creates enormous opportunities for reducing the time and cost associated with implementing custom applications.

To demonstrate the simplicity of the object-oriented approach, a typical customer service operation is modelled in Figure 1. The model is composed of the following business objects:

- *Customers*: characterized by name, shipping and billing addresses, credit card account number, customer type and phone number. Customers processes include information updates and establishing credit limits.
- *Major Accounts*: a subtype of customers which are characterized by the same attributes and which perform the same processes. However, the algorithm employed to determine credit limits for major accounts is different than that used for ordinary customers.
- *Products*: described by product codes, descriptions and terms under which products may be traded. Products respond to requests for current yield, risk, and other evaluation criteria.
- *Portfolios*: describe the set of products owned by an customer. Portfolios respond to requests for current valuation and requests to buy or sell.

This simple example, demonstrates how the object-oriented concepts of encapsulation, inheritance and polymorphism reduce the complexity of the business model by removing unnecessary details, reusing common object definitions where possible and enabling process definitions to be overridden when necessary.

1. *Encapsulation* means the implementation details are hidden (i.e. that data structures used to store customer information and the details of processes need not be revealed.)
2. *Inheritance* means that the subtype of an object shares the same data and process definition as its parent unless those definitions are overridden.
3. *Polymorphism* means that a subtype can override its parent's definition of a particular process and that it responds differently than its parent to the same process request.

This model of the business enables analysts, users and managers to communicate and understand the most important aspects of the operation because it represents business entities in a form that naturally combines information and process.

Implications of the Object Model

Only after the business model has been defined, should business objects be implemented. During the design and implementation of the application, the business model acts as the blueprint for building application components. Externally, it will appear that each of the components behaves the same way its real world counterpart behaves. Internally, however, the implementation of these objects requires detailed knowledge of process logic and data storage and retrieval. For example, customer information might be stored in one or more tables of a relational database. It will become necessary to understand and implement the business logic for product pricing, credit approval and other business processes performed by these business entities. If the actual business information system is deployed in a client/server architecture, considerations for how the objects will be deployed on the network will also be required.

Business objects minimize the amount of code required to construct applications

Applications are typically composed of screens, often referred to as forms. These forms display business information and allow users to directly manipulate data. The generally accepted approach for incorporating business process into an application is to implement the logic as part of the application interface. For example, a special process for updating customer information would be programmed into each and every form that would allow customer updates to occur. However, this creates a costly problem. Every copy of the customer update process would require modification if the customer update process needed revision.

Business objects can minimize the amount of code required to construct applications by localizing business information and the process logic in the business object itself. Application screens are merely a means for viewing specific attributes of particular entities and for capturing user input in the form of action requests. For example, a customer contact screen would display and allow updates to customer information and might allow the user to click a button that causes the customer object to establish a credit limit based on credit available using his credit card.

Any number of screens could be created that display customer information and none of the screens would require any programming to be able to update displayed customer information or to request a line of credit. Users could initiate these business processes from within any screen that allowed the user to send an action request to the object.

This represents a fundamental departure from the generally accepted approach used today which requires that all process logic be physically implemented as a part of the interface.

Moreover, the business model spans application boundaries. After a customer object has been defined and incorporated in the customer service application, it can be reused in another application. Because all applications make use of a common business model, business objects can easily be reused.

Business objects encapsulate data storage and retrieval

In addition to physically implementing business logic as a part of the application interface, data storage and retrieval is typically coupled directly to the interface. Specifically, application screens that display customer data directly reference the tables and columns containing that data in a relational database.

These direct references have serious implications for the long-term maintenance of applications. Not only is maintenance difficult, but very little, if any of the application can be easily modified and reused. For example, changes to the underlying database structure require modifications to all screens that display information from the modified tables, including all of the associated processes. For example, any screen displaying the data element PART_NO from a table named PRODUCTS, would require interface modifications if the PART_NO data type was changed from CHAR(8) to CHAR(12). A simple change to extend part numbers also affects every process in the entire application that involves part numbers.

The business processes that are implemented within an application are tightly coupled to a single data structure. As a result, it is nearly impossible to construct, for example, a single health care records management system that provides a consistent set of functionality for multiple HMOs unless each HMO is willing to convert their data to a common format.

An object-oriented approach can be used to solve these problems. By localizing data storage and business processes within objects, rather than propagating copies of data and process to application screens, the resources required to implement enhancements or modifications is drastically reduced.

The business objects themselves can be used to insulate the internal processes from changes in the external data structures used to store data. Each instance of a business object contains properties, or data, that characterizes that particular instance (e.g., each customer has a name and phone number). All of the business processes that are initiated by the customer object references only those attributes; these processes never need to reference the external location of data. An external service, or technology, can be employed to provide data storage and retrieval facilities for moving data between business objects and the external data source.

By delegating the data storage and retrieval function, the business objects provide a standard business process through applications that access data stored in different formats.

Business objects can be deployed anywhere

Business objects are location independent, meaning they can take advantage of their environment to perform more effectively. Just as a shipping and receiving function might be optimally performed in a large warehouse rather than an office setting, applications composed of business objects will likely provide better performance if computationally intensive operations are performed on large servers.

Applications developed using procedural languages and 4GL tools typically have little or no flexibility in distributing business processing. By tightly coupling the process with the application screens, the process logic must usually be performed locally on the client workstation. While it is possible to use remote procedure calls and other mechanisms to distribute some expensive processes, this approach is difficult.

An object-oriented approach using business objects is considerably less complex. Business objects communicate with each other by sending messages that trigger business processing, much like a customer picking up a phone to call in an order. The customer might be far away from the office where the order is placed and the office might be far away from the warehouse where the order is filled. Similarly, an application composed of customer objects, order objects and product objects could distribute these objects across a network and do so without the user of the application knowing where the objects are actually performing their processing. More importantly, the application developer can implement the business objects without concern for how the objects might be deployed at runtime. Developers have complete flexibility in configuring the distribution of objects.

Applications making use of business objects can dynamically configure where the business objects are deployed without requiring any special program modifications. This capability enables applications to take advantage of networked resources to perform computationally intensive processes simply by relocating a business object.

II. Enterprise Objects Framework

Enterprise Objects Framework, the second generation database application development framework available from NeXT, is unique in the industry in its ability to define and use business objects.

Components of the Enterprise Objects Framework

Enterprise Objects Framework consists of a number of components that support the definition, implementation and distribution of business objects.

Enterprise Object Modeler Application

This application allows developers to create both the database schema and the business model for an application. Enterprise Object Modeler allows an analyst to define associations between properties of the business model and the underlying entities and attributes which will be used to store the data. For example, the *Customer* Enterprise Object might have properties (such as *name, address, credit limit*) which are actually stored in a relational database tables *Customers* (in the fields *CUST_NAME, CUST_CREDIT_LIMIT*) and *Addresses* (in the fields *ADDR_STREET, ADDR_CITY, ADDR_STATE).* Both the object model and the data model are then used to dynamically transform requests for data into explicit data manipulation commands.

The business model is extensible and provides a public API that allows developers to access extensions to the model. The model can be referenced or dynamically modified by the application at runtime. Defining alternate associations between a single business model and different database schemas allows applications constructed using business objects to consistently process information stored in different data sources or data structures.

The non-proprietary format of this model also allows developers to use their preferred design and analysis tools. Enterprise Objects Framework is designed to enable development of robust database application development tools, integrate design information from existing CASE repositories and enable creation of end-user data access products.

In addition to allowing analysts to create business models and database schemas, the Enterprise Object Modeler application provides utilities to create a default schema from an existing database and a default object model based on the database model. These utilities help to quickly build new applications that access existing databases by creating default models which analysts can quickly refine.

User Interface Control Framework

It is impossible to create all the possible views of customer information. However, it is possible and highly desirable to define a single set of business processes for managing customer information, regardless of how the information is being viewed.

The UI control framework is designed to provide enormous flexibility in the visual presentation of data representing a business object. The process and data is fully encapsulated within the business object, which uses basic messaging facilities to pass information to the user interface.

Object persistence

A common problem facing application developers is that most database development tools build applications that fully depend upon the underlying schema. Modifying the underlying table structure directly affects the application. Enterprise Objects Framework insulates the application from the underlying data structure. Changes to the database schema can easily be accommodated simply by remapping the properties of Enterprise Object classes to alternate attributes of the database schema.

The persistence framework makes use of information contained in the business model to dynamically generate default DML (data manipulation language) required to access or modify data in an external data source. As a result, developers are never required to explicitly define data access mechanisms within a business object. This facility completely insulates business objects from the underlying data structures used to provide data storage.

However, the persistence framework can be tuned to improve application performance by providing programmers with the ability to intercept and modify default DML and replace it appropriately. Developers can also employ stored procedures and alternative locking strategies or transaction models to improve performance of the persistence mechanism transparently to business objects.

Concurrent Access to Multiple Data Sources

Enterprise Objects can access data simultaneously from more than one data source. In addition, by taking advantage of the dynamic binding capability of the runtime environment, Enterprise Objects can fetch data from one data source and store it in another, if necessary.

Seamless Distribution Capabilities

Enterprise Objects look and feel just like other NEXTSTEP objects and they have the ability to be deployed on a heterogeneous set of platforms just like other NEXTSTEP objects. The fact that they have the ability to store information in a relational database is irrelevant. However, this capability sets Enterprise Objects apart from objects that can be created in other development environments.

Portable Distributed Objects (PDO) bring NeXT's Distributed Objects technology to other operating systems such as HP-UX, SOLARIS, OSF/1, and AVIION, facilitating the development of server-based applications that make use of objects that can send messages to objects on other computers.

Enterprise Objects Framework runs transparently on PDO and allows server-based applications to access external data sources. Enterprise Objects can both access and process information anywhere on the network. This capability provides developers extraordinary flexibility in building applications that make most the effective use of resources on the network.

III. Summary

Object-oriented technology, including modeling techniques and programming environments, have long been promised as a means for building more robust business information systems more quickly.

Business objects, which tightly couple business information with the processes required to correctly manage that information, provide a useful construct for modeling business requirements. Fully understanding this construct and its implications for technologies required to support the concept makes possible the development of valuable technologies for building business systems that reflect a business model.

Enterprise Objects provides fully enabled business objects that leverage NeXT’s core object technologies including Portable Distributed Objects and Distributed Objects. Portable Distributed Objects and the Enterprise Objects Framework programmers can more easily define, implement and distribute business objects, setting a new benchmark for the capability and scalability of object-oriented systems.