EDWARD BOWES

The Three-Tier Shuffle

When more high-volume transaction-processing capabilities are required than conventional client/server can provide, adding a middle tier to the mix can make all the difference

OVER THE PAST TWO years, one of my clients, the Florida Department of Revenue (DOR), has been on an ongoing technological mission shared by state governments and Fortune 500 firms nationwide: moving from an expensive, proprietary information-solution, which is based on a 20-year-old mainframe approach, to a newer and less expensive one rooted in emerging open technologies.

DOR's primary business is collecting and administrating approximately $17.5 billion a year in tax receipts for the State of Florida. Inherent in that figure is the seriousness and sensitivity of the department's business: Being only 99 percent correct in that process translates into a $175 million mistake. After doing the math, it's clear that to make this move to new technology a successful one, DOR must achieve at least 99.99 percent accuracy in tracking, maintaining, and administering monetary data relating to tax receipts and taxpayers. That's why my client has chosen the most powerful technology and the most sophisticated technological approach—three-tier architecture, sometimes referred to as *n*-tier, multitier, or enhanced client/server—as the nucleus for its information systems in the next century.

For DOR, using a variety of technologies has been the best approach for reengineering legacy systems into a new three-tier paradigm. After formal evaluation, DOR has chosen Unix as the primary server operating system, Tuxedo from BEA Systems as the middleware and transaction processing (TP) monitor, Oracle7 as the primary RDBMS, Microsoft Windows as the client operating system, and JAM/Prolifics from Prolifics (a JYACC company) as the Rapid Application Development (RAD) toolset for the client and server pieces. This configuration runs on top of an Ethernet topology using TCP/IP as the communications protocol; Novell is DOR's network technology of choice. On the hardware side, AlphaServers from Digital Equipment Corp. and IBM-compatible PCs provide the physical infrastructure.

Essentially, DOR already has the basis for a conventional two-tier client/server architecture. However, the intensive demands of its business call for a higher-end, higher-volume transaction-oriented information system, much like the legacy systems on its Unisys 2200 and 4800 mainframes. The extra ingredient added to the mix is the middleware component, Tuxedo. This addition transforms DOR's two-tier architecture into a three-tier architecture. The goal is to match the processing power of the legacy system that has faithfully served DOR for the past 15 to 20 years. With over 3,200 users in central headquarters in Tallahassee and scattered across 40 field offices statewide, recognizing the usefulness of such an architecture was little more than common sense.

GOING FOR THREE

In deciding to become a three-tier shop, my client was able to take advantage of Tuxedo's benefits. The AlphaServer made increased performance and logic centralization possible; and this architecture has meant minimal network traffic across DOR's huge Novell LAN and WAN

Delete (CRUD) set of stored procedures that form the third logical tier. This process transforms the application view of data to the physical enterprise schema.

The CRUD layer of stored procedures perform the actual INSERT, SELECT, UPDATE, and DELETE SQL functions for data access. It also performs data domain validations that the client screen may not have enforced, and/or the database check constraints that may not have been programmed to enforce and check before insertion or update. The CRUD layer also contains field-level checks for security. In structuring this architecture for future use with Tuxedo and the TP monitor, we had to account for the fact that given the multiplexing nature of the middleware, the Oracle database would see only one "user"—the name given to the middleware user account. Therefore, field-level security, or any effective security provided by Oracle (users, roles, and so on), would become void. Thus a security database maintaining users, roles, and field-level permissions/ranges was used by the CRUD layer to validate any of the four CRUD operations before performing the actual data access.

The CRUD layer of stored procedures was another consideration. Keeping in mind that at some point the middleware would be hooked in and only one user would be visible to Oracle, effective row-level locking based on session ID—easily available in conventional two-tier technology—would be obsolete in the three-tier architecture. Therefore, a new scheme for the prevention of dirty writes (basic concurrence algorithm) had to be developed. Because the system is to be high-volume, high-performance, real-time, and transaction-oriented, locking rows of data in the database is not necessary. The simple comparison of an audit field, delineating the last date and time a physical row in a table was updated against a change date captured in the retrieve of that row, is sufficient. If the two dates and times are equal, the operation in the CRUD stored procedure is continued. If they are not equal, the potential for a dirty write exists and an exception is raised, forcing the rollback of the entire transaction.

The application/transformation layer and the CRUD layer of stored procedures, although residing in the Oracle RDBMS and physically on the second tier of this two-tier mode, are logically removed from the physical Oracle tables to which they interface. In theory, three tiers still exist: the client, the application/transformation layer (or service), and the physical Oracle tables. The client screens call on the application/transformation layer, which in turn

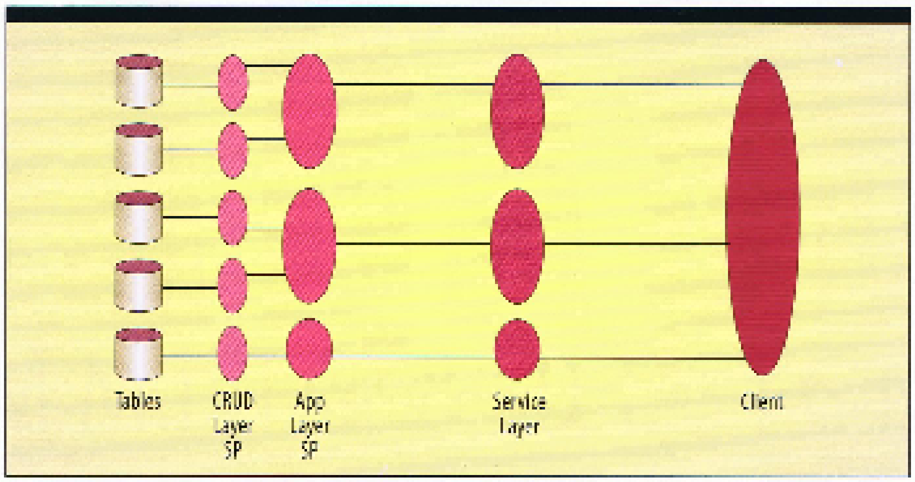

FIGURE 2. Scaling to three-tier.

invokes the CRUD layer.

My client successfully used this model and continued development for about nine more months, exponentially growing the code base. All development was performed using Oracle PL/SQL and JAM/Prolifics. Another function, estate tax, was absorbed into this new system and taken off the mainframe. We had approached the one-year mark and our user community had already grown to 300. It was time to consider hooking in the TP monitor.

THE CURRENT STATE OF AFFAIRS

With a healthy amount of development undertaken, the window of opportunity opens up to plug it all back into the middleware and the TP monitor. The idea in doing so is to create "dumb" or "silly" services that, in terms of their parameter lists, are identical to the application layer of Oracle stored procedures: A client will call a service with the same parameters used in calling the stored procedure. The code for calling the stored procedure will then be put into and executed in the service. This whole scheme centers around creating a simple "middleman," a "dumb" service (see Figure 2, page 76).

The benefits of this strategy are tremendous. SQL*Net will no longer have to be on each client; all access to the Oracle RDBMS will be performed via these dumb services. This factor is an important one because Tuxedo can talk to Oracle, both of which are on the same AlphaServer. Nowhere on the network will there be a need to find SQL*Net packets.

At this point, the true power of the JAM/Prolifics tool shines through. Because Tuxedo services can be written in JAM/Prolifics, the code in the client to call the stored procedures can be cut-and-pasted into the new service container. The client will simply call the service using the built-in support JAM/Prolifics has for Tuxedo.

All the mapping of data to and from the client screens and services is automatically handled by JAM/Prolifics. Calling a service is as simple as calling a stored procedure; it's only a matter of a quick syntax change. As a result, my client now has the power to take all the current work, convert it to three-tier, and preserve all of the application's same behavior, look, and feel. At the same time, the power of asynchronous transaction support, event brokering, load balancing, multiplexing, and so on all comes alive. The cross-platform nature of JAM/Prolifics—its ability to move the interpreted JPL code without changes from the client to the server in particular—made this all possible.

THE FUTURE

Plugging back into Tuxedo gives rise to some even greater possibilities for the future that, in the beginning, were not fully defined—or even seen. For example, in the future model, the application layer will be migrated into the middleware services (see Figure 3); only the tables and the CRUD layer will be located within the RDBMS. However, the client interface to the services will not have to change.

DOR still maintains all the programs and data that have resided on its mainframe for the last 15 years. The tough issues of how to get that data over to Oracle become a day-and-night brain teaser. Full-blown data migration can be so expensive and risky that in many cases it just isn't an option. The alternative—forcing the users to use two sets of applications until the mainframe is phased out—can be inconvenient and is often politically disastrous.

Having the three-tier architecture in place yields some further options. Some because all transactions are reduced to Remote Procedure Calls (RPCs), as opposed to the long, contextual, and conversational modes that occur between a client and a database in two-tier architectures.

In using the middleware, DOR also gained the technological advantages of the TP monitor, which provides event brokering, queuing, function forwarding, and asynchronous transaction support. Furthermore, all transactions can be brought "under the gun" of the TP monitor: If any transaction is "in-flight" for more than a system default time (30 seconds), it's "shotgunned" and rolled back, thus preserving the integrity of the system's response.

The business advantages are easily apparent as well. By choosing Tuxedo, DOR gained the power to minimize its long-term risk. With middleware, the client tier and the database tier become less significant, enabling a degree of independence from the client tier and the relational database—as well as from the database vendors.

A more sensitive, and somewhat unintentional, issue—that of multiplexing—arises out of the three-tier paradigm. Some middleware products, including Tuxedo, are multiplexing technologies. Essentially, multiplexing lets the middleware maintain a series of its own connections to the database. All middleware services (RPCs) that are invoked by clients use the middleware's database sessions for database access. Thus the clients (all users on the system) are connecting to the middleware, not directly to the database. For example, 200 users could be connected to Tuxedo at any given time, but the middleware needs only 10 database connections to manage all data access.

Over the past few years, I've seen database vendors charge up to $2,000 per concurrent connection. When hundreds (and especially thousands) of users are involved, this price can add up to a heavy chunk of change, possibly extending into the millions. Professionally, I feel that some of these pricing schemes are more dangerous to the legitimacy of client/server technology than anything else. However, the multiplexing power that can exist in the middleware can reduce this dollar amount to under 10 percent of the database's software cost. (Some database vendors are now introducing price plans that account for multiplexing.)

Having chosen the hardware and software components of a new three-tiered system, DOR was poised to prototype and pilot the primary systems targeted for implementation. It was time to start developing.

SETTING THE TABLE

After the initial hardware and software setup, six months of prototyping ensued. Simple screens and Tuxedo services were quickly developed in JAM/Prolifics. Oracle PL/SQL stored procedures were also prototyped. At this time, somewhere between five and 15 developers and analysts—most of whom had mainframe skills focusing on Cobol—were formally trained in SQL, JAM/Prolifics, Unix, Tuxedo, and so on.

The JAM/Prolifics tool and its interpreted 4GL, JYACC Procedural Language (JPL), is structural (or imperative), like C or Cobol—and is, therefore, *object-based,* not object-oriented. However, the tool does provide many of the beneficial aspects of object technology, such as the Visual Object Repository (VOR), object inheritance, persistence, state, identity, and consistency. As a result, DOR was able to quickly adapt many of the structural programming strategies it had practiced for years to this new programming. Furthermore, it could now take advantage of a subset of object technology and minimize the impact of programming theory required in this move to new technology. Most important, the quality of programmers and analysts on staff at DOR was well above standard for accomplishing the desired goals.

At the end of this six-month prototyping and education period, a three-month pilot program was launched to handle the processing of physical (handwritten or typed) applications for eight tax obligations relating to the sale, transport, and storage of fuel. This pilot used many of the features and benefits of the three-tier technology, including asynchronous service invocation and reduced network traffic.

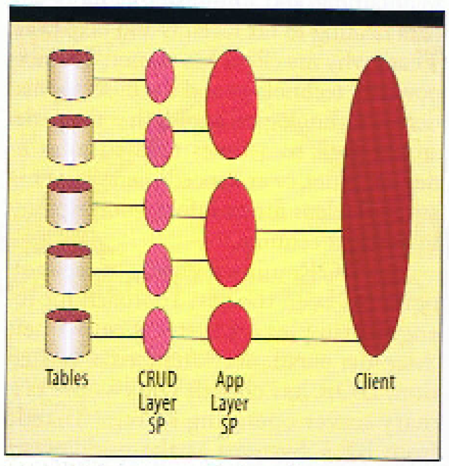

FIGURE 1. The logically three-tier/ physically two-tier model.

The pilot was deemed a technological success; the political realities of the project soon set in, however. My client's executive management and the Florida legislature demanded to see a faster return on investment (ROI) from the new technology. Therefore, we had to develop a new strategy to retain all the benefits of the three-tier paradigm while producing deliverables as quickly as possible.

After taking a step back and analyzing user and system growth in the short term, DOR analysts determined that within 12 to 18 months, the user community would grow to only 300 of the eventual 3,300. In this context, all the power and features of three-tier technology are not necessary; in fact, they could be considered overkill in terms of the cost/benefit ratio. At the time, one analogy likened the project to "hiring a Wells Fargo truck to haul a roll of quarters a mile down the road." The acquired knowledge and established proof-of-concept were taken into account as the basis for the next five years of development, however. Therefore, the project was rearchitected to allow for development on a smaller scale, to preserve the power and advantages of the three-tier paradigm for the future, and to enable a faster ROI.

We decided to remove the TP monitor and middleware for approximately 12 months and continue the building and executing of applications in a physical, two-tier paradigm. However, the design and construction of the software would still be logically achieved in three tiers. That way, in the future DOR could hook back into Tuxedo to take advantage of three-tier technology.

The architecture was broken into three logical tiers (see Figure 1). The first logical tier remained the client presentation layer, which is responsible for data entry, simple data validation, and presentation of the data to the user. The second tier became the application/transformation layer of stored procedures, written in Oracle's PL/SQL and stored in the RDBMS. This layer of stored procedures appeared application-specific because the parameter lists more or less mirrored the data being input and viewed on the client tier (screens). Inside the application/transformation layer, the logical data sets are disassembled and sent (or retrieved) to (or from) a Create/Retrieve/Update/

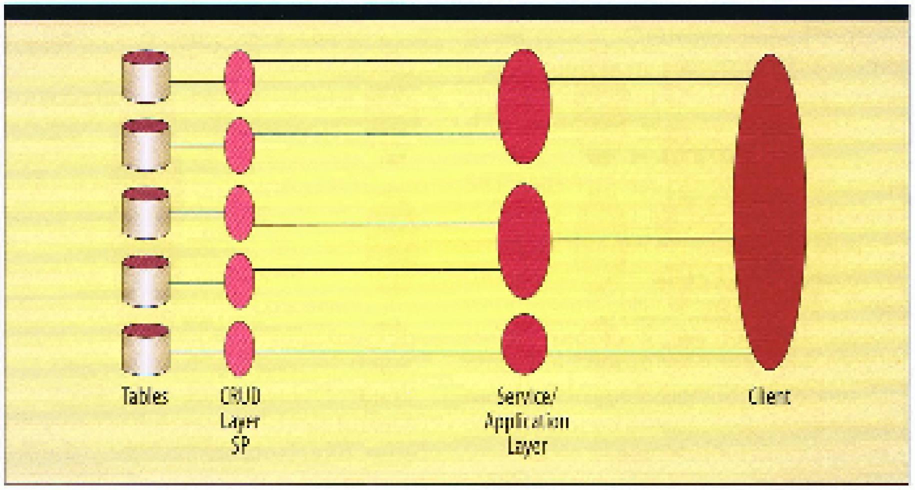

FIGURE 3. Decreasing our dependence on the RDBMS in the future.

new mainframes, such as the Unisys 4800/ ClearPath, are now being designed to include an Intel side that can run either Unix or Windows NT. Furthermore, if the current requirements of the business call for a mainframe upgrade, the technology for linking legacy TP monitors to new Unix TP monitors is available.

As a result, in theory, Tuxedo on the AlphaServer Unix side could talk to Open OLTP on the Unisys ClearPath (Unix/ NT) side. Therefore, the possibility of the two TP monitors communicating with each other enables access to legacy data while keeping the whole foundation transaction-oriented. The three-tier applications residing on the desktop could have access to legacy data, and the user would never know the difference. To the user, it would all be one application and set of screens. Although mostly theoretical, this approach could give the three-tier architecture greater flexibility over the long term.

To discuss the future possibilities and not mention the Internet/intranet would be almost sinful. The Internet/intranet will definitely be part of the evolving architecture for my client. With the inherent design of three-tier architectures, it is theoretically possible to move some Windows-based client interfaces into browser technology and Java applets. The benefit here would be the ability to reuse all the code stored in the services and stored procedures because the Internet/intranet applications could call the existing Tuxedo services. The current toolset being used, JAM/Prolifics, also makes this potential move to the Internet/intranet less painful. Many of the Windows client screens, if carefully constructed, are portable to the Web though the extensions of the toolset. Furthermore, in future releases of the tool, Java applets can be embedded in the Tuxedo services it can create.

Another future approach with potential is that of the Object Management Group's Common Object Request Broker Architecture (CORBA) and Microsoft's Distributed Common Object Model (DCOM). The three-tier architecture is compatible with ORBs as well as ActiveX objects. Again, many Tuxedo services could be wrapped as ORBs and made available as algorithms in common objects. In the three-tier architecture, ORBs as well as ActiveX objects can play a role; the approach doesn't bet the farm on either of the two future standards.

THE WAY TO GO

For DOR, three-tier architecture has been the way to go; so far, its experiences have been edifying as well as beneficial. One important point in the success of such an undertaking has nothing to do with the software or hardware explicitly: The quality of people, as with any successful project, makes the whole three-tier architecture possible. DOR had the fortune of possessing highly skilled mainframe developers and analysts who were willing to accept the move to new technology. Consequently, in my experience, a good mainframe programmer *can* make a good client/server three-tiered developer.

Ed Bowes is a senior consultant with Parabolic Technologies in Winter Park, Florida, and serves as assistant architect in DOR's three-tier architecture project. You can contact him via email at edb@parabolictech.com.

PARABOLIC TECHNOLOGIES INC.

Parabolic Technologies Inc.
www.parabolictech.com
1-800-717-0982

Parabolic Technologies provides systems architecture and consulting in Prolifics Technology.