Migrating to Object Data Management

Arthur M. Keller*
Stanford University
and Persistence Software

Paul Turner†
Persistence Software

Abstract. We discuss issues of migrating to object data management. We consider reasons for migration, pitfalls in and benefits of migration. We also address risk management, medium term issues and the migration process. We identify options for migration to object data management. In particular, we describe the alternative of storing data in relational databases and building object-oriented applications using C++.

1. Reasons for Migration

Let us first consider some reasons for migrating to object data management. Object technology enables application development that can effectively model the organization and structure of the real-world environment. Objects encapsulating both state and behavior can concisely describe the semantics of the application while facilitating code re-use. Using object technology, data can be organized according to the needs of the application.

When a company engages in process re-engineering, object technology allows information system technology to mirror a company's business. The organization of the company can be reflected in the organization of the software, and the processes of the company can be supported by the processes of the software. Software that allows flexibility can be restructured to reflect new business processes.

2. Benefits of Migrating

There are a variety of benefits that companies expect by migrating to object data management. By programming in an object programming language, companies can achieve faster program development, improved maintainability, and better performance. Companies intend to use migration to realign software to better serve the needs to the company. Migration can facilitate support for new applications. The combination of migration and re-engineering allows better integration of various information system processes with corporate processes. Note that migration does not require complete elimination of the old systems: co-existence is possible. Experiences from Persistence's customers confirm much of the above: several telecommunications customers have reported application extensibility and ease of application upgrade through the use of object technology.

3. Pitfalls to Migration

However, there are several potential downsides to consider when migrating to object technology. How will any given application benefit from object technology? What will be lost? One common example is legacy systems. Many companies have a significant investment in legacy software and legacy data. It is critical that any migration

* Author's address: Stanford University, Computer Science Dept., Stanford, CA 94305, ark@cs.stanford.edu; Dr. Keller is also Chief Technical Advisor to Persistence Software.
This work was supported by Persistence Software, but the opinions expressed in this paper are solely those of the author.

† Author's address: Persistence Software, 1700 South Amphlett Blvd., Suite 250, San Mateo, CA 94402, turner@persistence.com..

path accommodate these. Accommodating legacy data is particularly important, because data formats, once created, can live for decades.

Over the last decade, many companies have made a significant investment in relational databases. They have converted many applications using CODASYL and other data organizations to use relational databases. These companies are rightfully leery of converting once again to object technology. An important question is whether companies can retain the large investment in relational technology, while adopting object technology. Within this question, there are several sub-issues, including how far to migrate and which mechanisms to employ. We will investigate these issues in more depth below.

Another strategic consideration is vendor relationships and longevity. For those interested in object technology, a pressing question is which object-oriented database companies will be around in 5 years. There are probably more OODBMS vendors around than the market can support. In part this is because there are currently more OODB vendors than relational DBMS vendors. Will the OODBMS vendor you choose be one of the survivors?

4. Risks to Migration

Application developers need to carefully manage the risks associated with migrating to object technology. By risks, we mean those factors that affect the successful implementation of a migration (as opposed to the value of the end result).

At the outset, we must stress that technical factors may be the least of one's worries. We have seen several projects run into trouble due to non-technical issues. For example, while many are aware of external political considerations, internal issues are often ignored. The development team must be on-board; they must view the OO migration as an opportunity, not a threat. Other issues include resources, training and tools. While such issues may be the critical risk factors to manage, discussion of them is beyond the scope of this paper.

At a high-level, we think that the increments and speed of migration may be one of the most important factors. We believe companies should not migrate all at once but should keep old systems running in parallel with experimental systems and develop a cut-over/cut-back plan for deployment.

Object data management also requires a different design process. Developing an object schema requires more care than developing a relational schema because there are more choices, that is, there are more decisions to make. There is no formal notion of an object model, so developers need to think more about object design.

More strategically, developers need to consider the second system problem – what do you do for an encore? Modifying systems that already exist has its own set of issues. Relational schemata are flat and can be restructured by access as needed, while object schemata are already structured according to the needs of a set of applications. The next application that is built using the set of objects may not have the same needs as the original application.

5. Medium-term Issues in Migration

Let us consider some medium-term issues in migrating to object data management. First, is it better to migrate the existing data into an object-oriented database, create multiple copies in both the existing formats and object-oriented database, or leave all the data in existing relational databases? Migrating existing data into an OODB requires that all applications be modified to handle the new database. Such modification is expensive and probably must be done over time. Storing data in both formats allows a transition period but entails integrity

problems. Which copy of each data item is the latest one? What procedures are there for synchronizing the data in the two formats. How expensive is it to maintain both systems? Concurrency control is also an important problem with the replicated data approach. The last alternative is to keep the data in relational databases and continue using existing applications, but write new applications (and also re-engineer existing ones) using object technology. For many situations, we believe this last approach is the most flexible and safest approach. Developers can afford the time to implement and evaluate without putting existing users at risk. Altering migration plans is much easier and feedback from evaluation cycles is encouraged instead of avoided. Persistence Software provides a product, Persistence, that provides a C++ to relational database solution.

A related issue is that of the degree of migration. Application developers should be mindful of the appropriate data source for an application. Even when object databases become a reliable and tested technology, relational databases will always be better suited for certain types of data. Even beyond the legacy database issue, there is going to be an ongoing need for a connection between objects and relational data. Planning an object-based information system involves determining what kind of data storage to use for what types of data. In our experience, we have found that telecommunications applications are good examples of using C++ and RDBMSes together. Much of the data is well-suited to an RDBMS (customer information, call records, billing information, etc.) and, because of the premium on leveraging information, high-performance C++ applications are desired. These applications often require quick turnaround and must be enhanced on short notice.

Can all applications use the same object schema, or do we need a different object schema for each application? Relational databases support views and queries that allow applications to organize their data according to their needs for representing data. Object-oriented databases do not support the view concept, and their queries do not support data reorganization. When there are future corporate reorganizations or mergers and acquisitions, it is likely that there will be the need to integrate multiple object schemata, but object-oriented database systems have little or no support for object schema integration or object views.

What techniques exist for integrating multiple object schemata? In the near term, no commercial technology for such integration exists, merely limited experiments in the laboratory. In the long-term, we believe that these problems will be addressed by a combination of deductive and object-oriented databases. With this combination, different programming teams working on different parts of a system will describe declaratively what basic functions they want–the deductive part. Then they will throw the specifications together in a compiler that will synthesize the specifications into a single declarative structure and generate objects to implement basic functions. Above this structure, programmers will develop detailed algorithms using object technology using the generated object schema. We can see the beginnings of such an approach with products like Persistence, where the object schema and code to support it are generated automatically, while detailed application algorithms are written on top in C++.

6. Future of Relational Databases

Relational databases provide very little flexibility for data structuring. In comparison, object databases provide more flexibility. Relational database vendors are working on adding more flexibility and generally incorporating many of the features available in OODBMSes. Though the current proposal for SQL3 is too complex, it is an indication that relational database vendors recognize the need to provide more support for flexibility. But despite these

developments, there will still exist an "impedance mismatch" between object programming languages and enhanced relational databases, so the niche for interfacing functionality like that provided by Persistence will continue to exist. We expect that the coming improvements in data structuring, representation, and storage will make the combination of relational databases with access through Persistence comparable to that claimed for object-oriented databases.

7. Future of Object Databases

As previously mentioned, the lack of a robust view mechanism makes it difficult for objects designed for one application to be effectively re-used by other applications while retaining the encapsulation central to the object paradigm. However, adding views to object-oriented database systems is still a research problem that has not even been solved in the laboratory yet.

Object-oriented database vendors now recognize the need to integrate with relational databases. Furthermore, object-oriented database vendors are rapidly expanding their suite of support tools. Currently, most OODBMS support tools are significantly behind those of state-of-the-art relational DBMS products.

We do not expect the object-oriented database market ever to exceed the size of the relational database market. Consequently, there are more companies producing object-oriented databases than the market will support, as there are more OODBMS vendors than relational DBMS vendors.

8. Risk Management

We identify several principles for risk management in migrating to object data management. Most importantly, do not migrate at once. Instead keep the old systems running in parallel with the new (experimental) systems until the latter have been proven successful. Develop a cut-over/cut-back plan. Choose a simple application to migrate first; build in adequate evaluation and refinement time.

Furthermore, insure that sufficient resources are available to both the new system and the maintenance of the old. We observed a severe problem in one project due to personnel being required to maintain the existing system while trying to design a new OO implementation. Do not hesitate to bring in outside help (or second opinions) *early*. In our experiences with the telecommunications industry, the willingness to bring in outside expertise has been a success marker.

Organize data as most appropriate for the data and applications using it. Some data really does fit best in relational databases. Keep that data in the relational databases, and use object interfaces for object-oriented programming. Insure that you have the necessary expertise: you will need both OO and relational skills. Be wary of religious factioning (OO vs. RDBMS) amongst the developers (we have witnessed such factioning far too often).

Create a clean, quality design first, tune later. However, do make sure you take the time to effectively evaluate and tune. In one project, we observed upwards of a year of wasted time due to inadequate evaluation and tuning periods. If you choose the OO language and RDBMS approach, you will find the maturity and variety of measuring and tuning tools very helpful.

Take extra care in developing object schemata. There are more decisions and choices than there were in developing relational schemata. The absence of view mechanisms for object-oriented databases makes further migration more difficult. Remember any schema you create will need to be used by many applications and for a long time. Consider the second system problem. How well will object schemata and classes developed for the first (experimental)

application fit usage patterns required by subsequent applications?

9. Conclusion

We have analyzed the process of migrating to object data management. We have considered risks and benefits and how to mitigate those risks. We have observed that often, much of the benefit of using object data management arises from object-oriented programming and does not require storing data in an object-oriented database. In fact, continuing to store data in a relational database, and accessing that data using a product like Persistence to allow programming in an object programming language provides much of the benefit at significantly reduced risk and cost, as compared with migrating to an object-oriented database.

References

Thierry Barsalou, Niki Siambela, Arthur M. Keller, and Gio Wiederhold, "Updating Relational Databases through Object-Based Views," *ACM SIGMOD*, Denver, CO, May 1991.

Michael R. Brodie, *Migrating Legacy Systems: Gateways: Interfaces & The Incremental Approach*, Morgan-Kaufmann, 1995.

R.G.G. Cattell, *Object Data Management – Object-Oriented and Extended Relational Database Systems*, Addison-Wesley, 1991.

R.G.G. Cattell (ed.), *The ODMG-93 Standard*, Addison-Wesley, 1993.

Arthur M. Keller, "Unifying Database and Programming Language Concepts Using the Object Model" (extended abstract), *Int. Workshop on Object-Oriented Database Systems*, IEEE Computer Society, Pacific Grove, CA, September 1986.

Arthur M. Keller, Richard Jensen, and Shailesh Agarwal, "Persistence Software: Bridging Object-Oriented Programming and Relational Databases," *ACM SIGMOD*, May 1993.