Workshop Report on Experiences Using Object Data Management in the Real-World

Akmal B. Chaudhri
Interoperable Systems Research Centre
Computer Science Department
The City University
Northampton Square, London EC1V 0HB, UK

akmal@soi.city.ac.uk

Abstract

The OOPSLA '97 Workshop on *Experiences Using Object Data Management in the Real-World* was held at the Cobb Galleria Centre in Atlanta, Georgia on Monday 6 October 1997. This report summarises some of the commercial case-study presentations made by workshop participants.

Introduction

This workshop was organised in an attempt to bring together academics, users and vendors from the object database community, following the previous successful workshop on *Object Database Behavior, Benchmarks, and Performance* at OOPSLA '95 [Zorn95]. The workshop Call for Papers (CFP) attracted a mixture of submissions from academics and users and 16 position papers were selected for presentation. There were several main themes to the submitted papers:

- Object Database Applications and Design.
- Benchmarks and Standards.
- Architectures and Frameworks.
- Object to Relational Mapping.

The position papers were organised into short and long presentations, with an opportunity for workshop participants to pose questions and raise issues for discussion after each presentation.

The remainder of this paper is organised as follows. The next section describes some of the commercial case-study presentations in the order that they were presented, (due to space constraints, however, other workshop presentations cannot be summarised in this paper). This is followed by the major conclusions.

In the Trenches with ObjectStore

This paper was written by David Hansen, Daniel Adams and Deborah Gracio and presented at the workshop by David Hansen. All three authors are with the Information Systems Department, Pacific Northwest National Laboratory (PNNL).

David's presentation focused on experiences that his group has had using ObjectStore for a Scientific Data Management (SDM) system.

Fast performance is provided by ObjectStore through its Virtual Memory Mapping Architecture (VMMA), but at the expense of pointer safety and increased burdens on the programmer. David concluded that safe pointers, at the expense of a little overhead, would have been better to avoid some of the design compromises that needed to be made for SDM.

The second issue discussed by David was that of physical database organisation. In ObjectStore, objects that are likely to be referenced together can be placed in clusters (fixed-size containers) which reside in segments (expandable storage containers). Cluster size can only be set at creation-time. Similarly, segments need to be carefully sized. David argued that such physical considerations should be the responsibility of a Database Administrator (DBA), rather than hard-coding them directly into each application.

The third issue raised by David was that of vendor marketing. He (very diplomatically) suggested that the marketing department at Object Design, Inc. (ODI) was ahead of product capabilities. For example, SDM uses the Silicon Graphics platform, running the IRIX operating system. However, ODI does not appear to give high priority to this platform in terms of support, development or maintenance.

The fourth issue relates to tools support. David described some experiences with ODIs Internet Solution Suite (ISS). He argued that several tools of interest for SDM, such as ObjectForms and the Extended Object Manager were not up to quality. For example, the SDM team wanted to view objects from their database, but found ObjectForms to be very limited in capability and only providing relational-like data access. Similarly, the wrapper class for the Verity Developer Kit (VPI) was somewhat limited in capabilities. Again, the application programmer was left to deal with performance and safety problems.

David concluded that experiences with ObjectStore had been very mixed, but on the whole, the product was a superior solution to using relational or object-relational products for SDM.

CORBA-Based Applications, Services, and Object Data Management

This paper was written and presented at the workshop by John Chen. The author is with Silicon Graphics, Inc.

John described the results of a project to build a set of CORBA applications and some object services needed by those applications. One of these services being persistence. A pure object database was used as the first persistence mechanism and John's experiences reported below are based on this.

One of the first issues discussed by John was how to locate a database object for a particular CORBA object:

1. Use the OID generated by the object database.
2. Use the names selected by the CORBA object.
3. Use the names managed by the Persistence Service.
4. Use the OID generated by the object database and add a tag generated by the Persistence Service.

The first three options are unsatisfactory, since the particular object database system reuses OIDs (option 1), which makes it difficult to keep track of deleted objects or a mapping table is needed for names → OIDs (options 2 and 3), which introduces an overhead. The fourth option does not incur a mapping overhead and was the one selected.

In a distributed environment, it is important to access local data as much as possible to avoid unnecessary network traffic. The CORBA applications do this by using the *Federated Database* mechanism of the object database. This enables those objects that are owned by a particular CORBA server to be physically held on the same machine as that server. Furthermore, good concurrency is achieved by using the *Container* mechanism of the object database - all objects of the same type as their CORBA objects are held in the same container. Also, database interaction time was found to be the limiting factor for I/O-bound applications, so tradeoffs were necessary to achieve good throughput, such as batching requests and committing transactions per request batch.

John then discussed several issues related to database management and application packaging. These included shutting down the database server in a distributed environment - not an easy task, since all applications need to shut-down first and this can only be done when any outstanding transactions either commit or abort. A related issue is that of the lock server used by the particular object database which also requires careful handling. The object database also provides very minimal security and access control, so user authentication and authorisation are performed at the CORBA object level. Schema management was also a little difficult in this object database system, since all application schemas are held in one database.

Overall, despite the problems just described integrating CORBA and the particular object database, John ended on a positive note and felt that the inconveniences and problems were not insurmountable.

Using Objectivity on the IRIDIUM[1] System

This paper was written and presented at the workshop by Jeff Garland and Dick Anthony. Both authors are with Motorola, Inc.

The IRIDIUM project involves a large number of components, including satellites in low earth orbit, ground stations, etc. OO software and development techniques are currently being used to develop the System Control Segment (SCS) which is central to the entire IRIDIUM system.

Objectivity/DB has been successfully used for sometime in several SCS ground sub-systems. Other subsystems also use OO tools and techniques, such as Orbix, Rogue Wave and ILOG. An obvious issue is that of combining these diverse tools to work successfully and this was achieved in IRIDIUM by the SCS team developing high-level design and integration techniques, including the use of design patterns. For example, one use of Orbix within SCS is to perform distributed event notification to database clients. This is modelled by a design pattern, called `Observer`. When a database client receives an alert, it requests the database object using Objectivity/DB's direct interface.

One of the problems with any large-scale development is trying to ensure good software practices. Within SCS, this was addressed by a series of measures, which can be summarised as follows. First, coding guidelines were used with Objectivity/DB. In particular, calling `ooUpdate` for all non-const

[1] IRIDIUM is a registered trademark and service mark of Iridium LLC.

methods and ensuring that there were no direct interfaces across major sub-systems. Second, by a framework of objects to unify Objectivity/DB's features. These included enhancements to session classes, using wrappers around transactions, the development of specialised collection classes (since the collection facilities provided by Objectivity/DB were inadequate for this project) and developing an error handler to convert Objectivity/DB errors into exceptions. Third, standard `make` rules were used to ensure that all system-level schemas were correctly built. Fourth, C++ coding guidelines were used and enforced through code inspections. Fifth, all system-level interfaces were carefully developed with Programmer Interface Guide.

Jeff and Dick summarised their presentation by noting that Objectivity/DB was a key element of the ground system software and was being successfully integrated and used with other technologies through careful attention to interfaces, implementation issues and using only the core features of the product. The SCS team had also developed a framework for the rapid development of applications with the object database, which reduced the development time for new applications. Finally, an important factor to the success of the overall project was the large number of experienced developers being used.

The Electronic Library Project: SGML Document Management System Based on ODBMS

This paper was written by Philippe Futtersack, Christophe Espert and Didier Bolf and presented at the workshop by Philippe Futtersack. The first two authors are with Direction Etudes et Recherches, Electricité De France (EDF) and the third author is with Ingenia SA.

At the R&D division of EDF, large quantities of documents are generated. These include such things as technical reports, project proposals, organisation charts, etc. These contain large quantities of text, image and possibly video data and currently number 140,000 documents. There are also hyperlinks between documents, to represent relationships, which are stored in separate HyperIndex documents. At present there are 80,000 SGML documents and 60,000 HyperIndex documents. Since the actual documents represent a considerable corporate asset for EDF, one aim of the project described by Philippe was to make this information available in a more accessible and consistent manner.

The project team decided to use SGML, since there is already an ISO standard and due to its widespread use in many other industries. To enable any links to be defined between any documents an extension, called HyTime, was also used. Documents stored in the object database are represented as a tree structure with an average of 1,000 nodes and leaves. Each node or leaf is an instance of the ElementSGML class with the HyTime links being generated by a HyTime engine developed by the team. The actual database schema comprises more than 70 C++ classes and can be used to store any SGML, XML or HTML document. O_2 was used to provide persistence and was chosen because of previous experience some members of the team had using this product.

Since the full-text query facilities provided by O_2 were inadequate for this project, the development team decided to use the Search '97 full-text engine from Verity. The indexing was performed on the textual content and structure of the 80,000 SGML documents by scanning each document tree in the object database. A web interface provides the mechanism for end-users to query the database with the results being displayed in a table showing the documents that satisfy the selection predicate. If the user selects a particular document for display, this requires a recursive tree traversal of 1,000 objects on average. For a document search or displaying a document, the response time was found to be less than 1 second.

Philippe concluded that the integration of three different technologies, namely an object database, a full-text retrieval engine and a web interface, was far better than expected and provided superb performance. The object database also demonstrated that it was very efficient and well-suited for the persistence mechanism for a structured document management system.

O_2 and the ODMG Standard: Do They Match?

This paper was written and presented at the workshop by Suad Alagic. The author is with the Department of Computer Science, Wichita State University.

Undoubtedly, one of the major reasons for the slow uptake of object databases for commercial applications has been the lack of appropriate standards. The standards work that was begun a few years ago by the Object Database Management Group (ODMG) was forced to work on standards issues after many object database vendors had already begun offering commercial products. Since object databases provide seamless bindings for object-oriented programming languages, eliminating the so-called "impedance mismatch" problem, the task of standards development has been further complicated as each object-oriented language has its own data model and type system. Suad's presentation, therefore, focused on three issues: (i) the type system, (ii) the model of persistence and (iii) non-traditional object-oriented features.

For the type system, the ODMG Object Model attempts to provide a common model for several object-oriented languages: Smalltalk, C++ and Java™. However, as Suad noted, Smalltalk is dynamically typed whilst C++ and Java are mostly statically typed. Furthermore, there are currently large differences between the typing systems of C++ and Java. This affects parametric polymorphism and obviously has implications for the correct typing of collections which are essential for querying. For example, the ODMG Object Model has a root object type that is supported by the Smalltalk and Java bindings, but not the C++ binding. Similarly, the C++ binding uses templates to support parametric polymorphism, whilst the Java binding does not.

When looking at a particular object database, such as O_2, Suad highlighted a number of significant problems. For example, O_2 supports the Object Query Language (OQL) and C++ binding as defined in the ODMG standard, but its Object Definition Language (ODL) is totally different from the ODL defined by ODMG and differs from the type systems of Smalltalk, C++ and Java. The O_2 object model is therefore different from the ODMG Object Model. Furthermore, whilst parametric collection classes are supported in the C++ interface of O_2, as required in the ODMG standard, they are not supported in the actual O_2 system.

For the model of persistence, Suad showed why orthogonal persistence is desirable for object databases and where the ODMG standard and O_2 are again controversial. Complex object support, he argued, cannot be managed without an orthogonal model of persistence and in the ODMG standard, both the C++ and Java models of persistence are not orthogonal.

O_2 originally supported an orthogonal model of persistence, as well as persistence by reachability. However, its C++ binding achieves persistence by inheritance from a root class, called `d_object`, which is obviously not orthogonal persistence. Furthermore, this causes problems for Java, since it implies multiple inheritance, which Java only supports for interfaces. The ODMG standard is also inconsistent in other aspects of its Java binding, since it seems to support persistence by reachability, but the model is not orthogonal persistence.

The final issue of non-traditional object-oriented features dealt with issues such as relationships, name-space management and object identifiers. For example, the ODMG Java binding does not provide support for relationships. Similarly, the O_2 model does not support relationships, but does support relationships in its C++ binding. Also, both the ODMG standard and O_2 use a flat name-space, which is somewhat limiting for multi-user or distributed applications. Finally, in the O_2 C++ binding, object identifiers are directly accessible, which is unsafe. This is caused by pointer mechanisms being available in the language.

To summarise, Suad concluded that it was evident that O_2 had influenced the ODMG standard, but there were also many important differences. For example, differences in the type systems, support for parametric classes and support for relationships.

Charting Unexplored Waters: An Adventure in ODB-Land

This paper was written and presented at the workshop by Adam Taylor. The author is with High Road Innovation.

A problem that is sometimes faced by organisations today is whether to use an object-oriented programming language with a relational or object database. Documented examples (e.g. see [Loomi98]), demonstrate the practical benefits of using an object database over a relational database in certain cases. One of the reasons why an organisation may wish to do this is for improving performance where considerable navigational access is required as, for example, in a bill-of-materials or parts-explosion hierarchy. It is precisely this type of problem that Adam described in his presentation.

After initial performance problems experienced with a relational database, Adam's group evaluated several object database products and eventually found ObjectStore to be the best fit for their requirements. However, a number of problems were encountered, which can be summarised as follows:

1. An initial and serious mistake was simply to use the existing relational database model as the basis for the new object database model. This resulted in too many classes and an inflexible design.
2. Casual misuse (e.g. stray deletes) of database objects by developers often caused serious problems.
3. Transforming objects from one form to another was overly-complex, even for simple operations. Despite the transformation being encapsulated, the code was still difficult to write and maintain.
4. Many programming languages were being used in the development group, such as Visual Basic for the GUI and C++ for database development. Consequently, an in-house language interchange format was written,

with developers providing some additional ("glue") code to connect C++ classes and Visual Basic. This interchange format was extended for persistent classes and TCL (used for database loading, unloading, report generation), but resulted in many maintenance problems for the glue code whenever schema changes occurred.

After a period of development, a major reconstruction of classes and schema changes required complete database rebuilds, which reduced database-related errors to almost none. However, further problems subsequently emerged:

1. The schema was overly-complex, which caused schema-evolution problems, particularly in restructuring relationships to meet new requirements. The ObjectStore schema evolution tools were inadequate for this project.
2. The new schema caused synchronisation problems, since not all the developers were working at the same level.

Further restructuring simplified the design, providing easier maintenance, as well as enabling the development of pre-formulated queries. These changes provided significant performance gains.

To conclude, Adam noted that two of the three applications that were being developed on the object database had already been successfully deployed, with the third application due for release by the end of 1997.

Overall, there were many issues related to the use of object databases that were not foreseen. These could be summarised as integration, architectural and deployment strategies.

Object to RDB Mapping: When Attributes and Fields are Not Enough

This paper was written and presented at the workshop by Gerald Zincke. The author is with GMO GmbH, Vienna, Austria.

A common feature of some database systems today is the use of a large centralised mainframe and users "in the field" with small personal computers or laptops (for mobile computing). A typical sequence of events for remote users would be to:

1. Copy portions of the main database to a smaller machine.
2. Make offline changes to data.
3. Write the changed data back to the main database.

However, there are a number of complex issues involved with these requirements:

1. The centralised database has a complex data model. If all the data regarding a small number of business objects are required, this may involve the transfer of a large number of records to the remote offline database.
2. Data manipulation performed on the remote offline database needs to be recorded to enable changes to be correctly applied to the main centralised database.
    - New business objects inserted into the offline database will have to have their keys translated when the data are transferred back to the main database, since there are well-defined rules for key allocation.
    - The same business object may be updated at more than one offline site, so a pre-image log is maintained on each laptop for every updated object.
    - Deleting business objects is also performed by using a log.
    - Transactions must be correctly recorded and applied when data are transferred back to the main database.
3. When transferring the data back, the various changes need to be repeated against the main database. The issues presented above regarding new, updated and deleted objects are carefully applied to ensure that the main database remains consistent.

Gerald described the following general solution to these problems, which was developed using a framework. Business objects:

- Know their primary key and the rules to create that key.
- Know how they are mapped to the database.
- Know pre-image information about themselves.
- Know how to check the database to see whether they already exist.
- Know what relationships they have with other business objects.
- Can only be accessed by a behavioural interface.

So both data and model information are maintained and, using this approach, a solution to the problems outlined earlier was developed in 1.5 months. The framework also allows the same solution to be used for other applications.

Mapping Objects to an RDBMS Using Message Passing

This paper was written and presented at the workshop by Patrick Noonan. The author is with the First Union National Bank, Charlotte, North Carolina.

Patrick described a three-tier system being developed by a large commercial bank. This bank had experienced performance problems with a two-tier solution and was looking towards middleware to provide improved performance for an on-line transaction system that would support 200+ users in 8 states. The servers were large mainframes running Oracle and IMS, with 486 and Pentium PCs being used for the clients. The application uses 110 business objects and 80 Oracle tables.

Early project decisions were to isolate the client and client developers from any knowledge of the database. In this way, they were not influenced by SQL or relational table considerations. Furthermore, business logic would be maintained on the server with thin clients to support data entry and querying using windows and dialog boxes.

Mapping rules for object attributes and classes to row attributes and tables were maintained in an extended data dictionary. This dictionary was used by a custom-built code generator to build server-side code.

Persistent and non-persistent objects were managed by subclassing. Within the development team, the decision to subclass business objects from a persistent superclass caused disagreements, since some developers felt that business objects should not have any knowledge of the persistence layer. However, the solution worked and continued to be used although it was felt that major drawbacks would occur if the relational database was replaced with an object database.

The original performance requirements were that 250,000 bytes from the server should be retrieved in a single transaction within a reasonable (not quantified) time. At the present time, the largest transaction is 80,000 bytes and takes approximately 20 seconds, which is acceptable to customers, since it is relatively infrequent. Other transactions are typically in the 3-5 second range.

To summarise, Patrick felt that the system would continue to use relational databases, although he would prefer to move away from this type of database technology. Object databases were unlikely to be used in the foreseeable future, as the bank was taking a conservative view of this technology.

Conclusions

The previous summaries show that, on the whole, object data management is being successfully used for commercial applications. However, there are still a number of outstanding issues. For example, some vendors have adopted a very aggressive marketing strategy, which means that whilst their products may be used for more applications, there is also the likelihood that there are more failures (which the user community at large generally does not hear about for obvious reasons). Furthermore, from the discussions during the workshop, it also emerged that the participants developing commercial systems were not using the query capabilities provided by a particular object database product. Similarly, none of the same participants had used any public benchmarks for performance evaluations; most preferred to undertake their own benchmarking tests. On the issue of standards, there are some problems with the current efforts and a number of areas that ODMG needs to address were clearly identified. Another important issue was that, for successful commercial systems, it is essential to have suitably qualified people. Retaining such people can also be a major problem.

Workshop participants may make available electronic copies of their position papers and slides. If so, links to these will be added from one of the workshop home pages located at `http://www.soi.city.ac.uk/~akmal/oopsla97.dir/workshop.html`.

Acknowledgements

The workshop organisers would like to thank all those who took the time and effort to prepare position papers, presentations or participated directly in the workshop.

Thanks to Doug Barry for his comments on an earlier draft of this report and to Ling Liu for proofreading.

References

[Loomi98] M.E.S. Loomis and A.B. Chaudhri (eds.) (1998) Object databases in practice (Upper Saddle River, New Jersey: Prentice-Hall) `http://www.prenhall.com/ptrbooks/ptr_013899725x.html`

[Zorn95] B.G. Zorn and A.B. Chaudhri (1995) Object database behavior, benchmarks, and performance. *OOPS Messenger*. 6 (4):159-163. `http://www.cs.colorado.edu/homes/zorn/public_html/oopsla95/addendum.ps`