Persistent Java Objects for AS/400

By Scott N. Gerard, Object Oriented Infrastructure, IBM

Computers are an integral part of modern business. Businesses depend upon their computers to hold their data persistently, securely, and with integrity. And if anything should go wrong, the data store had better do everything it can to recover the data. In short, businesses need for their data to be in object databases.

Object-oriented technologies have progressed from toys to production-quality tools. IBM Rochester has rewritten its underlying operating system to include over 2 million lines of C++ code (Berg, Cline, and Girou: "Lessons Learned from the OS/400 Project", *CACM*, Vol. 38, No. 10, October, 1995). This code is stable and more easily maintained, and it gave a performance boost over the older non-OO code.

However, business programmers view C++ as too difficult to learn. They want a simpler language so they can focus on their business problems and not on computer science. Java is much easier to learn and to use than C++. It is also receiving tremendous attention and interest throughout the Internet community. Because of its advantages over other languages, Java will be around for many years.

The AS/400 will soon have the ability to build persistent Java objects that can be concurrently shared by multiple users. The state of an object is persistent from job to job and across IPLs. But this object is much more than a persistent object mechanism. It is an object-oriented database management system (ODBMS).

The AS/400 object will be

- Persistent
- Shared
- Scaleable
- Secure
- Transactional
- Queryable
- Recoverable

and it is integrated with

- San Francisco
- AS/400

- Java

Pools

Persistent objects are physically grouped together into a pool. A pool is a convenient grouping of objects for administration. For example, users are authorized to an entire pool of objects. It would be far too cumbersome to authorize users to each of the millions of objects that might be in a single pool. The pool is also the smallest unit that can be backed up in auxiliary storage (or in the save file).

A system can have many pools. Dividing objects into multiple pools often makes administrative tasks easier. Users could be given access to some pools but not to others. You can also use different pools to reduce the amount of data that must be backed up on a regular basis: Objects that change frequently can be placed in "read-write" pools that are backed up regularly, while objects that seldom change can be placed in "read-only" pools that are backed up less often.

Pools and Relational Databases

While not perfect, two analogies with relational databases (RDB) may help you form an approximate mental image of pools.

A pool is similar to a relational table. A persistent object is similar to a row in a relation table.

The analogy holds in many ways:

- Multiple users can concurrently share rows in a table; multiple users can concurrently share persistent objects in a pool.
- Users are authorized to tables, not rows; users are authorized to pools, not individual objects.
- Tables are backed up, not rows; pools are backed up, not objects.
- A single transaction can span multiple tables; a single transaction can span multiple pools.
- The data grouped together into one row is often the same data that would be grouped together into one object. A single
- department can have multiple Employees.

So a single Department row would be joined to multiple Employee rows. Similarly, a single Department object would have a collection of multiple Employee objects. But there are important differences between these concepts where the analogy breaks down:

- You must open a table before you can use the rows it contains. You do not need to "open" a pool before you can access the objects it contains. When a persistent object in one pool references a persistent object in a different pool, you can simply use the reference to access the second object.
- Rows are just data; no methods are attached to the rows. Persistent objects are objects with methods. Every object type can have its own, personalized methods attached to it.

- The column structure of all rows in one table must be the same. A pool can contain objects of many different classes.
- A collection of objects must have a common base class, but each object may be a subclass with data not present in the base class. A collection of rows (a table) must have the same column structure. For example, consider an application that has Employees and Department on the East and West Coasts. A RDB design would likely group things together that have the same structure so that Employees from both the East and West Coasts are in the same table. A design for pools would likely group objects functionally so that all objects (Employees and Departments) from the East Coast are in an East Coast pool, and all the objects from the West Coast are in a West Coast pool.
- Rows from different tables must be joined to bring their data together. Objects can directly point to other objects so that a simple pointer de-reference brings their data together. This "pointer navigation" is much faster than a relational join operation.
- Processing rows involves multiple copies of the data (operation on copy). Processing persistent objects involves a single copy (operation in place).

Upcoming Features

Persistence

Objects in pools get their persistence nearly for free from the AS/400's underlying single-level store (SLS). System/38 and AS/400 have been built on this technology for decades. The SLS address space is a huge (64-bit), persistent, virtual-address space. Every address in every process is in this single-address space. Different processes are given different blocks in the address space for private data. But they can also share data by using the data's single address. Building an object-oriented database with a 64-bit virtual-address space is qualitatively different than building one with a 32-bit virtual-address space.

The AS/400--not the user or database programmer--ensures that SLS pages are persistent from job to job and from IPL to IPL. Once a piece of data has been placed into SLS, it is as persistent as any file on a TLS system. A decade of persistence and hundreds of thousands of satisfied users prove that the SLS address space is reliable for "bet your business" applications.

Sharing

Persistent objects in pools can be concurrently accessed by multiple users on the same AS/400. Locks (single-writer, multiple-reader) are used to ensure orderly access to persistent objects. Locks can be acquired on individual objects to minimize contention, or one lock can be acquired for a whole group of objects to reduce locking overhead.

Scalability

AS/400 has a 64-bit virtual address space of 1.8E+19 bytes (18 nonillion bytes). This is enormous when compared to a 32-bit virtual address space of 4.3E+9 bytes (4.3 gigabytes). And AS/400's single-level store is behind the full 64-bit range of addresses. This enormous 64-bit virtual address space is qualitatively different than a 32-bit address space. Many small businesses would not be able to squeeze all their business data into 4+ gigabytes. Some ODBMS products spend numerous CPU cycles mapping large databases into a 32-bit address space. But even the largest businesses will not fill up 18 nonillion bytes. (Interestingly, rounding 1.844E+19 to 1.8E+19 casually [casually?] ignores over 100 million 32-bit address spaces).

A single pool can dynamically grow to about 30 gigabytes in size. And if that isn't large enough, simply add more pools. The only limit is the amount of disk storage.

Pools can even have large names. Pools are named using IFS (Integrated File System), which allows names to be thousands of characters long. Pool names are not limited by the 10-character limit for objects in an old-style AS/400 library. And these are Unicode characters, so they can support characters from all over the world.

Security

Each user must be authorized to access persistent objects in a pool. The authorization mechanism is integrated with AS/400's standard user profile support. There is no need to set up and maintain another set of ODBMS user profiles and access lists. Administrators can grant users authority to update objects in some pools and to read objects in other pools, and they may exclude them from yet other pools.

If desired, pools can audit all user access. Audit information is stored in the AS/400's security journal. The security journal is the integrated source of such information on the AS/400 and makes security audits easier.

Transactions

Pools have a transactional resource manager. All changes to persistent objects are either committed or rolled back--even in the face of system crashes.

The resource manager is based on the industry standard, XA specification from X/Open. It supports a two-phase commit protocol. The resource manager is normally controlled by the San Francisco transaction manager. Additional interfaces allow administrators to make "heuristic decisions" should that be necessary during recovery.

Query

For object queries, pools can automatically maintain sorted indexes of objects based on attribute values in those objects. Combined with the object query engine in San Francisco, you can quickly query persistent objects using a SQL-like syntax.

Recovery

Backups are a necessary precaution for all critical business data. One or more pools can be backed up and re-stored to removable media or to an AS/400 save file. Backup files also allow pools to be copied between systems.

Integration with other technologies

San Francisco integration

San Francisco is a set of business-object frameworks built on Java.

In the first release, pools are targeted specifically for San Francisco. Pools provide a high-performance object database for San Francisco's rich set of business objects. This support is provided transparently to San Francisco users.

AS/400 integration

Pools are integrated with AS/400. You use the standard AS/400 commands to work with objects in IFS, to authorize users to pools, and to back up and re-store objects. The resource manager uses the AS/400 security journal for auditing access. The standard "last-changed" date and "last reference" date are automatically maintained. Indexes are built on the existing, highly-tuned indexes that are built into the machine interface (MI).

Pools are currently supported on AS/400's running release, V4R3, or higher. And the code is shipped as part of the AS/400 operating system. All you need is the AS/400 Development Kit for Java product.

Java integration

In the first release, pools are targeted for San Francisco. In future releases, we are considering opening up pools to all Java programmers.

Persistent objects in pools do not require special Java compilers or other Java tools. There are no new Java byte-codes. However, since it depends on details of the AS/400, the AS/400 Development Kit for Java is required.

Because the lifetimes of persistent objects are different from those of the temporary objects normally handled by Java, there are differences in the programming style. Some of the changes include:

1. A persistent object is created in a pool by using a static create method on the Pool class. Java's new operator can not be used to create persistent objects.
2. Persistent objects are not garbage-collected like temporary objects. Persistent objects are deleted by calling method Pool.destroy.
3. Almost all operations on persistent objects must be executed inside a transaction.

Integration with Other Languages

The first release of pools provides language bindings for Java. It does not directly support other languages.

Java's native method mechanism can be used to access programs written in other languages from inside the Java environment. It is straightforward to write a native method in ILE C or ILE C++ for AS/400.

Use of Persistent Objects

Most businesses have too much invested in their current software to even consider moving everything from their relational databases to pools. For most current software, the rule is: "If it's not broke, don't fix it." But there are cases where persistent objects are the best answer.

When Should I Use Objects?

Objects are more flexible than relational rows. Once a programmer understands object-oriented concepts (don't ignore training costs), they are more productive writing OO applications than traditional applications. For one thing, it is easier to make changes to an OO application, since functions are localized in an object and not spread throughout all the code that touches a record. Each object has all the intelligence it needs buried in its methods. You no longer have only "dead data" in a row; you have "live objects."

Applications that are a good fit for an object implementation are:

- An app that has many different types of data.
- An app that has many or complex relationships between its parts. An engineering database is a classic example. The number of relational joins required to bring all the different pieces of data together would be expensive. By contrast, objects can store pointers to other objects ("pointer navigation"), so the app can find all the related objects it needs quickly without expensive relational joins.
- An app where you want to be able to quickly add new types of data without ripping up the rest of the app. For example, an app that basically "displays" objects sets up a framework for displaying (or playing or viewing) other objects. You add a new object and make sure it knows how to display itself.

When Should I Use an ODBMS?

Objects are productive, but Java provides only temporary objects, not persistent objects. If you want to use the data in a temporary object the next time you run your application, you must take extra steps to read and write the data to some persistent storage medium. These extra steps often require a lot of time to develop and a lot of run-time to execute. A better approach is to build persistent objects.

An ODBMS is the obect-oriented version of a traditional relational database. An ODBMS ensures that objects are persistent. An object can be built in one application and used in another. An ODBMS also enforces locks so programmers focus on writing their function and not on adding code to prevent problems between many concurrently-running users. In addition, it provides a transaction-processing model. If the application can't continue for some reason, it simply calls rollback to remove all the partial changes it's made so far in the transaction..

Let the ODBMS worry about data safety. An ODBMS makes your data safe as each transaction is either completely committed or completely rolled back. And data can be backed up and re-stored to removable media using the normal AS/400 commands. Applications that are a good fit for an ODBMS are those that

- need persistence
- need transactions

- must support concurrent users

When Should I Use Pools?

Pools provide an ODBMS for AS/400.

Pools are easy to learn. Programmers need to know Java--the hotest language on Internet today--plus a couple of classes specific to pools. And it takes just a few methods to make your OO applications transactional.

Performance is another key reason. Applications using persistent objects in pools run fast because they can use pointer navigation (avoiding relational joins) and because SLS does not make extra, intermediate copies of the data. Your application operates on the object "in place." There is only one copy of the object that all users share. You use your objects as objects. There is no need to read and write (or "fluff" and "stuff") your objects to a non-object-oriented persistent storage medium like flat files or a relational database.

San Francisco supports pools as an object-oriented, persistent store because of its performance. Its program model was designed with "operate in place" clearly in mind.

Conclusions

Pools provide an object-oriented database management system. They provide persistent objects that are shared, secure, transactional, recoverable, and queryable.

For more information on the pool technology, contact us at sgerard@ibm.us.com.

Pools come from the AS/400 development community. We're focused on helping businesses, not on writing games.

JavaTM is a trademark of Sun Microsystems, Inc.

Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.

Other companies, products, and service names may be trademarks or service marks of others.

Copyright Trademark

Java Feature Page Java Home

IBM HOME ORDER EMPLOYMENT CONTACT IBM LEGAL