Objectivity Case History

Customer: Cornell University Wilson Synchrotron Lab

Domain: High Energy Physics
Application Domain: Physics Research
Status: Under Development
Platform: Offline: Solaris, DEC OSF1. Online: DB on Solaris, other parts on Solaris, Windows NT, VxWorks for PPC and M68k.
Compiler: Offline: C++ (the main data access framework), Java (1 stand-alone application). Online: C++ 80%, Java 18%, C 2%

CLEO III Offline/Online Software Application Review

General Questions

(1) What does the application do?
We have several different applications:

i.) Offline
  a.) Event database
  b.) Constants database

ii.) Online
  a.) Histograms
  b.) Run Statistics
  c.) Run assignment and certification
  d.) Alarms
  e.) Program/library binaries

(2) What part of the industry is it in?
Basic Research: High Energy Physics

(3) What is it called (marketing name and internal name/anachronism if appropriate)?
No commercial application name.
(CLEO III Offline/Online software)

(4) Who is the target market (if deployed application-could be internally used)?
Only internal usage.

(5) Who is actually using it?
Offline: Physicists reconstructing and analyzing HEP physics data
Online: Physicists who operate the experiment.

(6) Profile of user?
Offline: Everybody is in principle a developer, because everybody may contribute new types of data to be stored in the database. So everybody has to be able to compile and link. This may cause problems (damage to schema etc.) so we will have to restrict who is allowed to contribute new types.
Online: 'Shifter', i.e. a Physicist with a small amount of training to operate the experiment in shifts (7X24). He/she knows little about the system.

(7) How are they using it?
1.) Offline: a user with C++ program. We use an "application framework" (the "Data Access Framework"). This framework forces upon the user the way how data is accessed. The user doesn't actually know if a certain data item is accessed from the database or from some other "source", because we have an isolating interface between data access and database, and the run-time configuration via dynamic

loading determines if the data comes from the database or from somewhere else. The user code doesn't have to change!

2.) Online: user operates it through simple GUIs and needs precise guidelines for normal operation as well as troubleshooting.

(8) What is its market differentiation?
N/A
(9) What language(s) is it written in?
Offline: C++ (the main data access framework), Java (1 stand-alone application)
Online: C++ 80%, Java 18%, C 2%

(10) What platform is it on?
Offline: Solaris, DEC OSF1 (if we ever get it ☺)
Online: DB on Solaris, other parts on Solaris, Windows NT, VxWorks for PPC and M68k.

Object Oriented Application General Questions

(1) Was your application a good fit for an "OO" application?
Offline: Yes. Lots of data/objects, lots of relationships between objects etc. Much more natural than Fortran, the historical language of choice.
Online: Yes. The data structure is complicated, and the base concept is exchange of 'objects' through a network.

(2) Does it manage a complex problem?
Offline: 200TB of "event" data. The relationships are not that complicated, but complicated enough.
Online: 6FDs, ~100 nodes on 4 different OS run ~300 processes, DB requires concurrency, read/write performance and data integrity, network (CORBA) 100BaseT challenged.

(3) Did you model real-world abstractions (example: a network element management system with objects representing different parts of a network node)?
Yes.

(4) Did you use object-oriented concepts or is their code "dressed up" procedural C code (such as large case statements based on type)?
Yes, OO.

(5) Do you use setters and getters?
Yes, when applicable.

(6) Is the complexity based upon class interconnections (associations)?
Offline: The main complexity doesn't come from the interconnections. The association complexity is not great, but the vast amounts of data!
Online: Complexity of DB schema is medium and uses associations.

(7) Was Multiple Inheritance used?
Rarely. We try to restrict ourselves to programming the safer "Java" way, where the second inherited class is abstract.

(8) Were virtual functions used?
Yes.

(9) Cache configurability?
Offline: If you mean that we have a way of caching data that are frequently accessed, yes.
Online: We currently do not configure caches actively, but it might be a tuning option for later.

(10) TCP/IP communication - no network modification?

Online: We use standard IP throughout (CORBA). Detector system runs in a separate isolated subnet.

Objectivity Unique Questions

(1) What features of Objectivity are you using?
Federated Database/db/containers…
Reading/writing/creating/deleting objects from/to database.
Associations
Iterators.
STL.
Naming Objects
Variable-size Arrays
String class
Indexes/scans/PQL
Informational functions
CommitAndHold, downgrade locks, MROW
OoContext without threading.
Online: Threading (ooContext)

(2) How do these features compliment the application?
Online: Threading, MROW necessary for multithreaded CORBA applications.

(3) How did Objectivity differentiate in those areas?

(4) Why did other vendors not meet your needs?
Scalability.

(5) Container inheritance - configurability, fit of product to model?
Yes, we like it!

(6) Scalable databases - 64 bit OID (raw quantity of data)?
Yes, very important.

(7) Single logical view - much less programmer overhead?
Yes.

(8) Distributed query/index features - seamless linking of indexes and query capabilities, built-in distributed query (FD wide) functionality?
Yes.

(9) Objectivity associations?
Yes.

(10) ooVArray(ooRef) or home-brew segmented?
ooVArray(ooRef).

(11) VArray types

(12) heterogeneous architecture (clients and servers)
Online: System processes are thick clients, data access processes are CORBA servers and thick clients, all other components are CORBA servers/clients in dynamic configuration.

(13) heterogeneous language compatibility
Since C++ and the other supported languages (Java, Smalltalk) handle schema stuff differently, this definitely could be improved.

(14) page serving architecture performance characteristics
Online: Seems suitable, has not been compared to other models.

(15) container level locking - less TCP/IP traffic
Have run into problems; but we will be able to get around container-level locking.

(16) multi-threading in Java - automatic serialization of persistent operations for all threads
Not used (yet).

(17) Joined to Session

(18) multithreading in C++ - no need to manage shared memory, threads can be viewed as processes in terms of persistent operations
Offline: Not used.
Online: Multithreading essential system requirement. Thread communication/synchronization through object pointers.

(19) composite object support - propagation semantics on associations
Very useful.

(20) explicit clustering - built-in pre-fetch
Very useful.

(21) predicate query language - flexible and powerful, any level of parenthesis nesting
Offline: Will use soon, not used yet.
Online: Used for all data retrieval.

(22) automatic index integration with predicate query language (index on multiple fields of class)
Offline: Not yet used.
Online: Yes.

(23) MROW - how is it being used?
Offline: We weren’t planning to use MROW, but with the lock problems when populating the disk from multiple processes, we will try it to see if it helps.
Online: Heavily used to optimize performance.

(24) object versioning?
Not used.

(25) schema evolution (class versioning)
Expecting 200TB of data, we have decided to stay away from schema evolution and use the poor man’s approach: give new versions a new name (e.g. MyClass1, MyClass2 etc.).

Object Model Design Specification.

(1) How are you using Objectivity, especially the relationship between transient and persistent objects within their application.
We have an intermediate insulation layer. Which means we translate between transient and persistent objects.
Online: Usually transient object = CORBA object.

(2) Did you abstract the database API?
Yes. The user doesn't even know that he is accessing data in the database even.

(3) Are you using any of Objectivity ODMG features?

Not anymore after running into problems with d_Ref's pinning objects in memory.

(4) Was safety a concern of theirs in terms of ODMG compliance, i.e. persistence by inheritance (handle vs. pointer support)?
N/A

Object Structure or Physical Hierarchy - both transient and persistent.

(1) What are the transactions characteristics:
a. long lived vs. short lived?
Offline:
Long, as long as we can make them. For populating the [Federated Database] we have multiple processes writing different containers in the system. One has to be very careful about locking problems. But currently we do a "commitAndHold" every 100 "events" or so.

b. MROW vs no MROW?
Will try it to get around some of the locking problems.

c. read vs. update FD open?
We use both in the same process. The user can run-time configure what will happen (read, or just write, or read-and-write). So if the [Federated Database] was opened in read mode, and then the user decides to also write, the [Federated Database] will be upgraded to update mode.

Online:
Transactions must be short, since the data must be available for other processes within seconds. Because of the overhead to open/close/commit incoming data is buffered and committed in bulk in a single transaction. MROW is used to improve concurrency (e.g. Alarm Manager can write alarms while GUIs and other clients can read access DB. FD open is read or update mode depending on application.

(6) How do these transactions map to containers?
Offline: Each transaction writes to its own container(s). The only lock problems come about, when we try to attach a new container to a central object which keeps track (in an ordered fashion) of all containers in a db.
Online: Container design is not sophisticated.

(7) Are container level locking issues based on locality of objects involved in the particular unit of work?
Yes, please see above. It's a problem, but we have ways around the lock problem.

(8) Number of databases in the federation?
Offline: The max number of databases in an [Federated Database] will determine how often we start a new [Federated Database] . We'll use [Federated Database] per new "dataset". Basically when we run out of databases, we will start a new [Federated Database] .
Online: Usually only one or two. The only natural division between objects that matches somewhat retrieval patterns is time of object creation.

(9) Are you using more than one federation?
Offline: Yes, many. In fact this is a really limiting issue now. We want two things:
i.) more than one [Federated Database] with a different schema at the same time.
ii.) More than one [Federated Database] with the same schema opened and closed sequentially (to loop over the "datasets")
Online: One federation per DB system. No multiple FD access needed.

(10) Number of containers in common databases by database purpose?

Offline: We use a different container per type. We have many hundreds of types. And we group the same type in a different container per “run”. So again, once we run out of containers in a db, we’ll start a new db, and once we run out of dbs, we’ll start a new [Federated Database] .
Online: Number of DBs/containers only increases when existing containers are full.

(11) Number of objects per container by container purpose?
Offline: As many events as there are in a run. Usually on the order of O(100k).
Online: See previous.

(12) How are you using clustering?
Offline: By container. The natural grouping is runs that contain events. So it makes sense to cluster events in a run together in a “run”-container.
Online: Clustering of event type data is natural (by creation time), of other data hierarchical (e.g. by originating component).

(13) Do you segregate your data by purpose (i.e. statistical information container, network topology database)?
Yes, please see above.

(14) Do you mix and match your data for performance (instead of segregating)?
Offline: Data that belongs to a category is clustered together (written into the same DB). This improves performance for analysis access.
Online: Retrieval patterns are strongly random. There are exceptions, like access by run number. Here the data is already clustered naturally (proximity in time).

(15) How does the application initially get into the object name space (maps, indexes, scope names, or system names) depending on transaction type or is there a common entry point for all transactions (Clustering, MROW, single-logical view and flexible naming mechanisms)?
Offline: Both scope names and indexes. Once you get a hold of the “ordering object”, you can iterate over containers and get what you want. The other way is to go directly to a certain index. We use both approaches for different purposes.
Online: Indices.

Physical System Architecture.

(1) Where do the transactions run (client vs. server or both)?
Offline: Objy is a client-heavy process.
Online: System processes run transactions directly (if performance not critical) or in a separate thread on buffered data. Data retrieval usually goes through a multithreaded CORBA server.

(2) How did Objectivity compliment each of these scenarios?
Online: Threading essential, MROW helps to improve performance.

(3) Do you take advantage of Objectivity's distributed nature?
Offline: Yes. Different databases will live on different disks hanging off different machines, and some of them will live on an HSM system.
Online: Administrative issue which will certainly be addressed later. Gives us a handle to tune performance by changing system configuration.

(4) If so, exactly how distributed (local vs. WAN)?
Local for now.

(5) Are you using Objectivity's platform heterogeneity support?
Offline: Yes, we're hoping to access the same database from different platforms (Solaris, DEC OSF1, and Linux).
Online: No. NT is a fallback solution.

(6) Number of clients (simultaneous users)?
Offline: At any one time probably on the order of, say 50.
Online: Usually 1-5. Program DB: ~50 threads within single DB server. Possible to distribute to several servers.

(7) Are these users "active" users simultaneously?
Yes.

(8) How many transactions are running simultaneously currently and envisioned in the future (Objectivity client side cache complements these architectures as you may have as many transactions as you can throw threads/processes at Objectivity)?
Offline: Each process will have its own transaction. So say on the order of 50 or so.
Online: See previous: one transaction per thread.

(9) Have you performed concurrency analysis?
Offline: Not yet.
Online: Yes. Program download performance simultaneously by 6 clients is sufficient but not terrific.
The study was not run in an optimized environment.

(10) Are you using MROW and various flavors of lock waiting modes?
Offline: We're about to get around some locking problems. See above.
Online: Yes.

(11) Are user activities homogeneous or are there different flavors of users with different database demand profiles?
Offline: In the short term we'll have two scenarios:
- many processes populating the fd[Federated Database] b
- from then on the [Federated Database] is read-only, and many processes/users will read from an [Federated Database]

Online: We only distinguish between writers and readers. There are different profiles for the different FDs.
(12) If users are homogeneous, can they select operations that require different database demands?

Offline: Because of the interface layer, the users don't directly access the database, they aren't really aware of it. In the future we want to use data-mining techniques.
Online: Some retrieval techniques (in search scope) are preferred because of performance.

(13) If so, how does this affect concurrency and database locking statistics?
Offline: Don't know yet.
Online: No.

(14) What concurrency issues arose during:

a. Development?
See above.
b.Deployment?
Offline: Haven't deployed yet.
Online: Requirements on concurrency. Needs further optimization and will affect system configuration.

(15) How did you resolve these issues as well as deadlock issues (perhaps you keep user statistical information in the database as well as error logs)?
Offline: We hope to shake down any deadlock issues well before any users get to access the database. We will start using oofs and keep logs that way.
Online: Open/commit/threading strategy. The system deployment has not yet reached the real-life stage. No deadlocks encountered so far.

User interface

(1) What are your GUI based transaction semantics:

a. Long lived (active view) vs. short lived (refresh - dialog box dismissal)?
Online: Active view information goes through transient object transportation (CORBA). This is the standard GUI situation. DB data is usually refreshed.

(2) Determine how lookups are optimized for the end user and lookups that need to be fast?
Online: DB application: Only index use. Hardware: CORBA servers operate on local disks.

(3) Does the end user know about any database underneath the application (Is Objectivity embedded and invisible)?
Online: No – all CORBA interfaced.

(4) How has the application (product) evolved over time (such as more users, larger database)?
N/A

(5) How well did Objectivity scale?
Online: N/A.

(6) Did you use Objectivity schema evolution features?
Not yet.

Backups

(1) How regularly do you backup your data?
Offline: since the raw data are saved on tape in our own format, and then populate the [Federated Database] , we may not back the databases.
Online: ~once per day planned.

(2) Is it automatic or user driven?

Online: Automatic planned.

(3) Do you take advantage of Objectivity's 24 X 7 support?
No.

Reports

(1) Do you generate reports from the database?
N/A

(2) What tools do you use for this purpose?
N/A

Other Technologies Integration

(1) Are other technologies integrated into your application (such technologies include MFC, ORBs, text search engine, web servers, email servers, etc)?
Offline: possibly use of an ORB to allow accessing a 2nd [Federated Database] in a process.
Online: CORBA (VisiBroker)

Objectivity Issues

(1) What issues were encountered with Objectivity and how were they resolved?
Offline: lock problems. Still working on the best solution.
Online: Large overhead to open/commit. Resolution: Make compromises and adapt a [transaction] commit strategy.