

Editor: Roberto V. Zicari

TechView Product Report: ObjectStore

Nov. 19, 2008

Product Name: **ObjectStore**
Company: **Progress Software Incorporated**
Respondent Name: **Don White, ObjectStore Engineering Manager.**

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?

Currently ObjectStore provides an interface for Java, C++ and C#. ObjectStore has been driven by the principals of complete language support and transparent persistence. ObjectStore allows the least intrusive integration with a language binding possible while supporting all language features and structures. All access to persistent data is done through the context of a transaction. This context allows users to access persistent data sizes that far exceed the amount of virtual memory available to a process. All persistent access is done in context to a transaction and all necessary ACID properties are automatically handled by ObjectStore. All data is accessed using the natural means of navigating relationships as allowed in the language of choice. ObjectStore does not require users to modify their source code to enable classes to be persisted. It also does not require users to modify their source code with load and store or even lock and unlock. ObjectStore allows the exact same user code to work seamlessly in transient as well as persistent context.

For C++ ObjectStore enables the user to have a persistent heap. Users can make data persistent by simply allocating an object in the persistent heap using an overloaded operator new, which is implicitly provided. Once an object is allocated in persistent heap then it will seamlessly move between disk storage and virtual memory as required by the application navigating through natural language pointers.

For Java ObjectStore again supports a seamless persistence, however persistence is primarily achieved by reachability. Once an object becomes connected to something persistent then the object is made persistent automatically. This prevents any dangling references which is not a concept that exists in Java. It should be noted that a JDO interface is also supported. Again this model is very similar to our own native model where persistence is achieved through reachability. The key differentiation between JDO and our native support is a richer set of language feature support.

ObjectStore currently supports the ability of C# to access a C++ database using a .NET/COM bridge. The next release of ObjectStore will have improved support for

Microsoft .NET Common Runtime (CLR). Customers will be able to write Microsoft .NET applications that execute within the .NET CLR and can directly access ObjectStore's C++ interface. ObjectStore C++ databases will be concurrently accessible from both managed and unmanaged C++ applications.

Queries

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

Yes, ObjectStore supports a variety of means to access data.

ObjectStore provides a query mechanism across all provided collection classes. The C++ interface provides sets, bags, lists, dictionaries. The Java interface provides support for the JDK collection types.

Queries are most commonly submitted as textual expressions to the query processor as part of the support API. The expressions allow users to return a particular set of objects where the set is restricted based on a variety of user supplied qualifications. Users can use Boolean expression that restrict on data member values, member function returns and can extend expression on data found through or simply the existence of relationship with other instances (of any class). Queries can also be precompiled for optimization and reuse. Queries will also optimize execution based on the existence of indexes as well.

ObjectStore also provides support for queries across type extents as well. We support an optimization to quickly determine the minimal set of pages in the database that contains instances of type to be searched.

ObjectStore also supports an Xpath. Customers can now submit through a utility or API Xpath expressions that will return a qualified XML representation of the database. Again the existence of indexes will be utilized wherever the index applies to the qualifications in the Xpath expression.

3. How is your query language different from SQL?

SQL provides a non procedural mechanism for finding data. It allows both restriction and projection. ObjectStore's query mechanism can also provide a non procedural means of accessing and restricting data but the restrictions are much more directed in terms of the data model. SQL will require users to materialize relationships, in essence create them as part of the query expression. ObjectStore maintains the users data model from the desired in memory representation all the way through to the persistent storage model. All the relationships stay constructed and are available as part of the query syntax and its direct execution.

Probably the most important thing to understand is ObjectStore will basically give you the ability to utilize the natural structure and behavioral model associated with data. Typical implementations of SQL are far more restricted on supported data models, usually boiling down to some representation of a relation as a table.

It is not realistic to represent the simplest form of some common data structures as tables, even a linked list can be complicated never mind something more advanced like a tree or a graph. If you compromise your data model then you compromise your

behavior model. As a result you are left to compensate by separately coding an application means to handle your data vs a persistent means to handle your data.

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

Yes, ObjectStore supports both ordered and unordered forms of indexing. Indexes are also managed on separate data pages from their main collections allowing the indexes to be compressed and fetched with the least amount of overhead possible. This applies to all variety of collection classes supported, which includes dictionaries that are like a structure with an index built in. ObjectStore also supports the ability to index member functions and even allow for customized indexing (both ordered and unordered) for any user class. It is also important to note that ObjectStore also provides mechanism to automate index maintenance in the event of modification to indexed values.

Data Management

5. Do you provide special support for large collections?

Yes, our collection classes have been designed to be space efficient, which is needed when a structure is put on disk storage, and structured to allow fine grain concurrency, which is important for multi-user applications.

It is also possible to inherit from our collection classes to plug user customized collections.

6. How do handle data replication?

ObjectStore provides an asynchronous replication that is managed as an independent process that interacts with our server. This removes the costs and burdens of having to impose replication management at the application level. It is also possible for the replication process to be on entirely different network node then the source database server or even the replicant server.

7. How do handle data distribution?

ObjectStore can be utilized in two ways, a complete data access library that is all data access is done in the application process or can be used in client/server mode, where the server is in an independent process from the application. In client/server mode, the application and server can be on independent network nodes and each node can be comprised of completely different hardware and operating systems. ObjectStore allows for completely heterogeneous interoperation.

Client/Server mode also supports multiple concurrent clients. ObjectStore supports strict two phase locking with guaranteed correct isolation and serializable transaction semantics. Each client has a client side cache. The cache makes it very efficient for clients to continuously use and reuse persistent data, while at the same time automatically maintaining the coherency of the cache as change continuously occurs to any data that may or may not be cached. There is no special API or action needed from the user to flush or populate the cache, ObjectStore performs all the necessary cache management automatically.

8. How do you handle schema evolution?

First it is important to understand that database schema and application schema have to be in sync at all times. As with any object oriented model the application code will expect a particular implementation at the lowest levels of implemented encapsulation. ObjectStore supplies tooling and automation to detect when an application schema is assuming a different class structure than what is stored in the database. ObjectStore will guarantee application schema is validated with persistent schema during application execution.

At the point it is desirable to evolve the schema used for persistent objects to a new schema, ObjectStore provides an administrative utility and API. The utility is capable of analyzing the a new application schema then effectively carrying out all necessary changes to all effected instances in a given database. It is possible for this utility to handle most common types of evolutions like adding or deleting a data member. It should also be noted that ObjectStore provides an API for Schema Evolution to enable more sophisticated class transformations. For example, the API can be used for an evolution that requires an algorithm to properly set data values.. It should also be noted that adding new class types or even adding classes that inherit from existing persistent classes do not require schema evolution. Evolution is only needed when the actual layout of the object needs to be modified.

For more dynamic applications that are challenged to maintain a static schema, ObjectStore offers dynamic modeling libraries. These libraries provide a Meta Schema that can be used to dynamically create, modify and delete class structures. This can be used by applications that need to construct or inherit from classes, add data members and even relations between classes all during program execution. ObjectStore also provides features to allow dynamic aspects of schema to participate in queries and even be indexed. Users can also intermix these dynamic classes with more traditional static classes in the same database and application.

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?

Once an object is considered persistent, i.e. stored in the database, then it can be brought out of the database by simply accessing it. In the simplest form this is just application code navigating to the object, typically this is done following a language reference. Once an object has been brought into virtual memory from disk storage then it stays in memory until it is understood that it is no longer needed. Typically a transaction boundary can signal the end of using persistent data. Users have the option to keep objects in memory beyond transaction boundaries. They also have the option to release an object before the end of transaction boundary. When ever persistent data is no longer needed then the virtual memory can be reused for additional persistent objects. The guiding principal for the longevity of an object in memory is simply based on whether there are transient references to the object. Once there are no transient references to a persistent object then the in memory persistent object can be released. In Java the garbage collector will dispose of the object, in C++ ObjectStore will simply reuse virtual memory whenever needed and available.

Data Modeling

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

ObjectStore is agnostic toward any modeling technique. ObjectStore has many different customers successfully using many different modeling techniques. UML may be the most common seen in the field.

Integration with relational data

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

Yes. Progress software offers integration via our DataXtend Semantic Integrator (DXSI). ObjectStore is released with support DXSI's Data Distribution feature. DXSI can be used to create and provide the runtime execution model to convert one model to another, in this case DXSI can convert a relational model into an object model. Once the mapping is established, the Data Distribution feature which automates the capture of changes on a relational database, can automatically send and translate those changes into an ObjectStore database. Not only can you automate the handling of the model but you can actually keep the ObjectStore model up to date with continuous modifications from the relational database.

Transactions

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

Transactions in ObjectStore define a window in which persistent data can be accessed in a consistent manner. Application programs simply demarcate the beginning and end of transaction boundaries via an API. If the transaction boundaries are within the same program scope you can also use additional API that will allow automatically handle deadlock and retry, which is typical of any good defensively written program that can be used in a multi-user database.

Here is C++ using transaction macros that handle deadlock:

```
OS_BEGIN_TXN(tx1, o, os_transaction::update)
// Put any desired application code here
OS_END_TXN(tx1)
```

Here is Java using transaction API directly:

```
Transaction tr = Transaction.begin(ObjectStore.UPDATE);
// Put any desired application code here
tr.commit();
```

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

Certainly having no strategy to discard objects from memory will mean you can only handle as much data that can fit into memory. The catch is that you can't discard objects from memory as long as there natural language reference to those objects. It is also not possible to discard objects that have been modified before the changes can be safely recorded to durable storage as part of the committed transaction state. ObjectStore automates the detection of changed data and will automate the process of saving data to durable storage. In the event memory is full and needs to be freed before a transaction completes, then there is API that allows the user to push data from application memory to the transaction backing store, thus making the limited amount of virtual memory available for additional data.

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. ObjectStore automatically detects how an application is currently using persistent data. ObjectStore has a mechanism to make sure any instance of an application sees a consistent view of the database. The ObjectStore server is aware of any potential conflicting lock state among clients and no client will be allowed to access inconsistent or stale data. The premise of ObjectStore locking is that a client is given permission to take a lock but that permission can be taken away if conflict arises and that client is not actually holding a lock on the conflicted data. This protocol can make sure all clients are viewing the latest version of persistent data. A client will not be allowed to view data that they do not have the permission to lock. The protocol allows for minimal client/server communication over locks yet makes sure a client will only be able to access the latest committed state of any data in the database.

ObjectStore automatically detects read and write locks taken on data. It is not necessary to add anything to application code to manage locks. ObjectStore fetches pages of persistent data as the user application accesses/navigates to data on a particular page. Pages that are fetched from server to client with the right granted to the client take a lock. Clients are allowed to escalate to actually taking the locks when they access the data. As clients ask for pages and the potential lock requirements conflict with pages previously given to other clients, ObjectStore will attempt to callback the right to lock the page. If right can be called back, the current client can proceed to utilize the page, if the right can't be called back then the current user remains blocked until the right can be obtained, this will happen when the conflicting transaction is committed or aborted.

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

Essentially a Transaction class is provided that has a begin, commit and abort methods. ObjectStore provides expanded API built on top of the basic transaction API that will assist the user in handling typical database application exceptions that can be fixed by retrying the transaction. Common problems that can be retried are transaction deadlock or a dropped connect to a server where failover or reconnection is possible.

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

Yes. At its core ObjectStore utilizes page locking. The simplest way to allow concurrent modifications is to keep multiple clients from utilizing the same page in a conflicting way during a common moment in time. ObjectStore allows users to have complete control over where to place objects in the database. Applications can allocate objects that are not used together into completely different locations in the database, thus avoiding conflict.

In the event you can't physically separate data or multiple clients are attempting to modify the exact same piece of data, then ObjectStore automatically handles lock conflict among clients. ObjectStore uses a strict two phase locking, it is impossible for more than one client to modify the same piece of data at the same time, all writes to a specific piece of data are guaranteed to be serialized.

ObjectStore allows a mode so readers do not create conflict for writers and writers to do create conflict with the readers. This is called Multi-Version Concurrency Control (MVCC). This allows the reader to see a consistent view of all the data in a database but they will not see any modifications to the database that resulted in a conflict to something they have previously read until the conflict is ended. The reader is allowed to see the latest committed state of the database starting with each transaction boundary, so simply finishing and starting a new transaction will make the readers view current again.

Persistence

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

In C++ you only need to pass header files through our schema generation process. This will build all the necessary information used at runtime to persist data. There is no need to inherit from a persistent type.

In Java ObjectStore requires a persistent type to implement a particular interface. This and the actual implementation of the interface can be added by hand but it is almost always done by a class file postprocessor. The post processor will annotate class files to properly manage any object used in a persistent context.

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.

In the context of C++, ObjectStore does not require any modifications to C++ classes to enable persistence. The only thing required is the proper use of an overloaded operator new, which is automatically provided.

For our Java language support, ObjectStore requires the support of a particular interface used to support persistent state operations. This could be done with API modifications to application code but that is completely unnecessary. The enhancement process is almost without exception performed by using the ObjectStore class postprocessor. This utility simply executes as a standalone utility accepting a list of classes you wish to put into an ObjectStore database. It is also available as an API to enable additional IDE integrations. The postprocessor will augment the associated class files with all the necessary code to manage the object in

a persistent context. The resulting class file after post processing is perfectly valid Java byte code, it will work in a completely transient context as well as persistent. The post processed class files are completely useable in all normal and debugging contexts.

Storage

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

In ObjectStore a database is a single file. ObjectStore allows users to utilize cross database pointers and references. From an execution model perspective an application is unaware whether an objects came from the same or different databases.

Architecture

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

Yes, locking is at a page level so there is no restrictions to how many clients can connect, open and utilize data from a database.

21. For which Operating Systems is your product running?

ObjectStore supports all popular operating systems including Microsoft Windows Sun Solaris, Redhat Linux, HP HP-UX, and IBM AIX. ObjectStore also support both the 32-bit and 64-bit versions of these operating systems.

Application

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

We provide the most value for customers that desire transparent persistence and effectively can leverage our client side cache. Our customers are typically in the domains of finance, utilities, defense, telecommunications, travel, information services, geographical information management and web portals of all kinds.

Performance

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

Primarily we benchmark based on customer applications, both internal and external customers to Progress Software. We have utilized homegrown benchmarks and we have utilized more well know benchmarks like the OO7 benchmark.

"Useful Resources"

ObjectStore Product page: <http://www.progress.com/objectstore/index.ssp>

Cache-Forward Architecture Introduction:

http://www.progress.com/objectstore/publications/cfa_intro/index.ssp

