Enterprise Objects™ Framework

A Second Generation Object-Relational Enabler

Charly Kleissner
Director of Software Engineering
NeXT Computer, Inc.
charly_kleissner@next.com

Introduction

Today's information system executives desperately need to improve programmer productivity and reduce software maintenance costs. They are demanding flexibility in frameworks and architectures in order to meet unforeseen changes (see *[Yankee 94]*). Adaptability is a major requirement of most company's information systems efforts. Management of change is one of the key computing concepts of the 1990s.

Object-oriented tools and development frameworks are starting to deliver the benefits of increased productivity and flexibility. These next-generation products now need to be combined with relational databases to leverage investments and facilitate access to business data. *Object-Relational Enablers* automate the process of storing complex objects in a relational database management system (see *[Aberdeen 94]*).

The *Enterprise Objects Framework* product is a second generation product bringing the benefits of object-oriented programming to relational database application development. Enterprise Objects Framework enables developers to construct reusable business objects that combine business logic with persistent storage in industry-standard relational databases. Enterprise objects are first class citizens in the NEXTSTEP and OpenStep developer and user environments. They can be geographically distributed throughout heterogeneous servers within an enterprise using the *Portable Distributed Objects* product (see *[NeXT-DO 94]*).

In this extended abstract we first describe the enterprise object distribution model and then give a brief synopsis of how relational data is mapped into objects. We then present an outline of the system architecture, explain how objects are mapped to multiple tables, and summarize the transaction semantics as well as the application development lifecycle. We conclude with an outlook on future development.

Distribution Model

Enterprise Objects Framework maintains a clear separation between the user interface, business objects, and the database server. Business policies are implemented as methods on business objects as opposed to being integrated in screen definition code of 4th generation language products or being stored away as stored procedure code in database products. Therefore business policies and objects can be subclassed and re-used in different scenarios.

The user interface usually executes on a client machine, the database server on a server machine. The business objects may execute either on the client, on the database server, or on a separate application server. This flexible, three-tier client / server architecture permits developers to build robust, scalable, enterprise client / server applications. Objects at each of the three tiers can be deployed to take advantage of network resources. Sophisticated load balancing based on networking traffic and other resource usage patterns becomes feasible.

The designer does not need to make the distribution choice at design time. The object messaging system makes the location of all objects transparent to the services they provide, i.e., application code does not need to be changed to redeploy objects to different platforms (for details see *[NeXT-DO 94]*). Design decisions are separated from deployment issues. Enterprise objects - just like any other object - can be run anywhere on a network and decisions about where objects run can be made at execution time.

Object Model

The most significant problem that developers face when using object-oriented programming languages with SQL databases is the difficulty of matching static, two-dimensional data structures with the extensive flexibility afforded by objects. The features of object-oriented programming concepts such as encapsulation, inheritance, and polymorphism and their associated benefits like fewer lines of code and greater code reusability are often negated by the programming restrictions that come with accessing SQL databases within an object-oriented application.

In his recently introduced "Third Manifesto" (see *[DD95]*), Date faults the relational database vendors for not correctly implementing the relational concept of "domain" which he falsely equates to "object class" (also see *[Date 95]*), thereby mixing up user-defined data types with object classes. There is - of course - no one-to-one mapping between an object class and any single relational concept (like domain or relations) since objects contain not only data but also methods and adhere to an object model which defines - among other things - the semantics of inheritance and polymorphism. It does, however, make sense to define the mapping between the data portions of the object model and the relational model.

The Enterprise Objects Framework provides a conceptual bridge between a mature object model as described and implemented in OpenStep (see *[NeXT-OpenStep 95]*) and existing relational database products by taking the more practical approach of allowing data of multiple base relations to be mapped to an enterprise object. Enterprise objects - like any other object - also contain logic, algorithms, and methods. These methods represent business policies and procedures. Only part of an enterprise object's data may be stored in a relational database management system, other parts may not require the same level of persistence or may be stored in different storage engines.

Data within an enterprise object are not constrained by a single table or a domain specification within a relation. Its data mapping can extend across tables and across physical databases. Furthermore, the mapping of an enterprise object to the database can be dynamically controlled at run time. In order to adapt an enterprise object to a different database or a different database schema, only the mapping needs to be changed, not the application.

Object technology improves developer productivity and simplifies maintenance by enabling applications to be built out of reusable, pre-built components. Relational database systems have the reliability and performance needed to build large-scale, multi-user decision-support and production on-line transaction processing applications - capabilities still unavailable from most object database systems. Enterprise Objects Framework provides a flexible mapping between the two models. This is accomplished through the *Enterprise Object Modeler* tool which allows developers to specify the mapping between the relational schema and the entities and their relationships.

System Architecture

The architecture of the Framework is divided into two major layers, the interface layer and the access layer (see Figure 1). The interface layer provides a mechanism for displaying data, while the access layer provides data access methods to storage engines. The layers are separated by a data source protocol which provides a generic object interface to data. The information which describes the mapping between the relational world and the objects is stored in model files.

In this section we briefly describe the data source, the two layers, and the model.

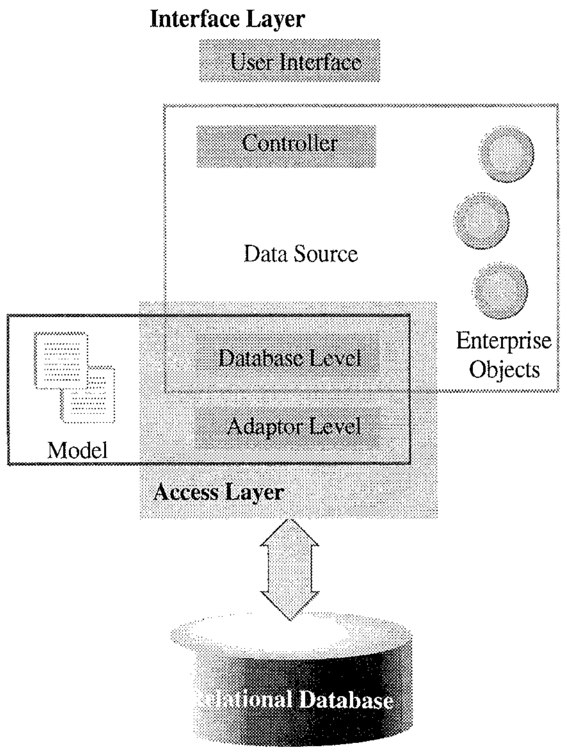

Figure 1. System Architecture

Data Source

A data source is an object that has the ability to fetch, insert, update, and delete enterprise objects. It is the means by which the interface layer accesses stored data; from the perspective of the interface layer, how data is stored (whether in a relational database or a flat-file system, for example) is of no consequence. The interface layer interacts with all data sources in the same way.

Interface Layer

The interface layer contains a controller and the user interface objects themselves. The controller coordinates the values displayed in the user interface with its enterprise objects. The user interface objects display data from enterprise objects.

Access Layer

The Framework currently provides an implementation of an access layer which conforms to the data source protocol and which provides access to relational storage engines. Access layers to other storage backends (e.g., flat file systems, object oriented database management systems, etc.) may be plugged into the infrastructure provided by the Framework. Note that the classes in every component in every layer have public interfaces and may be subclassed, adapted, and extended by the customers.

The access layer itself is comprised of two components: the database component and the adaptor component. Adaptors provide data access to a particular relational database server. Adaptors for Sybase and Oracle are bundled with the product. Adaptors for other database products (e.g., DB2, Informix, InterBase, etc.) are provided by third parties. The database component provides the translation from relational data into an object graph. It implements transaction semantics on the object level.

Model

The access layer uses models to define and resolve the mapping between enterprise objects and database data. A model defines, in entity-relationship terms, the mapping between enterprise object classes and a physical database. A model maps an enterprise object and its properties to an entity and the entity's attributes and relationships, which in turn map to the tables and columns in a database. While a model can be generated at run time, the most common approach is to use the Enterprise Object Modeler tool to create models that can be stored as files and added to projects.

Mapping Objects to Multiple Tables

The model is comprised of entities and attributes. An entity maps to an object class and to at least one table of the database management system. An entity contains either simple or derived attributes. A simple attribute maps to a database column. A derived attribute is a combination of other attributes or a computed attribute and does not directly map to a database column. Derived attributes are read-only.

A relationship creates a link between entities of the model. Relationships allow accessing data in a destination table that relates to data in a source table. A relationship creates a path that is traversed to resolve the relationship. Neither the object classes nor the data source need to be aware of the traversal path. The path is traversed as needed during model definition and at runtime.

A relationship can be used to flatten an attribute or flatten a relationship. A flattened attribute is an attribute of one entity that is added to another entity. Flattening an attribute is equivalent to creating a joined column; it allows the creation of objects that extend across tables. For each relationship, the developer may specify certain properties like the type of join (i.e., inner join, right outer join, left outer join, or full outer join) or the operator to be used to perform the join.

A flattened relationship is created by the elimination of intermediate relationships between two entities. Flattening a relationship gives a source entity access to relationships that a destination entity has with other entities. It is equivalent to performing a multi-table join.

Relationships can be either uni-directional or bi-directional. A uni-directional relationship has a single traversal path that has a source entity and destination. A bi-directional relationship has two traversal paths. It is created using an auxiliary entity. A reflexive relationship can be created using a single entity.

Uniquing and Faulting

The concept of *object uniquing* ensures that a row in the database is associated with only one object in the application. Uniquing of enterprise objects limits memory usage. Without uniquing, the system would create a new enterprise object every time an application fetches its corresponding row, whether explicitly or through resolution of relationships. Uniquing prevents redundant storage and the associated problems of consistency. It allows the application to know with confidence that the object it is interacting with represents the true state of its associated row as it was last fetched by the application. The database component performs uniquing of enterprise objects based on the primary key information from the database by default, but the developer can turn this behavior off if needed (e.g.,to implement a different uniquing scheme).

The Framework automatically resolves relationships defined in a model. It does so by delaying the retrieval of data until the data is actually needed - a concept referred to as *faulting*. This delayed resolution of relationships occurs in two stages: the creation of a placeholder object (fault object) for the data to be fetched, and the fetching of that data only when it's needed. When the database component fetches an object, it examines the relationships defined in the model and creates objects representing the destinations of the fetched object's relationships.

Transaction Semantics

The Enterprise Objects Framework uses *snapshots* to implement flexible and robust concurrency control mechanisms. A snapshot is a dictionary object which records the primary key and the attributes which are used for locking. A snapshot is recorded under the id of its enterprise object whenever the object is fetched or modified. An update strategy determines how updates should be made in the face of concurrent changes by others. The default update methodology is optimistic locking which assumes that the data won't be changed by others, but checks this assumption before writing changes back to the database. The database component performs this check by comparing the data in the snapshot with the data in the database. An exception is raised if differences for the relevant attributes are detected. The Framework also provides for pessimistic locking and no locking (i.e., write back dirty data independent of conflicts). The user may also declare that there are no updates; in that case the system does not take any snapshot at all.

The Framework provides a flexible buffering mechanism that determines when changes in the user interface are actually applied to the enterprise objects. The system keeps a stack of undo operations which may be applied before the changes are actually committed to the database. The buffering mechanism coupled with the concurrency control options provide flexible and powerful transaction capabilities. *[NeXT-EO 94]* contains a more thorough and comprehensive description of some of these concepts.

Development Lifecycle

The Enterprise Objects Framework supports the following three methodologies of developing applications: bottom-up, top-down and a hybrid mixture of the first two.

Bottom-up

In the pure bottom-up approach a development team starts out with existing business data stored in a relational database. It then uses the Enterprise Object Modeler tool to create a default entity / relationship model based on the relational schema. Developers may change the default entities, their relationships, and attributes to more accurately reflect design goals. They then relate entities to enterprise object classes, provide data mapping information and associate appropriate methods with the enterprise objects. Developers may chose to reuse (e.g., subclass, override) existing methods or develop new ones to accomplish their goals. Note that the methods which are associated with enterprise objects usually do not contain any user interface code or SQL code.

In the final stage of development the developers associate appropriate user interface widgets to their enterprise objects and deploy them on the appropriate platforms. Thereafter business policies may be changed or enhanced without having to touch the user interface or the mapping to relational database engines. The converse is true as well: developers may change the mapping to a different database engine or a different user interface without having to recode the business policies.

Top-down

In the pure top-down approach a development team starts out using an object oriented analysis (OOA) or object oriented design (OOD) tool of their choice. Most current tool vendors provide support for a variety of object oriented design methodologies (see for example *[Booch91]*, *[Jacobson92]*, *[RBPEL91]*, and *[SM88]*) as part of their toolset. The output of this process is then the input to the enterprise object class definition. Based on that definition, the Enterprise Object Modeler tool will assist in creating an appropriate entity / relationship model and its associated relational database schema. The appropriate business data will have to be loaded into the database before the enterprise objects can be deployed on the appropriate platforms.

Hybrid

The hybrid approach will allow iterations in the design process that lead to continuous process optimization based on feedback loops.

Dataflow

Using the bottom-up approach, data flows as follows between the various layers in the system: Data comes into the access layer from a relational database in the form of rows. The adaptor component packages the raw data as dictionary objects which contain key-value pairs; each key typically represents the name of a column, and the key's value corresponds to the data for the column in that particular row.

The database component creates enterprise objects from the dictionaries. The enterprise objects' properties get their initial values from the corresponding keys in the dictionary. An enterprise object typically adds behavior to the data it receives from a dictionary. The enterprise objects pass from the access layer into the interface layer through a data source, which supplies them to a controller.

The controller transports data from the enterprise objects to the user interface, where data is represented as values. Controllers coordinate the values displayed in the user interface with the corresponding enterprise object values. When enterprise objects are modified, the controller tells the data source, which is responsible for propagating changes to the database. The movement of data in the Framework is bidirectional.

Conclusion

The Enterprise Objects Framework extends and enhances the object-oriented software environment of NEXTSTEP and OpenStep as well as the capabilities of relational database management systems. It provides the framework for quickly deploying object-oriented business applications on a wide-scale basis. In combination with the Portable Distributed Objects product it allows enterprise wide deployment of business applications. Its flexible three-tier client / server architecture enables enterprise wide scalability. Its open architecture allows for easy customization and extension to non relational data storage engines. It provides a robust and flexible infrastructure for ad-hoc as well as production workflow applications. Its integration with the NEXTSTEP and OpenStep development environment, in particular with *InterfaceBuilder* and *ProjectBuilder* ensure an integrated software development lifecycle for enterprise objects. Together, these tools form one of the most powerful software engineering tools to incorporate, access, edit, and manage custom enterprise applications.

Future Work

The Enterprise Objects Framework will be extended in many dimensions. At this point in time the Framework provides a robust infrastructure geared towards professional software developers. It will be complemented by a set of tools that make it more approachable by end-users as well. Another enhancement will include the integration with object oriented storage engines which will enable object warehousing on the second level of the three-tier distribution model. A third dimension is the strengthening of the top-down development life-cycle which will result in tighter integration with various business process re-engineering tools.

Acknowledgments

The Enterprise Objects Framework product reflects the work of many people. The author wishes to thank all members of the project team, and particularly the members of the design and implementation team for pioneering and developing the concepts and implementing the Framework. The main designers and implementors are Craig Federighi, Linus Upson, Dan Willhite, and Richard Williamson.

References

[Booch91] Booch G., "Object-Oriented Design with Applications", Benjamin/Cummings, 1991.
[Aberdeen 94] AberdeenGroup, "Technology Viewpoint", Volume 7 / Number 12, August 31, 1994.
[Date 95] Date C.J., "An Introduction to Database Systems", Sixth Edition, Addison-Wesley, 1995.
[DD 95] Darween H., Date C.J., "Introducing The Third Manifesto", Database Programming & Design, January 1995.
[Jacobson92] Jacobson I., "Object-Oriented Software Engineering - A Use Case Driven Approach", Addison-Wesley 1992.
[NeXT-EO 94] "Enterprise Objects Framework: Building Reusable Business Objects", White Paper, NeXT Computer, Inc., 1994.
[NeXT-DO 94] "Interoperability Through Distributed Objects", White Paper, NeXT Computer, Inc., 1994.
[NeXT-OpenStep 95] "The NEXTSTEP/OpenStep Object Model", White Paper, NeXT Computer, Inc., 1995.
[RBPEL91] Rumbaugh J., Blaha M., Premerlani W., Eddy F., and Lorensen W., Object-Oriented Modeling and Design, Prentice Hall, 1991.
[SM88] Shlaer S., Mellor S., "Object-Oriented Systems Analysis - Modeling the World in Data", Prentice Hall, 1988.
[Yankee 94] Susan McGarry, "Open Systems: The User Reality", White Paper, the YankeeGroup, 1994.