PANEL: Issues in Object Database Management

Jacob Stein, *ServioLogic, Moderator*

Tim Andrews, *Ontologic*
Bill Kent, *Hewlett-Packard*
Kate Rotzell, *Versant Object Technology*
Dan Weinreb, *Object Design*

While the availability of commercial systems from several vendors indicates maturity in object database management technology, there are numerous issues which remain. This panel will attempt to expose and discuss several of these issues.

Part of the performance advantage realized by object database management systems comes from linking application programs with the database management system, and the use of large virtual memory caches. This is acceptable in engineering applications where previously large amounts of data were read from the file system into an application program's data space. However, the potential impact on database integrity of giving application programs direct access to very large database management system caches will be of great concern in commercial applications. How can these concerns be addressed with minimal impact on the performance advantage of object database management?

There appear to be two distinct approaches to object query languages: extensions to SQL and programming language extensions. SQL extensions might provide a fast path to a standard, but would have the traditional impedance mismatch problems associated with embedded query languages. Language extensions are elegant in that they use the same syntax as the programming language and do not suffer from impedence mismatch problems. However, language extensions would probably make standards more difficult to arrive at as it would require the coordination of extensions to multiple languages. Which of the approaches is most appropriate? In which order should these two approaches be addressed by the object database vendors and standards bodies?

One might argue that a good deal of research in relational theory has had little or no impact on commercial relational systems (e.g., relational dependency theory). From a vendor's perspective, what are the hard, interesting research issues whose resolution would allow you to build better systems?

Tim Andrews, Ontologic

In contrast to some Object Database vendors that realize high performance via a one track approach of mapping the database address space into the process address space and using large main memory caches, ONTOS achieves its performance by taking advantage of the flexibility of its abstract object interface and the client/server process distribution model to allow development of the right tool for the problem. Thus we do not rely on any one tactic to solve all performance problems. Using ONTOS one need not compromise the database process by allowing the programmer to run amuck with pointers, allowing us to use the same architecture for both commerical and CAD applications. Our performance derives from many sources including:

- Fast and flexible development, allowing developers more iterations in building applications in which to try different strategies for performance gain. Thus our approach has always been a general rather than a specialized approach to

solving the performance problem which will work without change in a commercial environment.

- Asynchronous communication between client and server, allowing the client process to do profitable work instead of waiting for the server to complete.

- Flexible client memory management, allowing developers to transparently try different client memory management strategies, including but not limited to large main memory caches.

- Flexible storage management, allowing the use of different disk representations of objects in a manner transparent to the application programmer. For example, this allows the use of compressed representations for small CAD objects to cut overhead significantly.

Regarding query languages, we have always maintained that SQL is the only query language worth standardizing for Object Database systems today. This is not because it is a great language, but because it is "intergalactic dataspeak". The so called "impedance mismatch" can be mitigated with an intelligent interface; the real mismatch today occurs as much from the embedded-language-that-gets-pre-processed approach as it does from the differing type models. ONTOS eliminates this part of the problem by providing a class based interface to the SQL language so that both queries and their results are treated as objects. The more interesting theoretical issue is combining predicate based programming with imperative, or procedural programming. We believe this is one of the areas where Object Database Systems can provide real power, as complex application development should have comfortable access to either programming paradigm. ONTOS allows the programmer to intermix the extended SQL and C++ freely in both directions; SQL queries can use arbitrary expressions including C++ member function calls. This allows the programmer to use predicates for set based access mixed with navigation for deterministic access (such as tree walking).

The last issue we were asked to address is research areas. Relational theory research has not had a large impact in the object arena largely because the problems of commercial systems are not addressed. Data model issues are largely secondary problems. The hard problems arising today are:

- Advanced concurrency control. This includes non-serializability, multiple levels of transaction visibility to support group activities, hybrid protocols that support more than one concurrency model, etc.

- Methodological issues. There is as yet no accepted design methodology for object databases. As larger organizations with larger projects and staffs become involved, this will not be acceptable. The whole notion of "moving the domain of discourse up another level" or "after objects, what?" needs exploration.

- Models for system areas lacking them. Is there an object model for concurrency control, transaction management, security, and the host of other things that a database system does that are not addressed by the basic data model?

- Large class library organizations and interfaces. Re-use is only as good as the developer's ability to find the right piece to re-use. There are as yet no accepted interfaces for assisting humans in this process, which will become increasingly important as larger and larger class libraries become available.