Jay Almarode

The three-tier architecture and server Smalltalk

In previous columns, I have described the features that make up multi-user Smalltalk. These features include support for transactions, concurrency control and locking, versioning and instance migration, and security. These are but a few of the features required for a Smalltalk system to function as a server. In addition to these features, a server Smalltalk system must also provide persistence of objects, fault tolerance, and scalability. This column is the first of two that describe how multi-user Smalltalk fits in the emerging 3-tier architecture, and how partitioning Smalltalk applications between clients and the server overcomes performance bottlenecks and allows the implementation of shared business objects in a server Smalltalk environment.

There are three kinds of objects that exist in a typical application: presentation, application, and business objects. Presentation objects are the widgets, forms, and windows that present information to the end user. Application objects are the objects responsible for the sequencing of tasks and the management of how business objects are used by the end user to achieve a specific task. Business objects are general-purpose objects that model the processes and basic concepts of the business. Business objects are a hot topic these days, as companies undergo business process reengineering to better model the basic functions of the company. (See Rymer[1] for a detailed discussion of business objects.) There is even a Business Object Management Special Interest Group that was founded by the OMG.

When Smalltalk is used as the implementation language on the client machine only, the objects that implement the application and business logic, as well as the objects that implement the presentation logic, must all reside in the Smalltalk image on the client machine. Typically, when the application starts up, it connects to either a relational or object database and transfers the object state needed to run the application to the client machine. If the database is relational, the tabular data in the database must be mapped to objects in the image. The application may take advantage of the query capability of the database to selectively choose which objects are manifested in the client. However, to execute any business logic, the objects must be present in the client Smalltalk image. This is how Smalltalk is used in a pure client/server architecture.

The pure client/server architecture works well on a small scale, but has a number of drawbacks that hinder its ability to implement enterprise-wide, shared business objects. In this architecture, the server does not have the ability to execute complex business logic. The database may provide some query capability or stored procedures, but does not provide an object model or a computationally complete language like Smalltalk. Consequently, to execute any complex business or application logic, much data must be transferred to the client Smalltalk to be turned into objects that can execute behavior. As the number of client workstations increases, the network becomes overloaded. As applications execute more complex business logic, requiring more objects to be transferred to the client, the client machines need more memory and processing power. Increasing network bandwidth and CPU/memory capability for thousands of client workstations can become an expensive proposition.

There are business drawbacks to the pure client/server architecture as well. Transferring sensitive data to the client machine to execute application behavior can pose a security risk. Client machines are inherently insecure, and none of the client Smalltalk systems currently implement security features in the virtual machine. When the business logic is duplicated across thousands of clients, maintenance is expensive, and this discourages frequent updates to the application. The logic and algorithms implemented within business objects are a strategic asset to the company and should be a shared resource under centralized control. For example, consider the algorithm by which a portfolio management application distributes a customer's funds among stocks with different risks. The better that the application can evaluate market conditions and risk, the better the rate of return. The application's risk assessment algorithm needs to be shared by multiple clients, may need to be updated frequently based upon new strategies, and needs to be protected by theft from competitors.

Using Smalltalk since 1986, Jay Almarode has built CASE tools, interfaces to relational databases, multi-user classes, and query subsystems. He is currently a Senior Software Engineer at GemStone Systems Inc., and can be reached at almarode@slc.com.

To overcome the drawbacks of the client/server architecture, the 3-tier model has emerged. The 3-tier architecture is an evolution of the client/server architecture that defines a middle tier, called an application server, between multiple client workstations and the data server (a relational or legacy database). For a more complete description of the 3-tier architecture, see Lozinski.[2] The middle tier is where shared business objects are implemented in multi-user Smalltalk. This architecture reduces the amount of data transmitted to the client, because business logic can be executed in Smalltalk on the middle tier rather than transmitted to the client for execution. With this architecture, only a single, high-bandwidth connection from each legacy relational database to the application server is needed, and the relational-to-object mapping is performed in one place, and only when needed. The Smalltalk objects on the middle tier can be thought of as an objectified view of the legacy data, ready for each client when needed. This allows for easier integration of legacy data, live data feeds, or other external data sources into Smalltalk applications because each client only sees objects in the application server. Clients do not have to know where the server objects came from, and are insulated from changes to the source of these objects.

Trying to build a server using a client Smalltalk system will not provide the performance or functionality required for large-scale applications.

Having the shared Smalltalk business objects in the middle tier also provides a central point of control for updating business logic, defining security policy, and providing fault tolerance of important objects. In the previous example, the risk assessment algorithm of the portfolio management application should be implemented as functionality provided by a business object. This business object is available to clients through a message interface, but the implementation of the business object is located on the server. The methods that implement the algorithm can be updated in one place, and new implementations can be made available to clients immediately (after adequate testing, of course). The server Smalltalk objects define security policy so that only certain clients, say, those who have paid to use this service, are allowed to execute the risk assessment methods. Finally, the server Smalltalk provides a central repository of objects that can be backed up to tape for archival purposes or recover from hardware failure.

Because of the different roles played by client applications and the application server in the 3-tier model, the requirements for server Smalltalk are quite different than those of client Smalltalk systems. Client Smalltalks operate in a single-user environment. They provide an extensive GUI and graphics class library, and are integrated with the windowing environment of the client workstation. The virtual machines of client Smalltalks are tuned for virtual memory access. A Smalltalk on the server, on the other hand, operates in a multi-user environment. It must provide a model of transactions and concurrency control, and provide a class library designed with multi-user access in mind. The virtual machine of server Smalltalk is tuned for disk access, and must be able to handle very large objects and a very large number of objects. This Smalltalk is tailored to operation on server-class machines to take advantage of shared memory, asynchronous IO, and raw partitions on disk. Server Smalltalk is built with transaction throughput and client communication as chief considerations. Trying to build a server using a client Smalltalk system will not provide the performance or functionality required for large-scale applications.

By using multi-user Smalltalk as the application server in a 3-tier architecture, developers can implement shared business objects with the same language used to build client applications. This enables developers to move behavior easily from the client to the server, an activity called *application partitioning*. See Wadhwa[3] for a description of application partitioning issues. With a common object model on both the client and the application server, objects do not need to be transformed from one form to another. Because the same code can execute in either the client or the application server, the developer can initially build the application entirely on the client, then move portions of it to the server as needed. This might be done to share objects, optimize performance, enforce security, or backup important data. In addition, these partitioning decisions are easily changed as new hardware or software is added to the system. Having a common object model and language between the client and the server makes the repartitioning of an application much simpler, since there is little if any code to rewrite.

When partitioning an application, how does a developer determine where certain objects should reside, i.e., in the client or in the server? Here are some general rules-of-thumb to help in this process. The following kinds of objects belong on the server: business objects, sensitive objects requiring security, large collections of objects requiring optimized query capability, objects requiring shared access, objects requiring fault tolerance, and "gateway" objects (i.e., objects that provide a view of raw data on the data server). The following kinds of objects belong on the client: window or GUI objects, application-specific objects, and "view" objects (i.e., objects that provide a view of a server object). My next column will discuss the implementation techniques for application partitioning.

References

1. Rymer, J.R. Business objects, DISTRIBUTED COMPUTING MONITOR, 10(1), 1995.
2. Lozinski, C. THREE-TIER CLIENT-SERVER ARCHITECTURES, Berkeley Productivity Group, 1995.
3. Wadhwa, V. Partitioning apps: What are the issues? UNIX REVIEW, May 1995.