Objectivity Case History

Customer Information

Customer: Newport News Shipbuilding (NNS)

Industry: Ship design and manufacturing

Application Domain: CAD/CAM

Status: Deployed

Platform: Silicon Graphics IRIX (UNIX)

Compiler: C++

Other Tools: Application supports IGES transfers of data to drawing programs for the production of drawings.

Company Background

Newport News Shipbuilding operates the most advanced shipbuilding facilities in the world. Their range of operations enables us to service a wide variety of construction, conversion, repair and overhaul projects simultaneously, all with key advantages in the areas of reliability, service, quality, productivity and competitiveness.

With their diverse capabilities and unique facilities, they are a recognized leader in virtually every facet of shipbuilding and the technology it takes to do the job right, no matter how large or complex the ship.

Since 1900, NNS built 264 Navy Ships, 469 Commercial Ships, and 73 Other Vessels.

Newport News Shipbuilding is America's premier shipbuilding company with annual revenues of approximately $1.8 billion and over 18,000 employees. The company is the leader in the design and construction of nuclear-powered aircraft carriers and submarines for the U.S. Navy and produces a variety of ships for domestic and international customers.

VIVID® Solid Modeling System

VIVID® Solid Modeling System, UNIX workstation version is a 3D Solid Modeling tool, based on Objectivity/DB, is a smart product model to capture object behavior for the first time. VIVID® generates all the geometry for design, manufacturing, and analysis to build ships more efficiently. Previously they used wooden mock-ups to prototype design specs and manufacturing spec.

VIVID® was originally a DARPA funded project. Newport News has studied the practices worldwide of ship building techniques to incorporate them into VIVID®. Reason they are doing this is to remain competitive worldwide against foreign ship builders who are subsidized by their governments.

VIVID® key thrust is cost avoidance, get design right the first time so we don't have to make costly changes on the manufacturing floor. VIVID® has been involved in designing a super tanker and control tower for an aircraft carrier. Newport News has strategic future plans for enhancing VIVID®.

VIVID® users are skilled ship designers, experienced in the use of several different CAD tools, estimated to have an average of 8-12 years of ship design experience in their specific disciplines.

VIVID® market differentiation is that it develops a 3D-product model of the ship with sufficient information to assist in producing manufacturing documents for cutting structural steel. It is not a 2D drawing system.

VIVID® is written in C++ on Silicon Graphics IRIX (UNIX) with Objectivity ODBMS.

Why Objectivity?

NNS selected Objectivity for its support of true object-oriented development and its ability to manage large, complex databases with high performance characteristics. Strategic advantages for using Objectivity:

1. Good fit for OO application.
2. Data Objects model the elements of the ship's structure concept design and piece part design derived from the concept design.
3. Code was developed using OO analysis and design to develop the user interface and data model.
4. Set and Get methods are used for managing some attributes. Most of the data model is built on a higher abstraction than simply setting and returning data values.
5. Complexity was very much based upon class interconnections (associations). The design is product model based, and was developed with numerous hierarchical associations between the objects used to represent the design of a ship.
6. Multiple Inheritance was used extensively.
7. Virtual functions were used often.
8. Cache configurable – Data was organized by ship system and subsystem to take advantage of Objectivity caching and paging architecture, and to enhance the locality of reference once data had been retrieved.

9. TCP/IP – Only network modification required was in default configuration of file server to permit Objectivity clients to dynamically access the database files.
10. Most Objectivity basic features, plus dictionary lookup, indexing, variable length arrays of pointers and character strings were used.
11. Scalable databases (64 bit OID) – This allowed partitioning of the product model to support the concurrent design of several different ship classes in different databases under the same federation. Due to file size limitations on the host platform, it has been necessary to devote multiple databases to a single ship project.
12. Indexing is confined to indexing within a single container.
13. OoVArray(ooRef) was used in the ODMG styled association model.
14. Varray types used primarily on ooRef for associations. Some use was made for double precision floating point data, extensive use was made for variable length character strings.
15. Heterogeneous architecture – Primarily a client architecture without servers. Each user has a copy of complete application code and updates portions of the database containing the portion of the ship model they are working on. One or two server processes take care of some clean-up operations that lock contention prevents the user applications from accomplishing.
16. Page serving architecture performance – Attempt was to group data objects into containers that would be needed together. The performance has been reasonably good for this type of access.
17. Multiple Readers One Writer (MROW) – All application copies operate in MROW. This is the only mode in which multiple users are able to operate on the shared data.

Object Model Design Specifications

1. In general, transient copies of persistent capable classes are not generated.
2. To a limited degree, MACROS were used to accomplish the container and object creations and clean-ups if they failed.
3. Objectivity ODMG facilities were not available when the majority of the code was being developed.
4. ODMG biases were for portability and database independence in the event the application had to be ported to an alternative database product.

Objectivity Application Specifics

Principle Class Name	VivProduct
Data Complexity	1100+ classes
Database Size	Gigabytes of data.
Containers	30,000 per database.
Concurrent Users	20+ clients.
Domain object	3D product model of ship
Attribute value	relates to the domain
Relationship	OoVArray(ooRef)
Platform	Silicon Graphics IRIX (UNIX)
Language	C++
Components	The “component” model with interfaces as implemented in Microsoft COM was not used. The breakdown of objects was primarily functional to model the ship's conceptual organization of parts for design and manufacturing.
Evolved classes	Approximately 130 classes
Evolution Strategy	The data for the evolved classes were converted from the old representation into the new representation by one or more utility conversion programs run during the “down time” in which the new release of the application was being turned over from development testing to production.
Object versioning	Not employed

Object Structure or Physical Hierarchy

Common transaction involves the creation of a new part and the geometry data associated with that part (estimated at 8-12 persistent objects, and as many associations between the objects), or the modification of the geometry of parts already in the database (estimate updating of 4-8 persistent objects, and as many associations or reference tables).

Transactions generally span several minutes, but a transaction may be left open for an hour, if the user is interrupted and walks away from his workstation. No effort was made to cause a transaction to expire if uncompleted in a specified time frame.

Because many users share much data (even if only in read-only mode), MROW was the only operational mode that appeared to be viable.

Open Objectivity Containers as read-only, then elevating the lock to update when necessary, along with a retry strategy on failure, should handle most of your deadlock conditions.

All databases in the projects that the user will access are opened with the intent to update. There are essentially no "read-only" clients, except for maybe an occasional ad-hoc monitoring tool that you may want to develop for database administration use. Actual containers are opened in read-only mode. When an update needs to be made, the lock is

elevated to update mode. If the elevate of the lock fails, it is retried up to a select number of times over a span of say 10 seconds until it succeeds, otherwise the transaction is aborted, and the user is notified.

Grouping data objects into containers that would be needed together increased performance. The data clustering philosophy attempts to minimize transaction conflicts between users. Most transactions are estimated to impact no more than 4 or 5 containers in the Federated Database. Some transactions may impact as many as 10 containers.

The number of containers is allowed to grow to about 30,000 in a database, or its disk file size approaches 1.3GB, then a new Database is made available to the project for additional data.

Data was segregated by purpose. Special containers have been set up for part number look-up and name indexing, otherwise a separate container is used for each "system" which tends to group together the product model data for a group of structural parts in a volume up to about 150 cubic meters.

Lock-checking tools were used to identify when someone has locked a container for a long period of time.

Only one federation is used in the production environment. As many as four federations are used in the development environment, to support development, testing, debugging, and progressive schema evolution between the various development stages.

A hard-coded "global" name is used to access a "directory database" within the federation. This "directory database" has several "dictionary objects" providing pointers to the initial database file for each ship project.

Each ship project's initial database has a name index cross referencing character string names to the database objects they reference, as well as a special "project header class" that has lists of application assigned product numbers cross referenced to the database OID's. There are also pointers to the top levels of the "system" and "assembly" organization trees that hierarchically reference all products within the ship project.

Physical System Architecture

Transactions run on the client (no servers are used except for file storage).

Objectivity's distributed nature is used in the development environment where the number and size of database federations exceeds the available storage on any one machine. Currently, the file servers have sufficient capacity to hold an entire federation.

Distribution on Local Area Networks (LAN).

User activities are homogeneous. One application used by all users. Database demands do not vary much over the types of operations performed.

User Interface

GUI based transactions are long lived.

Lookups are optimized for the end user and lookups that need to be fast. Random-type lookups (name based) have been optimized through the use of an index. Search by application assigned product number uses the project's cross-index to locate the OID's for the desired object. This cross-index uses multiple fixed arrays of OID's to accomplish this look-up.

The Objectivity database is invisible to the end users (except when deadlock conditions actually occur).

The Database federation has grown over time. Additional functionality has been added to the application.

Backups

Daily backup of the data is performed by manual file copy. Backup is manually driven.

Reports

No user reports are generated from the data content of the database. Objectivity utilities are used to gather various statistics that are analyzed by the database administration.

Competition

Project was started under a sub-contract with another software development project led by another company. The lead contractor made the choice of database. No evaluation was done at NNS. ODMG biases were for portability and database independence in the event the application had to be ported to an alternative database product.

Issues and Resolutions

1. Limited use of Objectivity associations was made. Objectivity did not support the ordering required by the application. They utilized an association model developed from the early-published versions of the ODMG standard before Objectivity supported ODMG functionality.
2. One or two server processes take care of some clean-up operations that lock contention prevents the user applications from accomplishing.
3. Schema Evolution has been painful. Has necessitated special conversion programs to read old records, and reconstruct the replacement records, then to rebuild the

associations that exist between the modified and unmodified objects. Has not been done on Objectivity release 5.0 or later.

4. Concurrency issues arose during development from having multiple users be able to access and update objects within the same system (i.e., within the same cluster/locking unit). This issue was resolved by using a lock-checking tool to identify when someone has locked a container for a long period of time and using a technique of opening containers read-only, then elevating the lock to update when found to be necessary, along with a retry strategy on failure has handled most of the "deadlock" conditions.
5. Schema evolution has not been a problem for new and additional classes. Schema evolution where classes have been modified has not been able to make use of Objectivity evolution tools. Last such evolution was accomplished on Objectivity releases prior to 5.0.

Contact Information

Objectivity Account Manager: Al Fox
Objectivity System Engineer: Al Mannarino

Customer Contact:
Business: Patrick Kennedy
Technical: Andy Lehman