

Editor: Roberto V. Zicari

TechView Product Report: Objectivity/DB

December, 2008

Product Name: **Objectivity**
Company: **Objectivity, Inc.**
Respondent Name: **Leon Guzenda, Chief Technology Officer of Objectivity, Inc**

Support of Programming Languages

1. How is your product enabling Java, .NET/C#. C++, developers to store and retrieve application objects? What other programming languages do you support?

Objectivity/DB has Application Programming Interfaces that provide persistence and database management capabilities for C++, .Net for C#, Java, Smalltalk and Python. Other languages can use objects created in any of these languages across any of the platforms that Objectivity/DB runs on. The C++, C# and Python APIs support persistence by inheritance. The Java and Smalltalk APIs also support persistence by reachability. The C++, Java and Smalltalk APIs are compatible with the ODMG'93 standard.

Queries

2. Does your product support querying? If yes, can you describe the querying mechanism, giving examples of queries?

Yes. There are two styles of query:

- a) Using iterators that inherit from the target class and predicates in the native language format, e.g. if Person is an object class that has name as an attribute and age() as a method the query would look like this:

```
ooItr(Person) iP;  
iP->Scan("name=="Fred" && age()<21); // Initialize the iterator  
while (iP->next()) {  
    iP->doSomething();  
}
```

- b) SQL++ is an enhanced version of SQL that can handle inheritance, associations and nested fields. The SQL++ equivalent of the above query would look like this:

```
SELECT * FROM PERSON WHERE NAME="FRED" AND AGE<21;
```

The SQL++ syntax can also be used to qualify an iterator, i.e. in the Scan

parameters in example 2a). There is also an ODBC interface, making it possible to access Objectivity/DB databases with standard SQL tools.

Queries can be executed synchronously within a client process/thread or asynchronously via a distributed Parallel Query Engine (PQE). Objectivity/PQE has replaceable components in both the client and agent libraries. The client can invoke a splitter to direct queries to individual database or container data query agents. The query agent can invoke filters, e.g. image recognition algorithms, to further refine the results. The filters could also act as gateways to legacy databases or data files.

3. How is your query language different from SQL?

In case 2a) above the iterator is used in a native language loop and it starts returning qualified objects as soon as they are found. Like SQL, it uses indices if they are available. However, most relational DBMSs do not return any results until the whole query has been executed and the resulting view has been constructed.

Objectivity/DB supports uni-directional and bi-directional 1:1, 1:many, many:1 and many:many named associations (relationships). It automatically constructs iterator methods for rapidly traversing these links. Objectivity/SQL++ will exploit a named association, if one exists, rather than constructing join tables.

RDBMS servers generally parallelize a query based on system knowledge of the placement and population statistics of the target rows. Objectivity/PQE allows the user to control the degree of parallelism based on semantics. For example, envision a federation with a database for each country in the world. Each database has a container for each city in that country. The containers have objects that include Images. If the client wants to find all images taken on Wednesday afternoons in zoos in Europe the client side splitter can interpret “Europe” as corresponding to 49 country databases. PQE will launch up to 49 parallel queries to the respective query agents and marshal the results. Having found qualified images the query agents can then apply filters, e.g. to return only those images that contain people wearing red hats, before returning results to the client.

4. Does your query processor support index on large collections?

Yes. Objectivity/DB supports B-Tree indices at the federation, database and container (logical sub-division of a database) level. It also supports collection classes and there is a multi-dimensional indexing class library.

Data Management

5. Do you provide special support for large collections?

Yes, the scalable collection classes include ordered and unordered lists, trees and other standard C++, Java and C# collection types. The collection entries can span multiple physical files.

6. How do handle data replication?

Objectivity/HA (High Availability) provides synchronous database replication using a quorum based co-ordination mechanism.

7. How do handle data distribution?

Yes, Objectivity/DB meets all of the Codd & Date requirements for a distributed DBMS. The Objectivity/DB logical storage model is that of a federated database of databases stored anywhere that clients can reach them. There is a seamless “Single Logical View” address space that can scale to Yottabytes. The client does not need to know where databases are stored and client processors do not need to be individually configured to reach the various servers involved in an Objectivity/DB environment.

The largest deployment that we can discuss publicly spread a Petabyte of objects across tens of thousands of databases stored via 140 data servers and an HPSS mass storage cluster. Some telecom equipment deployments involve tens of thousands of physical locations.

8. How do you handle schema evolution?

Objectivity/DB implements the ODMG’93 metadata model. It has an extensive library of schema evolution and object migration methods, including complex operations such as changing the inheritance hierarchy and splitting an object class into two object classes and an association.

The database designer can decide when to migrate objects. They can be migrated in a single operation, when reached, or left as is and presented in the new format. Schema and object migration can be handled online.

9. Please describe an object lifecycle when an object is loaded from a database: When are members of an object loaded into memory? When are they discarded?

Objects are transferred from disk to memory, potentially over a network, in fixed size physical blocks called pages. Their size can vary across the databases and containers in a federation, but they are typically 8KB or 16KB in current filesystems. The overheads in reading a page and transferring it across an Ethernet link are about the same as transferring individual objects and substantially less if objects are tightly clustered, e.g. a Paragraph and its associated Sentences and Words.

When an object is required by an application the object identifier (OID) is decoded to locate the physical data servers, filesystems and files involved. The file is opened and the page is located and transferred into a cache controlled by the application process/thread. The actual object is located and activated for the architecture (32/64 bit etc.) and language involved. The application receives a reference (an OID) or a handle (composite smart pointer) that is used to manipulate the object in the same way that a standard ref or pointer is used in the native language.

The object may have associations to other objects and multiple varying length components. These constructs are not activated unless the application uses them. Neither is any other object in the cached page touched unless the application requires them.

The page is marked as requiring a write operation if any of the data in it is changed. The application can specify the initial size, extension size and maximum size of the cache. If the Objectivity/DB kernel needs more space to create or fetch a page it

generally discards the Least Recently Used page, which may involve writing it to disk. The changed data is not visible to other transactions until a successful commit.

Data Modeling

10. Which Data Modeling notation do you recommend for the design of your data?

We don't recommend a particular notation. However, it should be object-oriented, handling inheritance, and have the ability to model 1/many to - 1/many associations. UML meets these criteria. Our own tools are implemented as Eclipse framework plugins and we take an active interest in the Eclipse Modeling Framework initiative.

Integration with relational data

11. Could you integrate flat relational tables into your object database? If yes, how?

Yes, using a customized Parallel Query Agent query server. The same mechanism (a replaceable component in each query agent) could be used to access any database, structured file or WWW search engine.

Transactions

12. How do you define a transaction? Pls. use a simple example.

Objectivity/DB implements standard ODMG'93 and ACID transactions, with start, commit, abort or commitAndHold semantics. A Session class can be used to launch thread pools, each having its own transaction or within a single transaction.

Transactions can have classical semantics or Multi-Reader, One-Writer semantics. They can also perform automatic recovery in the event that another client has crashed holding locks.

Example:

```
OoTrans t;  
t->start();  
// do stuff  
t->commit();
```

The commitAndHold operation is a checkpoint operation. It leaves objects in the same state as they were before the operation. However, there is an option to downgrade write locks to read locks for long running transaction.

13. Is there a way to discard objects that have been modified in the current transaction from memory?

Yes, closing them (with a close() method) makes them candidates to be discarded once all other objects in the page are closed. Iterators automatically open and close objects as they loop.

14. Do you provide a mechanism for objects on clients to stay in synch with the committed state on the server? If yes, please describe how it works.

Yes. However, note that Objectivity/DB is a distributed processing environment, so there may be no servers. For example, a cluster of processors that access a multi-ported RAID will see local files rather than a data server.

The committed (persistent) state of an object may change while a client has a cached copy of it. The client will automatically receive the latest copy when it starts a new transaction and accesses that object again. If it tries to update the object it will receive an exception (because the kernel had to request an update lock from a lock server) and it must refresh the appropriate (container level) part of the cache before it proceeds. There is also a publish/subscribe event notification library that can notify registrants when certain kinds of event occur, e.g. creating or deleting an object of a particular class, or updating a nominated container.

15. How are transaction demarcations for your product expressed in code?

There can be explicit calls to transaction methods or implicit actions as a result of setting Session parameters.

16. Describe the strategies possible with your product for concurrent modifications by multiple clients.

The following strategies are supported: Multi-Reader One-Writer, Shared Read and classical Exclusive Writer. Objectivity/DB supports automatic hierarchical Gray-Code locking, implemented via lock servers that each control one or more databases in a federation. Locking a container for update automatically obtains Intention to Write locks on the enclosing database and federation.

Persistence

17. Are there requirements for classes to be made persistent? If yes, please describe them.

a) Persistence by Inheritance (C#, C++, Java, Python and Smalltalk)

A persistence capable class must inherit from an Objectivity/DB or ODMG'93 base class. However, it is still possible to create transient instances of the class.

b) Persistence by Reachability (Java and Smalltalk only)

An object becomes persistent once it is reachable (by name, association etc.) from another persistent object or root, such as a Smalltalk dictionary entry. This mechanism can impose garbage collection overheads, so it is not recommended for true real-time applications.

18. Does your product require enhancement of application classes for Transparent Persistence? If yes, briefly describe the additional steps required for a build process of an application.

C++ objects require a header file that defines the inheritance from a persistent base class. This adds an extra eight bytes to the raw data size of an object.

Storage

19. Does your product operate against a single database file or are multiple files required?

Objectivity/DB always requires a federated database file that contains the schema, catalogs, backup and security information. However, a process can operate on a single database file once it has cached the appropriate schema information.

Architecture

20. Does your product support multiple clients connecting to a single database?

Yes.

21. For which Operating Systems is your product running?

Linux (Red Hat variants), Unix (AIX, Altix, Irix, HP/UX and Solaris) and Windows (NT, XP, Vista and Server variants). All versions interoperate across all languages.

Application

22. What kind of applications are best supported by your product?

Data or compute intensive applications, either embedded or service oriented, that make extensive use of relationships between objects and inheritance.

Performance

23. What benchmark do you use to measure the performance of your system?

We use multiple benchmarks, including OO7, custom built tests and applications supplied by our customers.

"Useful Resources"

<http://www.objectivity.com> has extensive technical publications, case studies, webinars, free web based training and a 60-day free trial download.