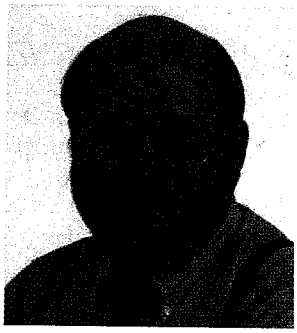
SOM 3.0: Lost in the Garden



BY ROGER SESSIONS

Monday, I got lost in a garden. I was so sure I knew exactly which path to follow that I didn't bother to look carefully at the trail markings. When I should have been in my hotel room, I was under a waterfall.

I stood under an outgrowth of rock while the water formed a perfect curtain before me. I was surrounded by plants that seemed vaguely familiar and smells that seemed connected to an earlier time. Finding my hotel room now didn't seem so important.

The Opryland Hotel in Nashville, Tennessee, is a series of buildings strung together like giant pearls on a string. To get from the main hotel to the conference rooms one must pass through the Conservatory, a very large, elaborate greenhouse. Actually, "greenhouse" fails to convey the scope of the Conservatory. The complex is almost a self-contained ecosystem with trails, brooks, flowers, trees, and the waterfall at which I had found myself.

I gave several talks at the IBM Technical Interchange about SOM (System Object Model) and distributed objects, and I made this nature trek many more times. Each time, I tried to retrace my steps back to the waterfall, but I was never again able to find this peaceful little spot. Perhaps it had been nothing more than my imagination. Several times during the week, I wondered if this interlude was symbolic of the IBM SOM strategy.

SOM 3.0

Over the last several months, I have been feeling much better about SOM. In the past, I've criticized the SOM marketing strategy. Since then, the SOM 3.0 beta has been made widely available on the IBM SOM-Objects Web site. All of the major hardware groups within IBM have announced plans to support SOM. IBM has set up a focused SOM marketing group under the leadership of Anthony Brown.

I've met Anthony Brown several times and have been impressed. The SOM brand manager assisting Mr. Brown is Chris von Schweinitz. Chris has been a strong advocate for porting SOM to other hardware platforms and is an effective spokesperson for SOM. Also assisting Mr. Brown is Rick Clark, manager of Object Marketing and Support. All of these people are very capable.

I called John Slitz, the IBM vice president of marketing for object-oriented (OO) technology and Mr. Brown's boss, just to make sure this group's future is secure. John assured me that having a marketing group completely focused on SOM is critical to the whole OO technology strategy, and that this group has his full support.

So for the first time, SOM's future seems to be in the hands of a group that understands its potential. Rather than presenting an inconsistent "SOM solves all the world's problems" story, we actually may have a group that is focused on the *raison d'etre* of this product: distributed objects.

SOM as a CORBA implementation

SOM 3.0 is first and foremost the industry's most complete implementation of the OMG CORBA specification. The OMG (Object Management Group) is a consortium of hardware and software companies based in Framingham, Massachusetts. Its charter is to develop architectural standards for Distributed Object Computing (DOC).

I believe DOC is an exciting field, and over the next five years will become the technology of choice for doing distributed computing applications. SOM is one of the industry's leading DOC technologies. With SOM, IBM is well positioned to lead in this whole technology area.

The basic SOM architecture implements the OMG-defined CORBA (Common Object Request Broker Architecture). This architecture defines mechanisms for invoking methods on remote objects. I gave a general introduction to this architecture as implemented in SOM 2.1 in the April 1996 issue.

There are at least three reasons you might use a CORBA architecture. The first is that you want to use objects, but want them in a different process. That way, if they crash, they don't take you down with them. The second is that you want to invoke methods on an object, but the methods can be implemented more efficiently on another machine because, say, the methods need to access data on that machine. The third reason is that you want to share objects with other processes.

This third reason for using CORBA, sharing objects, is the most important from the perspective of DOC. When sharing objects, two or more processes make method calls on the same remote object. As one process updates the state of the re-

mote object (through remote method calls), those state changes are visible to other processes (also through remote method calls). If, for example, one process orders two tons of bigDog food from an inventory object and another process checks the amount of bigDog food on hand in that same inventory object, the second process will see the inventory reduction resulting from the first process's order.

SOM 3.0 has a number of important enhancements over and above SOM 2.1, the current version. First of all, it is the first version of SOM to support the CORBA IIOP (Internet Inter-ORB Protocol). This protocol specifies how different implementations of the CORBA architecture interoperate with each other. With IIOP support, SOM clients can invoke methods on objects living in non-IBM CORBA implementations, and vice versa.

Object services

The second set of SOM 3.0 enhancements has to do with IBM's implementations of the so-called CORBA Services. This group of OMG-defined frameworks is intended to solve common programming problems in distributed applications. Let's take a look at some of these problems and the solutions CORBA offers.

The first problem is instantiation. Instantiating objects in a distributed-object system is a bit more complicated than instantiating objects in a simple single-process system. Traditional C++ programmers simply call for a new object of a given type to be created. Distributed programmers need to specify not only the type of the object, but also its location.

The process of instantiating objects is defined by the CORBA Life Cycle Service. In this model, objects are instantiated by special purpose objects called factories. A given factory object knows how to instantiate a given type of object in a particular process.

To find the appropriate factory object, we use a factory finder object. A factory finder takes a description of what you want instantiated and where you want it instantiated, and finds a factory capable of doing that instantiation.

The Life Cycle Service is actually much less important than you might think. The reason is that in distributed applications most processes do not instantiate objects. Most use existing objects. Consider our two processes using an inventory object to order and track bigDog food. Neither process creates the inventory object. One uses the inventory object to place an order. The other uses it to check inventory levels. If the second process instantiated a new inventory object, that object would have a new state that would not include the updates made by the first process. This state discrepancy, obviously, would defeat the whole purpose of the distributed application.

Millions of dogs all over the world depend on transactional systems for their health and well-being, as do bank customers (who are obviously much less important).

Web pages offer a useful analogy to a distributed object application. We do not—usually—create Web pages. Most of the time we are using existing Web pages, then moving on and using some new Web pages. The pages were there before we came, and they will be there for others to use after we are finished.

The next CORBA problem has to do with finding objects. If our two inventory processes want to invoke methods on this existing inventory object, they must somehow reach out into the CORBA ether and find the object they want to use. The mechanism CORBA defines to find objects is the Naming Service.

The Naming Service can be thought of as a well-known black box. Everybody has a reference to the same black box. Anybody can insert an object into the black box and assign a name to that object. Anybody can ask the black box to find the object associated with a particular name, reducing the problem of finding shared, distributed objects to a much simpler problem of agreeing on names. As long as I know the name of the inventory object, I can ask the Naming Service black box to find the inventory object.

Most distributed applications fall into one of three categories. *Initialization* programs are those that instantiate objects, assign names to them, and place them into the Naming Service. *Application* programs find and use these objects. Our inventory processes are such programs. The vast majority of programs will fall into this category. *Clean-up* programs do an ordered shutdown of a system. They remove the objects from the Name Service and de-instantiate the objects, freeing up whatever resources the objects are using. The initialization and clean-up programs are run by system administrators. The rest of us will be running application programs.

The next CORBA problem has to do with discovering that something important has happened. Suppose, for example, we want to have two distributed objects working together. One is an inventory object that keeps track of stock levels of various items. The other is a supply object that knows how to bring in new supplies when stocks of a particular item run low. How does the supply object know when the inventory object is running low on some item?

The CORBA solution uses the Event Service. Objects have a defined mechanism for defining, raising, and noticing events. An object might, for example, define a low-bigDog-food event. The inventory

item will automatically raise this event. A supply object could be looking out for this event, and, when it "notices" the event has occurred, jumps into action.

The Event Service divides objects interested in a given event into consumers and producers. A consumer is an object that wants to know the event occurred. A producer is an object that causes the event to occur. In our example, the supply object is the consumer, and the inventory object is the producer of the low-bigDog-food event.

The Event Service also supports two different programming models for events. In the pull model, the consumer continuously polls the producer to find out if the event occurred. In the push model, the producer tells the consumer the event has occurred.

It is possible, but unusual, for event consumers and event producers to communicate directly with each other. More often, they communicate through event channels. An event channel becomes an intermediary between consumers and producers. Event channels allow any number of objects to raise an event and any number of objects to notice the event. In our inventory application, we could have many bigDog-food suppliers watching for the low-bigDog-food event who start a bidding war as soon as the stock supply has gone down. They would do this by registering their interest in the low-bigDog-food event with the specific event channel coordinating the producers and consumers of low-big-Dog-food events.

The Naming Service is a fundamental service that ties into both the Life Cycle Service and the Event Service. The Life Cycle Service builds its Factory Finder on top of the Naming Service. The Event Service uses the Naming Service to make its event channels known to the world. Anybody who would like to know about low-bigDog-food events would use the Naming Service to find, for example, the object named **OutOfBigDogFood_Event_Channel**. Once one has a reference to the coordinating event channel object, one can register as either a consumer or a supplier of the relevant event.

Late-Breaking News from the Object Front

IBM has recently announced an "agreement to collaborate" with IONA Technologies Ltd. IONA produces Orbix, one of the most successful products to compete with DSOM.

According to the press release "the announced collaboration with IONA provides expanded platform support for IBM's SOM technology and increases our customers' choices in platform deployment."

I have been asking all my IBM friends what this "collaboration" means technically, and nobody seems to know. So let me offer my own speculation. SOM offers a rich object model, a full implementation of many CORBA object services, and a relatively nonportable ORB (DSOM). Orbix offers a minimal object model, almost no CORBA object services, but a highly portable and successful ORB. The most logical merger between these two products is to replace DSOM by Orbix, keeping the SOM object model and object services and the Orbix ORB.

Should IBM follow this path, a significant delay of SOM 3.0 on all the IBM platforms would likely occur, assuming SOM 3.0 is where the merger will take place. Customers might also become wary of starting to prototype with the currently available beta SOM 3.0, given the widespread changes one can expect as these two products become one.

One of the biggest technical problems IBM will have to face in merging SOM and Orbix will be the SOM "thunk" technology. This technology was introduced in a somewhat misguided effort to support the Direct to SOM C++ compiler technology, and is the least portable part of the entire SOM kernel. My guess is it will have to go, which may put the entire Direct to SOM C++ strategy at risk. Personally, I don't feel this is a major loss, but others may not agree.

Although any merger between SOM and Orbix will cause a lot of short-term pain, the long-term gains could be significant. The merger could add some desperately needed platform coverage to SOM and help focus SOM on object distribution. It could improve Orbix's object model and object services story. Overall, I think this announcement may be good news, or at least could be once we figure out what the news really is.

The Persistence Service solves the basic problem of having objects coordinate their state changes with data in a database. Our inventory object would almost certainly need to update a corporate database to reflect changing supply levels.

Distributed objects do not necessarily need a Persistence Service to store data. An object can decide on its own either to refresh its data from a database or to update a database. But where applications need a standard interface to store data, the Persistence Service comes in very handy.

The Persistence Service is similar in concept to an object-oriented database. Both are able to store and restore object state. The Persistence Service differs in that it supports a wide range of underlying database products and data formats.

Object-oriented databases are generally difficult or impossible to integrate with standard corporate databases and existing data formats. They offer little or no support for integration with existing applications. For these reasons, object-oriented databases have had little impact on corporate applications development, and, in fact, may have actually inhibited the adoption of object technology in this area. The Persistence Service, on the other hand, is specifically designed to work well with existing databases, data formats, and applications.

The Persistence Service needs some mechanism for moving data in and out of objects. This functionality is provided by the Externalization Service. The Externalization Service is based on a mechanism similar to C++ streams.

Most serious commercial appli-

cations need to work with objects within the context of traditional transactions. Suppose we have two inventory objects, one that represents a warehouse and one that represents a store. Let's say we have an order management system that moves inventory from the warehouse to the store. If our application fails between the removal from the warehouse and the receipt by the store, we have lost a heap of bigDog food. Not a happy state of affairs.

The Transaction Service ensures that either the bigDog food both leaves the warehouse inventory and enters the store inventory, or neither leaves the warehouse nor enters the store. Millions of dogs all over the world depend on transactional systems for their health and well-being, as do bank customers (who are obviously much less important).

In SOM, the Persistence, Externalization, and Transactional Services are closely related. Persistence uses Externalization to move data in and out of the object and coordinates its database update activity through the Transaction Service. A description of SOM 3.0 and the services related to persistence can be found in a brand new book called *Object Persistence: Beyond Object-Oriented Databases* by yours truly.

The SOM implementation of the Persistence Service ships with support for using the IBM relational database DB/2 as a datastore. This work was a fun collaboration between myself, then of IBM Austin, and Guylaine Cantin, then and now of IBM Toronto. I have tried to entice Guylaine away from IBM to ObjectWatch, but so far, no luck.

Back to the waterfall

The implementation of the CORBA Services is an important milestone for SOM. It means that programmers can, for the first time, write truly CORBA-compliant distributed applications using IBM's products. Not only that, but SOM is the industry's most complete implementation of these CORBA services.

If SOM has the first decent CORBA implementation, and finally has its marketing group working well, why do I feel like I am searching for my lost waterfall everytime I look at the IBM SOM strategy? The answer is simple: VisualAge.

IBM was trying to deal with a very real issue: lack of tools support for SOM. The response to this problem was to reorganize the SOM project under the Toronto tools group. This response shouldn't be too surprising. The standard IBM response to any problem is to reorganize. In fact, the standard IBM response to the new moon is to reorganize.

Placing SOM under Toronto tools definitely gave SOM some serious tools attention. It also put SOM near the VisualAge product line. VisualAge seems to be the class-library equivalent of a black hole, sucking in any framework that has the misfortune to cross its sphere of influence. SOM was no exception.

The current release of VisualAge includes about 500 classes (including SOM 2.1), most of which interoperate poorly, if at all, with SOM. According to John Slitz, the VisualAge scheduled for release next year will include over 2,000 classes, a fourfold increase over the current number. When John proudly made this announcement at the Technical Interchange, I believe a collective shudder ran through those of us who have actually had to deal with real class libraries.

Robert LeBlanc tells me not to worry, everything is under control. Robert is the object-oriented application development technology director. He owns VisualAge, SOM/DSOM, OpenDoc, and all of the SOM object services. First, he says that IBM will continue to make SOM widely available through other channels besides VisualAge, including the Web, at little or no cost. Second, he says that VisualAge is maturing into a well-designed tool for developing client-server applications, and that SOM is an absolutely critical foundation technology. Third, he says not to worry about the 2,000 class libraries. (I think he felt the shudder.) These classes will be organized into highly structured and layered frameworks, and most programmers will deal with only a small number of these classes.

I must admit to a bit of skepticism here. IBM has not done a great job integrating even 500 classes. Now they are quadrupling the number. I'll be hoping for the best, but I'll believe it when I see it. When I asked Mr. LeBlanc what the recommended programming layer will be, he gave me the standard IBM answer: different layers for different players. In other words, the customers will have to sort this out.

So where are we with all this? It looks like SOM is well on the road to recovery. A good CORBA implementation. A good marketing organization. A lot of customer interest. But SOM is only one of a very large number of trails, brooks, trees, and flowers. And perhaps more than one waterfall. Let's hope it doesn't get lost in the garden. OS/2

Roger Sessions is president of ObjectWatch Inc., a company specializing in training and consulting in the use of SOM, DSOM, and related object-oriented technologies. He has spoken at over 30 conferences and has written extensively. His three books include the newly published Object Persistence: Beyond Object-Oriented Databases. *Roger also publishes the SOMobjects Home Page (http://www.fc.net/~roger/owatch.htm) and an Internet newsletter called* ObjectWatch on SOM. *He can be contacted via e-mail at roger@fc.net.*

Resources:

Information on SOM and CORBA on the Web:

The SOMobjects home page, with pointers to many IBM and non-IBM SOM sites — *http://www.fc.net/~roger/owatch.htm*.

The OMG home page — *http://www.omg.org/*.

Kate Keahey's online tutorial to CORBA — *http://grouchy.cs.indiana.edu/hyplan/kksiazek/tuto.htm*.