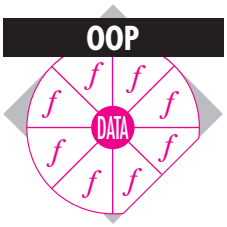


IBM Object Technology

A panel of senior IBM technical people discussed object technology at the POWER Conference in May 1994. Here are some of the questions asked by developers during that session.

Moderator:
Martha Harrington, manager of AIX Solution Provider Technical Support in Austin

Panel Members:
Phil Cannata, technical lead of the team developing the Taligent product on AIX
Chris Nelson, member of the AIX desktop architecture group, involved in OpenDoc
Ahmed Chibib, technical lead in the AIX Solution Provider Technical Support group
Malcolm Zung, member of the IBM Toronto PowerBench and C++ team
Jim Knutson, member of IBM's AIX Taligent development team
Roger Thornton, developer technical consultant for Taligent

Phil Cannata

Computing today is very data centric. Many customers have terabytes of data stored in databases. In the new object world, this data is encapsulated in objects floating around everywhere. How do they make the transition from the data-centric to the object-centric world? At a high level, how must they change the way they manage data?

Cannata: The object-oriented technologies do not address moving corporate data from where it currently resides. The Taligent Application Environment (TalAE) has data access frameworks that allow you to access the data. The problem is creating more data that must be made persistent. Each technology has its own way of doing that: using its own method or putting the data into the existing database.

In the future, TalAE will have properly behaved objects (semantically) that will help to reconcile the differences between databases that have similar properties. Then we can associate the data with particular types of objects and potentially make the data persistent with whatever mechanism is available.

Chibib: Today, we depend on relational databases to manage that data for us. We believe that those vendors' products will move to the object world to hide those mechanisms from users. That is, they will provide the object storage of that data as they themselves move into the object world.

Knutson: Object-oriented programming in itself is very data centric. You are actually dealing with the object, not the procedure, for modifying the data. So the translation from a data-centric environment to an object-oriented one is not as difficult as it might seem. It is not necessary to change the way you think; you are still dealing with data in its base form. Just encapsulate into objects the procedures that you normally use with data. Then it's a matter of getting the data from the databases into object form and vice versa.

Our business has a lot of information in relational databases and flat files. We use many vendor applications and some in-house programs. Although I hear a lot about object-oriented databases, it seems to me that they are sometime in the future. Would it be a reasonable strategy to do nothing and wait a year or so?

Chibib: The database vendors that we deal with (and we deal with all of them)—Oracle®, Sybase®, Ingres®, and so on—have definitely shown an interest in object programming. I can't speak for how soon they will have products, but there's definitely an interest in switching to the object-oriented world. It will not happen overnight. They have millions of lines of code to deal with, and that's not an exaggeration. But they probably will, over time, switch to that paradigm.

Thornton: As system providers, we want to ensure that you have a first-class programming environment (using our objects and frameworks)—the ability to use any one of those types of databases from any of those vendors.

We're probably not the ones to ask whether an Object Database Management System (ODBMS) is better than a relational DBMS because that depends on the application. Each product has its own strengths and weaknesses. Taligent's data access strategy provides a way to get to each of those without a massive difference in Application Programming Interfaces (APIs)—whether you're going after data in a legacy database, or you have data structures that make more sense placed in an object store, or you just want to put ad hoc data in a file. The data access framework design of Taligent 1.0 provides good availability for those data accesses.

Nelson: In the database arena, one of the first areas that you will see is databases integrated into OpenDoc and Taligent. It is reasonable to consider modifying the front end of an existing database so that it integrates well within a Taligent or OpenDoc environment. Think of the front end of the database as the object. You've encapsulated that entire database in this front-end object that now has an object-oriented interface.

How do you embed database objects into a compound document, and what does that mean? What are the scenarios? How would users do that? What's the concept of linking database objects with other database objects? What's the presentation to the user? How do you take advantage of the integration of all these other facilities when working in a cooperative way in the abstract compound environment? The front end is where you will see it. And it doesn't matter whether it's an object database, a relational database, or a flat file. The front end will really take the forefront.

Thornton: Two publications I would recommend that focus on object technology and how it applies to databases are the *Journal of OO Programming* and *Object* magazine. They contain articles about real-world customers building information systems in C++, Smalltalk®, or some other Object-Oriented (OO) language that accesses legacy data in relational databases. Most of the leading database companies are working to provide tools that will make schema mapping from object to relational data structures easier.

Another organization to watch is the Object Management Group (OMG), a consortium to which our companies belong. It is defining methods so that all vendors will use the same kind of semantics for this type of data. There's a lot of information available on this topic.

Chibib: There is a learning curve as you move from a procedure-oriented to an object-oriented methodology. If you wait until everything crystallizes and then begin to learn and design, you will probably be too late. Start now by evaluating how to reengineer your application. And get your staff trained in object-oriented programming and methodology. Then you'll be ahead of everyone else.

Ahmed Chibib

Why would you choose C++ as an object-oriented language over Smalltalk?

Zung: I don't want to start any religious wars, but there are a few good reasons to choose C++. After completing a major programming effort, we chose C++ for two primary reasons. First, we wanted to use our strong background in C programming. Since C++ is a natural step from C, it was a good match. Second, at the time we started the project, we were a little tentative. We didn't want to take the full step into object-oriented programming. So we decided to go halfway and use the encapsulation features of C++, but not actually do object-oriented programming.

Roger Thornton

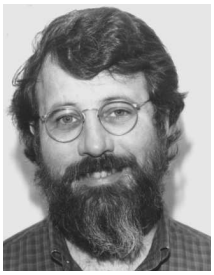
Once we began, we actually got into object-oriented programming very quickly. The point I'm trying to make is that C++ gives you the ability to do procedural programming if you choose (although we didn't in this case). Because Smalltalk is a pure object-oriented language, we would have had to make the full transition to object-oriented programming and design.

Nelson: By programming to an object model such as System Object Model (SOM), your result is really language-independent, at least within

several of the mainstream languages. You could have an object strategy in which the implementation language is C. Other objects could be implemented in C++ or Smalltalk. By placing the SOM interface on all of them, they effectively look like objects that work together.

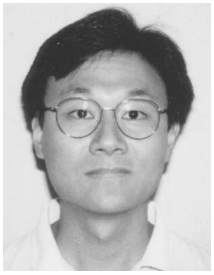
What are the performance aspects of using SOM in an application and then moving the application to a distributed environment?

Nelson: Common Object Request Broker Architecture (CORBA) has a three-level hierarchy of distribution. SOM is a single-process model, so objects are really linked into your address space. The overhead for objects in your address space is very small—only a couple of machine instructions. Transparently, you can move into Distributed SOM (DSOM). In DSOM, objects are in separate processes and in a well-behaved NFS®. This is similar to mapping on another machine—a homogenous kind of network.



Chris Nelson

Obviously you pay a price for communications over the network, but that's generally a known quantity. With a more robust CORBA, the Object Request Broker (ORB) will communicate through a heterogeneous worldwide network using a heavy-weight Remote Procedure Call (RPC) mechanism. You pay a higher price for using a worldwide network, but the one you get depends on where the object is located. If the object is in the local address space, you do not pay for a heavyweight RPC mechanism. If the object is on a worldwide distributed network, then you must use the heavyweight RPC mechanism, which causes some additional performance considerations. You do not want a high-bandwidth transaction with an object that's located far away.



Malcolm Zung

Thornton: The difference in added overhead and scalability is small between existing RPC models and an OO RPC model. The overhead to execute a function on some other machine is high compared to executing the function in the local address space regardless of model. An OO model does not add much overhead. It is easier with OO to conceptualize what is happening when many of these modules are getting called and a lot of data is being moved. With an object model, I don't have to remember all the specific implementation details.

Nelson: The X server has provided us with a lot of experience in client/server relationships. I routinely start an application on a machine that is across the country and have it display on my workstation. The response time and usability is very good. There's no reason to expect the response time to differ just because you are using an ORB. It uses the same basic kind of transaction, which is an intuitive way of saying how good the response time will be.

Can you compare the OO RPC mechanism to the Distributed Computing Environment (DCE) RPC mechanism?

Nelson: We're getting wildly enthusiastic cooperation from other vendors in working out standards for network issues. OMG began with many proposals for an object broker description. The next step was to determine the kind of object services that should be provided. The result of the last round of proposals was one joint proposal that was agreed to by most members of OMG.

We're using standards organizations, such as OMG, to ratify these proposals and make them standard. It's going fast enough—at a reasonably good pace for the industry—so that we can get solutions to our customers in a timely fashion. We don't have to implement products before the standards are adopted.

We're ensuring that the proposal is adopted—that all the bits are set and everyone is happy. We've already announced agreements for various levels of technology sharing with other key UNIX players, including HP™ and Sun®, in order to get that interoperability. Several mechanisms exist—at the standards level and with cross-license agreements—for the companies to ensure that it works.

Everyone is working diligently to come out with an interoperable solution.

Thornton: The Taligent RPC mechanism provides a framework to build distributed applications. The implication is that if the client application has enough flexibility, we're comfortable. We're committed to the various standards, including DCE, CORBA, and others. A framework or object technology can provide a lot of flexibility with the underlying implementation without breaking your source code.

Is OMG using DCE to resolve object references? The issue is that you could partially resolve the object reference to the basic object adapter, which is the server, through the global directory. The object adapter

would know how to get to the proper cell directory. From there, you would get to the object server, which would know about the thousands of objects in that server space.

An application might have thousands of objects. You generally would not need to know all their addresses.

Nelson: OMG produces several services that must be performed on objects, including naming. The functions performed by the object naming service will dictate what the underlying System Object Model (SOM) will be.

There are other services involved with life cycles: creating and eliminating objects, keeping track of object references, notifying or passing messages to objects, and storing objects. Once OMG defines and standardizes these services, the subsystem vendors will implement them in the standard.

Does OpenDoc have security features, such as when an application accesses information from a database?

Nelson: That is currently missing in OpenDoc. It's part of the architecture that has not been filled. Look to OpenDoc Version 2 for that. Component Integration Labs will manage the OpenDoc technology and set the direction for the type of security services and authentication necessary. They are also looking for direction from OMG, which is trying to solve that same problem.

I want to start developing a new palmtop device, but I have no legacy code. There should not be an appreciable degree of overhead involved in using object technologies to implement the code for this device. Are these technologies viable for a scaled-down, low-end device where price is important, or should I continue with traditional development?

Nelson: Yes. They're very appropriate.

Zung: I'd like to comment from the C++ point of view. When we were making our product, performance was an important concern. Everyone told us that you can't get object-oriented code to run as fast as procedural code in C. But we wrote a new version of XLC using C++ that runs faster than the original version, which was written in a procedural language (not C).

Nelson: In a Personal Digital Assistant (PDA) environment, many services must be provided for applications: communications, networking, printing, graphical user interfaces, graphics interfaces, and so on. The technology within Taligent that supports those services is very appropriate for that environment.

Thornton: Another key technology that Taligent obtained from IBM is their microkernel implementation. That was very attractive to us, because to build such a device, you port the smallest kernel you can find, or write your own from scratch.

In designing a microkernel, you put only the essential functions into the kernel: process scheduling, memory management, and very low-level IPC mechanisms. Functions such as file system, networking, and I/O services traditionally found in the kernel remain in a protected process space, but run outside the kernel. You can scale other services for particular hardware applications without recoding the kernel. Traditionally, it has been difficult to change a portion of the kernel without making changes to the whole.

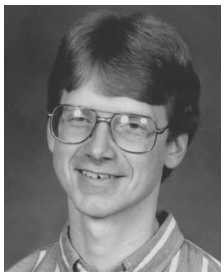
If you wait until everything crystallizes, you will probably be too late.

Most arguments in favor of OO are long-term cost savings for maintenance and future growth. How do you convince customers to take the first step and assume the risk involved when adopting a new technology?

Knutson: It's a tough sell. You cannot sell OO technology on near-term benefits unless you are using the application development environment from Taligent. It does most of the work for you. You only need to customize parts of it to make it fit your application.

That argument works well for a framework environment or for starting from scratch. But if you have a legacy application, you must approach it from a migration standpoint. Divide the existing application into parts so you can migrate the parts that will provide the least amount of trouble, and leave the rest of it alone. Beyond that, I don't know how you make a hard sell like that, because OO technology has a steep learning curve.

Cannata: Actually, there's a motivator in the compound document (as defined by OpenDoc). Within the next few years, users will demand that your application work in a compound document arena. The only way to work in a compound document arena is to make it object oriented. I've been talking to a company in Austin that



Jim Knutson

provides real estate management services. They have many applications and use standard tools including a spreadsheet, word processor, DBMS, and so on. They will not continue using those tools unless they have compound document functions.

Thornton: Many of these technologies are older than I am. It's only in the last 10 to 15 years that they have been talked about a lot—much of it has been theory. We are now beginning to produce technologies such as frameworks that provide functionality which is incrementally customizable without wholesale replacement. We could not have accomplished this without object technology. That drives the arguments that users will want it on their systems and vendors will identify ways to get it there. Hopefully, moving up that learning curve will not be as costly as it would be to not compete.

Learning how to do a good design will be key to your future.

Harrington: Try to convince the customer that there's some application that you can provide for them using OO that you couldn't otherwise provide. When you see some of the Taligent demonstrations, you get excited about new things that you can do.

Can you comment on the different types of analysis and design tools available? Do you have favorites, or have you discarded some in favor of others?

Knutson: The AIX® layers group has been using an enhanced version of the Booch notation. We had an OO design and analysis class earlier this year to bring some of our group up to speed. All the notations have strengths and weaknesses. You just have to fill in the gaps.

Thornton: It's worthwhile to invest the time to read about the principal authors in this field. It really ends up being a personal decision. They all have the same goal, but one might use a notation or style that connects better for you.

Cannata: Our team has been doing design reviews. We've developed some object diagrams to describe what should happen and what objects get instantiated to make that happen inside Taligent.

We developed scenarios for determining which objects will be created, the objects they're associated with, what methods are invoked, and so on. Everyone on the team is convinced that those object diagrams for a particular scenario are the most valuable concepts that we've produced. Not only does it help us understand the system, but we also use the same scenarios in testing; we can test all the objects in the same way. Everything else has fallen by the wayside except the object diagrams.

With these object diagrams, we could probably teach a class about Taligent internals, and maybe teach Taligent! The diagrams are a very good vehicle for teaching others what happens inside the system.

What is the Hollywood principle (don't call us, we'll call you) that I've heard mentioned regarding coding?

Thornton: When developing applications using a framework, the flow of control for that application has already been established and is controlled by the framework. When programming in this style, your code will be called by the framework. An analogy would be if a program consists of several black boxes that are pieced together, your code is inside one of those boxes. Periodically, the framework "opens the lid" and has that code execute. At first this seems strange compared to traditional approaches in which we must architect the application flow as we build it. But this is where we can achieve a great deal of programmer productivity as the programmer now only focuses on adding features to or changing the default behavior or flow.

If I'm a C++ coder and plan to write an application in Taligent, give me a scenario for what I should be doing now, and how long it will take to become really proficient.

Cannata: We have just started writing our first compound document: the calendar. Although you can use many underlying frameworks in Taligent now, I would recommend using the compound document framework. We don't have enough experience yet to say how long it will take to become proficient.

Thornton: Starting about a year and a half ago, we brought in several object-oriented programmers as consultants to write applications on the Taligent Application Environment (TalAE) and to learn about application development. In the past, we have even shut down development for one week while everyone in the company with a development system wrote applications.

From these experiences we determined what takes developers a long time to learn and what is easy to learn. Currently the API to Taligent is C++, although we hope to support other languages in the future. Therefore, the first and foremost suggestion is to learn C++ at the highest skill level possible.

Frameworks are the next thing to learn. We show demonstrations with many features that were written in a surprisingly short time period with very few people. But there is a learning curve to reach that level of productivity. Since you do not yet have the Taligent code, there are some products such as MacApp® and Object Windows Library (OWL) that you can look at or papers such as "Building Object-Oriented Frameworks" available from both IBM and Taligent. This paper also provides a bibliography to other sources of research materials.

Knutson: You can also partition your application into objects. Learn how to do a good design and OO analysis. That will be key to your future. Find out what it takes to partition an application into GUI parts, engine parts, and distributed applications. Learn the key concepts that you need to know about distributed applications and how they work. Taligent will be doing a lot of this, and the concepts and your mastery of them will be key to whether your application will perform well in a Taligent environment.

For legacy applications, is it better to start using OpenDoc since minimal changes are needed to start playing in the OpenDoc environment?

Nelson: Yes, that is a good point. We have applications, technology, and a body of thinking about migrating to the new environment. OpenDoc is intended to minimize what you need to learn and do to enter the compound document arena. That's why we believe that OpenDoc is a good first step for software vendors and end users moving toward Taligent.

You can upgrade your existing applications using your legacy of technology and programmer skills. You can also prepare to move to a richer framework with Taligent. It's not wasted because there is a large learning curve with OO technology, but you'll learn a lot with OpenDoc. Your applications can still interoperate within Taligent, so you will get a revenue stream from those applications.

You can take advantage of some Taligent frameworks that do not necessarily mean compound documents: graphics, printing, and internationalization frameworks, plus some of the other base frameworks. They are tools to help you develop applications by augmenting your existing software.

It's not going to be a piece of cake because there will be some design work about how to fit the applications into the frameworks. But you will go through that kind of thinking anyway. How do you take what you are using now and move it into a new environment? Along the way you will see benefits and be able to meld them together.

Roger, you mentioned that the concept of frameworks was a hot spot on the learning curve. I understand that the way Taligent uses the term "framework" differs from Apple's MacApp, Borland's OWL, and some other products. What technology is available now that we can review to help our people start climbing that curve?

OpenDoc is intended to minimize what you need to learn and do to enter the compound document arena.

Thornton: Both products—MacApp and OWL—would meet our definition of a framework. The primary difference is that TalAE is an entire operating environment consisting of numerous frameworks, while MacApp and OWL focus on one component of an environment—the user interface. A savvy programmer in either of those would have a very big head start on the learning curve.

If you're building an application from scratch, there's not much of a productivity story. But since you built it, you understand it. Every step was incremental. For frameworks, the theory is that someone wrote much of the application in the abstract sense. They put as much as they could into it and then handed it to you. You just modify it to do specifically what you want it to do.

You don't need to understand the entire body of code, but you do need to understand the key abstractions: the location of the interfaces to customize your application, and knowledge of the APIs. You may not need to customize some objects. And that's where the learning curve seems to be so much more than just starting up from scratch. MacApp and OWL are good example frameworks.

Zung: Working in C++ often involves using class libraries, many of which can be considered frameworks. You mentioned the difference between class libraries and frameworks. Class

libraries that actually allow you to define a new type and build on it are similar to Taligent's framework concept. The actual flow control is different, but the concept is similar. You start with a large class library and build the application-specific part of it by inheriting from the library.

Thornton: As a point of interest, many of the original MacApp designers are key architects to parts of the Taligent system, which explains why some similarity exists.

When will Taligent be available?

Knutson: General availability from IBM will be during the first quarter of 1995.

Thornton: Taligent has an early experience program whose purpose is to get the code in the hands of some developers we've been working with—to encourage them to build applications.

Those participating in the early experience program received code the first week of June 1994. This pilot program will allow us to tell our investors which class of applications was a piece of cake, which ones were difficult, the weak parts of the system that need work, and the ones that are strong. This feedback will provide valuable information for the investors (IBM, HP, and Apple) to make their early experience programs a success.

OOPS!

Here's Figure 4 that we inadvertently omitted from the article "Making Backups of Mirrored Filesystems on AIX 3.2" that appeared in the May issue. It is the shell script to reset the logical volume structure back to its original state.

```
#!/bin/ksh
# Restore system to original state.
# Root user authority required.
# Required parameter is the same lv name of previous script
umount /mnt
rmlv /dev/backuplv
#
# Find number of copies and add 1 to it. LVM supports only
# three copies (original + 2) of a logical volume. It should not
# be possible to get an error here if the first phase was done OK.
NUM=~getlvcb -c $1~
if [ NUM -eq 3 ]
then echo "Process error. Too many copies already exist. Aborting."
     exit 3
else
     NUM=~expr $NUM + 1~
fi
#
# Re-create the logical volume copy, using the allocation map file
# created by the first part of the process.
mklvcopy -m allocmap.file $1 $NUM
if [ $? -ne 0 ]
then
  echo "New mirror copy allocation failed. Process terminated."
  exit 2
fi
echo "New mirror copy allocated. Resync process begins."
syncvg -l $1
RC=$?
echo "Resync complete. Logical volume restored to original state."
exit $RC
```