UNISQL'S NEXT-GENERATION OBJECT-RELATIONAL DATABASE MANAGEMENT SYSTEM

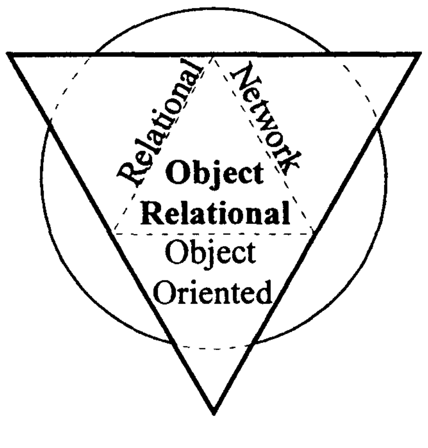
Albert D'Andrea and Phil Janus

UniSQL, Inc.
8911 Capital of Texas Hwy., Suite 2300
Austin, Texas 78759

Object-Relational DBMSs have been receiving a great deal of attention from industry analysts and press as the *next generation* of database management systems. The motivation for a next generation DBMS is driven by the reality of shortened business cycles. This dynamic environment demands fast, cost-effective, time-to-market of new or modified business processes, services, and products. To support this important business need, the next generation DBMS must: 1. leverage the large investments made in existing relational technology, both in data and skill set; 2. Take advantage of the flexibility, productivity, and performance benefits of OO modeling; and 3. Integrate robust DBMS services for production quality systems. The objective of this article is to provide a brief overview of UniSQL's commercial object-relational database management system.

ORDBMS FOUNDATION

During the late 1980s, UniSQL founder Dr. Won Kim observed that it is possible to develop a single database engine which allows a data designer to model data as either object-oriented, relational, or networked, and as such, the models may be unified and coexist within the same database. In fact, the "Uni" in UniSQL represents a *unified* data model. This bold insight is based on a surprisingly simple observation:



If one starts with an Object-Oriented data model, where the object-oriented features are *Discretionary,*

And removes the notions of Inheritance and Encapsulation...

There remains a Network Data Model.

If we then remove the notions of Pointers and Collections...

There remains a Relational Data Model.

UniSQL's definition of an object-relational database system implies support for: N-dimensional OO modeling, 2-dimensional relational modeling Inheritance, encapsulation, object persistence, Class composition, polymorphism, navigational object access, relational (join) access, non-procedural query access, traditional 3GL interfaces, object 3GL interfaces, 4GL language interfaces, language-independent data storage, database services independent of the file system, and on-line DBMS services. UniSQL's commercial object-relational database system products support all of these important capabilities today.

STANDARDS-BASED IMPLEMENTATION

UniSQL has implemented its ORDBMS, called the **UniSQL Server™**, such that it provides *discretionary* object-oriented modeling features for inheritance, encapsulation, pointers, and collections. The UniSQL Server allows object-oriented data modeling with pointer-based navigation, *in combination with* relational modeling using dynamic joins and indexed-key access.

UniSQL Inc. is an associate member of the Object Management Group. In keeping with UniSQL's commitment to standards, the UniSQL Server uses as its foundation an object data model that is OMG Core Object Model compliant -- the most widely accepted definition of object orientation. (In fact, UniSQL founder Dr. Won Kim coined the term "Core Object Model"). As a true database engine, the UniSQL Server enhances and extends the OMG Core Object Model with robust on-line database services independent of the underlying file system. UniSQL is also a member of the ANSI X3H2 (SQL3) committee and an active contributor; for example, UniSQL's set syntax and set-to-table conversions have been adopted within the standard. UniSQL, Inc. is also a full member of the ODMG, Object Data Management Group. This group is comprised of OODBMS vendors. UniSQL embraces and has influenced the architecture specifications of the ODMG, which include: Language bindings for C++ and Smalltalk; OQL -- Object Query Language. The ODMG is working toward merging OQL with SQL3, and on UniSQL's lead, has endorsed SQL3 as an accepted object query language. The ODMG is committed to working as a collective voice to influence the emerging ANSI SQL3 specification; ODL -- Object Definition Language; and OML -- Object Manipulation Language.

The UniSQL Server provides a tightly integrated ad-hoc query language -- a significant departure from the language-centric approach used by earlier OODBMS implementors. It is unrealistic that "empowered" end-users might not need to acquire their data in an ad-hoc non-procedural manner. As such, UniSQL provides the choice to *combine* procedural object-oriented languages, such as C++ or Smalltalk, with a non-procedural ad-hoc SQL/X language -- in a manner very familiar to the traditional relational community.

UniSQL's SQL, called SQL/X, uses a cost-based heuristic SQL optimizer directly integrated into the database engine. SQL/X is an enhanced ANSI SQL2 compliant language which incorporates the most useful object-oriented extensions emerging from ANSI SQL3. There are only four fundamental object extensions to SQL to understand which represent a minimal learning curve for relational developers and DBAs: 1. Class composition: The datatype of a column can be a class; 2. Collections: An attribute (column) may be assigned multiple values (sets / repeating groups); 3. Encapsulation: A class/table can have functions (Methods) associated with it which are the "operators" used to query and manipulate objects; and 4. Inheritance: A class/table may be specialized through inheritance. Taken together with extension #1, class composition, the data model is a Directed Acyclic Graph (DAG).

UNISQL'S DATABASE-CENTRIC APPROACH

The UniSQL ORDBMS Server is a true database management system in the tradition of hierarchical, networked, and relational DBMSs. UniSQL does not rely on the underlying file system for its database services (such as security and backup) as most first-generational OODBMSs do. Rather, UniSQL's database-centric approach integrates all services into the database engine. Traditionally, the services provided by a database management system include a variety of functions which manage and protect data, including: physical placement, schema definition, access, access methods (indexing), manipulation, security, disaster protection, availability, multi-user concurrency control, and monitoring and tuning. The file systems that first generation OODBMSs are based upon were not designed to provide the robust data management services listed above. It therefore becomes necessary for the DBMS vendor to integrate these services into the database engine.

OBJECT-ORIENTED DATA MODEL

UniSQL's underlying object-oriented DBMS provides the ability to model data as true objects. An object is a combination of "data" and/or "program" that represents some real-world entity. Objects can be as simple as a single binary bit, or as highly complex as an aircraft carrier. Reusability is fundamental to object-orientation. Objects, by definition, are reusable. Many will agree that, in particular, the concepts of inheritance and encapsulation differentiate an object-oriented DBMS from a non-OODBMS. Every object is referenced by an Object Identifier -- OID. Objects are defined in terms of other objects. Facts describing an object are represented as attributes. Attributes may refer to other objects, or may be of some atomic scalar base datatype such as a numeric, character string, etc. All objects that have the same attributes and program parts are collectively known as a class. An instance is a record within the class. Each class has a class instance which contains the metadata regarding the class. Shared attributes may be stored once in the class instance and shared amongst all instances of the class.

RELATIONAL MODEL SUPPORT

As an Object-Relational DBMS, UniSQL offers relational capabilities in addition to its aforementioned object-orientation. A sampling of UniSQL's relational model capabilities include: Normal-form modeling, Row/column tables, Indexed-key access, Set-based access, Cartesian cross product, Join, Intersection, Union, Subquery, Derived tables, and Ad hoc query language -- SQL/X

In UniSQL's Object-Relational DBMS, relational tables may be defined in conjunction with object classes (to the extent that the datatype of an object attribute could be a row from a relational table). For example, an order processing system may define product configuration as objects, while the financial data is defined as relationally flat tables.

DATATYPES

UniSQL provides a rich set of system primitive datatypes. These system types include multinational characters, date and time, monetary, and a variety of multi-valued collection types. UniSQL also provides a robust set of multimedia types, called Generalized Large Objects or GLOs (see the "Multimedia" section).

GLOs are used for complex objects such as text, HTML, voice, image, video, HotJava applets, etc. UniSQL also allows users to define their own arbitrary datatypes, a hallmark of object-oriented systems. The mechanism for implementing user-defined datatypes is class creation. Once a class is created, it is available for use as a datatype or domain anywhere within the database. UniSQL-supplied classes, including GLOs, may also be used as datatypes, and may be specialized through inheritance to meet application-specific requirements. Collection operators include membership, casting, equality, subset, superset, union, difference, and intersection.

LANGUAGE-INDEPENDENT DATA STORAGE

A true database management system stores its data in a language-independent manner. However, with the advent of OODBMSs, a key tradeoff was made between language transparency and language-independent access to data. In first-generation OODBMSs, objects stored in the underlying file managers are tightly coupled to the programming language those objects originated from. For example, once an object is made persistent in C++, it is difficult to retrieve it from any other language such as Smalltalk, C, ODBC, or SQL.

UniSQL has implemented a language-independent database storage system. In the UniSQL client/server environment, the UniSQL client Application Programming Interface automatically manifests the data in the format the client language expects. Thus data is stored in a language-neutral format, while access is simultaneously acquired from an object-oriented language such as C++ or Smalltalk, a traditional 3GL language such as C, or a 4GL interface such as ODBC or SQL.

BALANCED CLIENT-SERVER ARCHITECTURE

UniSQL implements a balanced client/server architecture. There is a shared responsibility of tasks between the client and the server. By contrast, some first-generation object-oriented database vendors implement client-heavy products, and others implement server-heavy products. Client-heavy products place most of the processing burden of the system on the client, while the server is relegated to primarily page management services. Server-heavy products are much like traditional relational systems where much of the processing burden is in the server, and very little processing occurs in the client, relegating them to "expensive dumb terminals".

A balanced client/server architecture divides the burden of processing between client and server. Workstation and PC-based client platforms provide increasingly more powerful processing capabilities, yet few systems take advantage of the client processing cycles available. Off-loading some of the server's functions to the client not only allows the server to provide more throughput, but also enables a network of high-powered client machines to become a large parallel processing environment. Some of these client functions include caching to take advantage of locality of reference, query parsing and optimization, schema access, security, trigger and method execution, etc. The balance also provides the centralization of key server-based functions. These functions include logging, backup and recovery, transaction management, lock and concurrency management, index management, etc.

Object-Based Client/Server Protocol

UniSQL's object-based client/server protocol is a self-describing messaging system that passes objects between the client and the server. The client Application Programming Interface (API) passes messages containing objects to a UniSQL Server. The UniSQL Server contains a listener which processes inbound messages and returns outbound messages. The server also manages the location transparency of other servers and back-end drivers. The back-end of the server contains the client API which can talk to either another UniSQL Server, or an open driver kit. As described below, the open driver kit exposes the server's message handling API, allowing developers to custom build servers which communicate via object-based messages. UniSQL is open on the client since any application or third-party can integrate with the client API. UniSQL is open on the server since any legacy function or database can be integrated with the server API. These client and server components can communicate with one another in any combination, providing a highly flexible distributed object messaging architecture.

MULTI-DATABASE INTEGRATION

UniSQL offers a fully distributed multi-database management system, called UniSQL/M. By providing location transparency, UniSQL/M supports distributed transactions and queries across distributed heterogeneous database management systems. The data distribution is transparent to the user or developer. A homogenous view is provided to disparate data -- It appears to the user as a single logical database. This is carried out in the context of transaction boundaries. For example, two-phase commit is supported across heterogeneous databases (where the underlying databases support a two-phase commit protocol).

Furthermore, UniSQL's strong support for distributed joins and updates enables developers to transparently integrate legacy relational data with new object-oriented database applications. This, in effect, "mind-melds" the two data models in one unified model. This provides a smooth migration path from relational technology to object technology. Developers porting relational systems to an object model can leverage UniSQL's strengths to remain backwardly compatible with existing systems, while rearchitecting for the future.

UniSQL /M is a tightly integrated component of the UniSQL Server ORDBMS (historically called UniSQL /X). UniSQL /M rides underneath the data access layer of the UniSQL Server routing queries to their appropriate destinations. Technically, UniSQL /M is an integral component of the UniSQL Server (enabled by a license key). In contrast, other gateway products run as separate processes incurring far more networking overhead than UniSQL /M.

UniSQL/M provides read/write access to all databases. UniSQL's ORDBMS is accessed natively, and drivers are provided for non-UniSQL products: Oracle, Sybase, Informix, Ingres, Rdb, and Versant; and for mainframe access, EDA/SQL and Sybase/MDI. Object classes, virtual classes, and proxies define the unified multi-database. UniSQL/M transparently unifies object and relational models, and may include remote access to non-UniSQL systems.

CLIENT INTEGRATION

The UniSQL Client API is a full-function, open, flexible interface into UniSQL. It allows third-party tools and value-added partners and to easily integrate their products with UniSQL. The typical program contains just a few API calls to manage connections, manage transactions, execute queries, and navigate results. The client API call interface is much like a traditional relational API such as those offered from, say, Oracle or Sybase. The learning curve for C developers is fairly quick and straightforward. The UniSQL client API, in conjunction with the server API, implements a self-describing client/server object-based messaging protocol. API functions include: connection management, transaction management, object passing and receiving, dynamic message interpretation, object, value, set management, cursor/navigation execution, memory management, dynamic schema management, database administration, and error handling.

MULTIMEDIA

UniSQL provides support for large multimedia objects such as image, audio, text, and other "unstructured" objects. Since UniSQL's implementation of these large objects go far beyond traditional BLOB datatypes, UniSQL has called them Generalized Large Objects: GLOs.

GENERALIZED LARGE OBJECTS

The UniSQL Server provides an extensive collection of GLO classes which define a foundation of DBMS services. GLOs may be used for a variety of complex object types including text, HTML, audio, image, video, HotJava applets, etc. On some platforms, many methods are provided to manage and manipulate the large objects, for example, on Sun Microsystems computers there are numerous methods for X11bitmap and SLCaudio GLOs. The developer is encouraged to inherit and extend these classes as needed. The UniSQL GLO classes are fully extensible and reusable unlike other competitive large object multimedia and extended-relational offerings. GLOs may reside in the database or in external files (FBOs). In either case, full transaction control may be applied to GLOs. Access to FBOs requires a file path expression. In cases where a path is not allowed, such as with certain high capacity storage devices, methods are developed which call the device's API. These methods may be incorporated into classes which inherit from UniSQL's GLO class (or any of its subclasses), the benefit being that the developer could still use "SELECT * FROM ALL <glo_subclass_name>" syntax to provide access to the device.

LANGUAGE INTERFACES

UniSQL provides a variety of client language interfaces. These interfaces range from more traditional languages such as C and Embedded SQL/X C, to object-oriented languages such as C++ and Smalltalk.

C

The C API provides a rich, flexible interface into the UniSQL environment. The C API is the foundation for all other language interfaces, including SQL/X. Architecturally, this provides a single point of entry, and therefore, a single point of maintenance, into the UniSQL Server.

SQL/X

UniSQL's cost-based heuristic SQL optimizer, called SQL/X, is tightly integrated into the database engine. SQL/X is an enhanced ANSI SQL2 language which incorporates the most useful object-oriented features emerging from the ANSI SQL3 specification. UniSQL is a contributing member of the ANSI X3H2 SQL3 committee, and also a member of the ODMG which has endorsed SQL3 as an acceptable Object Query Language. UniSQL allows the intermingling of procedural languages (C, C++, or Smalltalk) with non-procedural ad-hoc SQL/X -- in a manner very familiar to traditional relational developers.

ODBC

As of this writing, the UniSQL PC Client runs on Windows 3.1 and Windows NT, with Windows 95 and Macintosh in development. The PC Client has been implemented by porting the UniSQL C API to the PC platform. The UniSQL ODBC interface rides above the PC Client C API. ODBC (Open Database Connectivity) is used in PC-based development tools, report writers, and ad-hoc query tools. ODBC is a strategic part of UniSQL's commitment to openness and PC connectivity.

C++

The UniSQL C++ interface provides ODMG-compliant transparent access to the UniSQL Server. The UniSQL C++ interface was the first to offer an ODMG 1993 compliant C++ binding. UniSQL is a member of the ODMG and embraces the ODMG architecture specifications. UniSQL is committed to supporting the ODMG bindings for C++ and Smalltalk as they continue to become available.

SMALLTALK

Smalltalk is a dynamic object environment unlike statically compiled C++. Any object may be dynamically generated and made persistent at any time. As with the UniSQL C++ interface, the UniSQL Smalltalk interface provides ODMG-compliant transparent access to the UniSQL Server.