Customer Q & A

A major international company in the Process-Automation Products and Technology business evaluated Objectivity/DB as a core technology for their digital plant architecture. The initiative was to build a unified persistence platform that enabled all of the company’s disparate software applications to store and share data cohesively.

Objectivity Systems Engineer Todd Stavish presided over the interview program that is transcribed in this article. The interviewee is the chief architect of the project, whose name and company can not be released due to company disclosure policies.

Q. Could you briefly describe the nature of your development project?

Objectivity is being used at our company in several areas as:

1) the database for an existing console product
2) the database for our batch historian
3) the integrated database for a future product line

In these three areas Objectivity is being used to store configuration data and to store production data. In the first release of the future product we anticipate a very complex configuration of over 500 classes with a “web” of over 1,000 associations. The configuration database will store approximately 100 Mbytes of configuration. The production database will store several GBytes of data.

Q. What were the selection criteria for your database choice?

The selection criteria for us included:

1) Reliability – the database must be able to operate in a 24x7 situation for many months at a time.
2) Concurrency – the database must be able to be “pounded” on by several users and many processes at the same time. We have several transaction models in place for dealing with different levels of access.
3) Performance – with such a complex schema, i.e., many different objects with very different compositions and many associations between classes, performance is a key consideration.
How One Automation Vendor Gained Competitive Advantage by Choosing an Object-Oriented Database

Q. How did you end up evaluating Object-Oriented Databases?

Object-Oriented Database Management Systems (OODBMS) and Relational Database Management Systems (RDBMS) share a common goal; to manage and protect mission-critical data used by applications and individuals. In this regard it appears that any problem that may be solved by one database system may be solved by the other. This conception is not true. There are fundamental technical differences between OODBMS and RDBMS that make certain classes of problems tractable by OODBMS but not by RDBMS. Productivity differences in application development between using OODBMS or a RDBMS are sourced by these same fundamental technical dissimilarities. OODBMS possess certain features not provided by RDBMS that allow applications to offer benefits that would otherwise not be possible.

OODBMS combine an object-oriented metaphor with conventional database management functionality. This combination allows database technology to be used in applications that were prevented from using database technology in the past. For instance, historically we managed our own storage of persistent data through flat files.

Q. Describe how database performance played a role in your decision.

RDBMS decompose all data into relational tables. The rules for this decomposition are collectively referred to as the first, second, and third normal forms. These rules provide a simple and powerful way to organize data into a structure that provides quick and convenient access to information. A good example would be a checking account. Individual checks are represented as a row in a table with columns for the check number, date, payee, and the amount. A RDBMS could provide a software application expedient access to this data.

Another example would be blocks in a Distributed Control System (DCS). Here the RDBMS would set up relational tables to hold information pertinent to a block. Much of the information in this problem domain is hierarchical and may be modeled using 1-to-N or N-to-M sets of relationships. To show these hierarchical relationships in RDBMS, another table(s) would be set up mapping one row from the first table to another row(s) in some other table. As an example, consider that the outputs of one block would be mapped to the inputs of other blocks. With a RDBMS, a separate table must be inspected at the time of the query to determine which output is routed to which inputs. This operation is an example of selecting a set of rows from a table. If we need to determine what types of blocks receive input from a specific output, then multiple tables must be inspected at the time of the query. This operation is referred to as a "join," and it is expensive. The expense of a query dramatically increases as more tables are brought into consideration.
How One Automation Vendor Gained Competitive Advantage by Choosing an Object-Oriented Database

Alternatively, an OODBMS provides support for directly modeling the block and the mappings of block outputs to block inputs. No run-time selection from a table is required to determine which inputs are mapped to a specific output. No run-time joins of multiple tables are required to determine what types of blocks receive input from a specific output.

The OODBMS provide explicit support for 1-to-N and N-to-M relationships. These relationships and the semantic information they represent become an integral part of the data managed by the OODBMS. Not only is the data available to an application, but this semantic information is available without the expense of a run-time selection join. A RDBMS does not maintain this semantic information.

The way that an OODBMS stores information on disk is the same as that in memory representation. RDBMS store data in a format on the disk, which is different from the in-memory form needed by applications; selects and joins are used to bridge these two formats at run-time. This fundamental dissimilarity causes dramatic differences in run-time performance of applications that deal in problem domains with 1-to-N and N-to-M relationships.

A set of benchmarks run by Sun showed, for certain classes of data, [that] an OODBMS would outperform a RDBMS by at least one order of magnitude. Our company’s applications primarily work with data that fits this pattern.

Q. What were your scalability concerns?
Most RDBMS are based on technology using a single server with multiple clients. Some OODBMS follow this pattern, but many are based on peer-to-peer technology. A peer-to-peer database system does not have the same potential performance bottleneck that a single server system poses. A peer-to-peer database system makes extensive use of the compute power and storage capacity on the client’s computer.

In a peer-to-peer system, performance problems are often solved by adding more compute power in the form of additional computers. In a client/server system, adding more compute nodes makes the bottleneck problem at the server worse; the only option is to increase the server’s horsepower.

Peer-to-peer systems scale better than client server systems. A peer-to-peer database system may easily and incrementally grow to handle the demands of an expanding application.

A fully distributed OODBMS has the capability to use disk storage anywhere in the network. This [distribution] allows much flexibility in adding incremental storage capacity. An additional performance
benefit may also be realized by placing persistent objects on the computer's disk where that information is most likely to be used.

Q. How was your productivity affected by the use of an OODBMS?
The data model used by a RDBMS is not natural from a programmer's point of view. A software engineer working in object-oriented languages like C++, Java, or C# is used to dealing with data abstractions. Object-oriented languages directly model these abstractions and bind data and operations on that data together into a single entity. An OODBMS makes these common object-oriented data abstractions persistent. From a code view, there is almost no difference between a transient instance that lives and dies in RAM versus a long-lived instance that is managed by the OODBMS. This means that the persistent entities are made available to the engineer in a seamless fashion. The OODBMS is not obtrusive in the application's code.

Alternatively, a RDBMS is very visible to a software engineer working in an object-oriented environment. Here the engineer is fully cognizant of the fact that two representations of the entities exist: the natural language version of an object and its RDBMS counterpart, which may be spread over multiple relational tables. This mapping between the tables in the RDBMS and the natural in-memory form of an object must be managed with code provided by the software engineer. There is no alternative here; the mappings must be identified and code must be written to handle the mappings.

The basic types of information stored by a RDBMS are very limited. A RDBMS has support for primitive data such as: (integer + real + text + date). A RDBMS has no provision for storing complex data abstractions based on aggregation or inheritance.

An OODBMS also maintains semantic information that is not available in an RDBMS. Relationships that must be derived in a RDBMS through selection and join operations are explicitly kept in the OODBMS. The operations that may be invoked upon entities managed by an OODBMS are defined by the data abstraction that the engineer designs. The RDBMS has no way to bind behavior to objects. This must be done through convention, and it represents a large opportunity for introducing errors.

The data model produced by the software engineer during the analysis and design phase of a project may be directly and simply mapped to an OODBMS' schema. With a RDBMS, data abstractions must be decomposed into a set of tables that conform to the normal forms of the relational rules.

There's also an impedance mismatch between the way the data is available through a RDBMS versus an OODBMS. An engineer working with an RDBMS works in SQL; all access to information managed
How One Automation Vendor Gained Competitive Advantage by Choosing an Object-Oriented Database

by the database is framed in the context of SQL. SQL is an excellent way to work with data on an ad hoc basis, but a developer wants information in terms of data abstractions that weren’t identified in the analysis and design phases. This is not possible using the SQL access offered by a RDBMS.

A developer working with an OODBMS directly sees the same data abstractions in the database that were identified during analysis and design. No queries are used. This gets back to the fact that objects managed by an OODBMS appear to an application developer in a seamless fashion.

OODBMS directly support reuse through inheritance. This concept is absent from an RDBMS. Commercial grade container libraries and other third party class libraries may be easily stored in OODBMS. This is not possible with a RDBMS.

All of these differences between OODBMS and RDBMS result in a real and sizable productivity gain for developers working with OODBMS.

Q. Describe your high availability requirements.

Conventional RDBMS deal with fault tolerance issues and high data availability with hardware based solutions. An example is the use of RAID technology to guard against medial failure. These options are also available to an OODBMS.

OODBMS that are based on peer-to-peer technology are designed without any single point failures. They are set up to degrade gracefully. If part of the distributed information store is not available, the remaining information can still be accessed. These systems are more resilient to catastrophic failures.

Some systems also offer additional protection by replicating important pieces of data across multiple compute nodes. In this situation, failure of a single node does not result in loss of data that was replicated and therefore available on some other node.

Q. Are there any additional OODBMS features you’d like to discuss?

Sure. OODBMS commonly support certain features that are not available in a RDBMS. These features would be exploited by any of our applications to offer additional value and benefits to customers.

Object Versioning is important to our requirements. This feature allows the OODBMS to manage multiple versions of the same object. In its simplest form, Linear Versioning makes a sequential series of the same object available to an application. Each subsequent version represents the incremental change in the object from the previous version to the next. The idea of a “current” version is also
How One Automation Vendor Gained Competitive Advantage by Choosing an Object-Oriented Database

supported. These features would be used by many of the entities in our applications. Good candidates for versioning include: (batch recipes + process graphics + alarm configurations + block configurations).

Schema Evolution was critical as well. This feature allows the OODBMS to deal with objects as they evolve from one release of our software to the next. Procedures to upgrade an “old” object to a “new” object are handled by the OODBMS.

Off-line databases are a big benefit to our customers. OODBMS offer the concept of on-line and off-line databases. To our company this means a single application may be written to handle both off-line and on-line configuration. To our customers this means that they may run their plans with the on-line database, but prepare for future plant expansions with off-line databases. The configuration work with the off-line database may actually be performed at a different site. The resulting off-line database containing the “new” configuration for the plant expansion may be transported to the site via network, USB drive, or laptop. A simple menu selection could merge the off-line database with the customer’s on-line database.

We also found the need for long term transactions. CAD applications were some the first users of OODBMS. These applications as well as our applications require transaction models that are not supported by conventional RDBMS. A user may require long-term access to information managed by the database.

As an example, consider a process graphic that may take days to be finished by a configurer. The database must provide exclusive access to the graphic until the configurer is finished and decides to check the graphic back into the database. Only then would the graphic be available to the rest of users.

OODBMS provide this type of long-term transaction in addition to the conventional locking strategy supported by RDBMS.

Q. Are there any business benefits that your customers experienced that you can attribute to choosing an OODBMS?
Yes, our mandate on this project was to show tangible results to our lines of business before we made the complete commitment to OODBMS technology.

In order to do this, we built a beta version of the new platform and ported over our existing product suite. Our consultants took the beta release to some of our larger customers to use with real data (although not on a live system).
How One Automation Vendor Gained Competitive Advantage by Choosing an Object-Oriented Database

The results are encouraging. The Uniroyal deployment experienced a 60 percent reduction in both configuration and training costs and a 30 percent reduction in installation costs.

On the Monsanto pH control benchmark, we were able to complete in six hours what our competitor could not complete in two days. And with the Dow Chemical experiment, they measured a 30 percent reduction of total project costs.

Obviously, most of these benefits are due to the fact that the OODBMS-based architecture doesn’t require the same level of DBA administration, installation, and configuration work, nor did we see as much client-side consultation from our services group.

The real excitement will begin when we start introducing new product features based on the OODBMS benefits to the actual products themselves.

Conclusion
As the interview indicates, an OODBMS can be used with a wide variety of applications. However, it is a tool that coexists with other persistence tools in a modern software developer’s toolbox. OODBMS are not a replacement for all relational or other legacy database applications.

It is appropriate to use an OODBMS instead of RDBMA when you need to:
- avoid DBA costs
- support a customer base that has many different operating systems and hardware platforms
- build an architecture that includes distributed processing of remote or disparate information feeds
- support a suite of applications with different data models that still need to share information
- respond to incoming information in real time
- scale with respect to the database size
- increase the number of simultaneous database clients

To discover if your software application would benefit from the power of Objectivity/DB contact Objectivity at www.objectivity.com to participate in a formal database selection criteria process. The selection process determines if your business or technical needs match the solutions provided by OODBMS architecture. This service is provided free of charge.