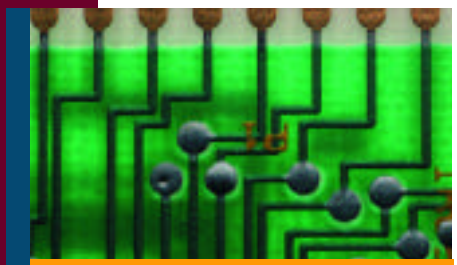


Legacy Systems Enablement: Part I — An Introduction

BY RICHARD B. VIPOND

Companies are rushing to take advantage of the Internet and all the potential it holds for the future of the enterprise. However, these companies also need to embrace the legacy systems that have supported their enterprise for all these years.

THIS series is based on two of the most important and controversial issues facing any enterprise today, namely adapting to the rapid rise of the Internet and the fate of legacy systems. Each article will share my views and experiences on these issues as well as views and experiences provided by other professionals. My experience is based on more than 33 years of developing systems, maintaining operating systems and staying current with emerging technologies.

Companies are rushing to take advantage of the Internet and all the potential it holds for the future of the enterprise. These same companies are already supported by some of the most advanced and reliable systems on the planet, known as legacy systems. The series will examine the use of these systems over the largest interconnected set of networks in the world, the Internet.

My motivation for this series stems from being overwhelmed by the available Internet/legacy solutions. Since being over-whelmed is a common occurrence in our profession, I find that diagrams such as the one shown in Figure 1 enable us to put those pictures into words and usually put the subject into a manageable perspective. The process depicted in Figure 1 will also provide a very worthwhile series of articles from which others can benefit. Part II will provide the details of the research and development environment that will be examined. Part III will further examine the development environment and object repository; and the concluding article will examine integration with the WWW servers.

LEGACY SYSTEM ENABLEMENT AND OBJECT TECHNOLOGY

Legacy system enablement and object technology is not a subject that anyone can be totally knowledgeable about because technology is changing too rapidly for any one person to keep up. Using the input from other practitioners combined with the results of my current research and development efforts, each question and solution presented here will be further discussed and analyzed. I will not be endorsing any products in this series, only technologies and standards that will produce results that will be continually advanced well beyond the Year 2000 (another lesson to be learned from our legacy systems).

KEEPING IT SIMPLE

My first thoughts reflected on how to keep things simple. Well, needless to say, that isn't really an option nowadays! I have found that the next best thing to simple is standards. The following from *IBM OS/390 Java Object-Oriented Technology* sums up my thinking:

"In 1989 the Object Management Group (OMG) was formed with the intention to create standards for object-oriented technology. Since then over 500 companies have joined the OMG. The first OMG publication was the *Object Management Architecture (OMA)*, which was released in 1990. It described the four major components of any object-oriented system: Application Objects, Object Request Broker, Common Facilities and Object Services. The next step was the release of a more detailed specification of the Object Request Broker (ORB), the Common Object Request Broker Architecture (CORBA) in 1991. CORBA provides a basic framework on how objects can send and receive requests.

Object-oriented (OO) technology is an approach to meet today's and future requirements in application development:

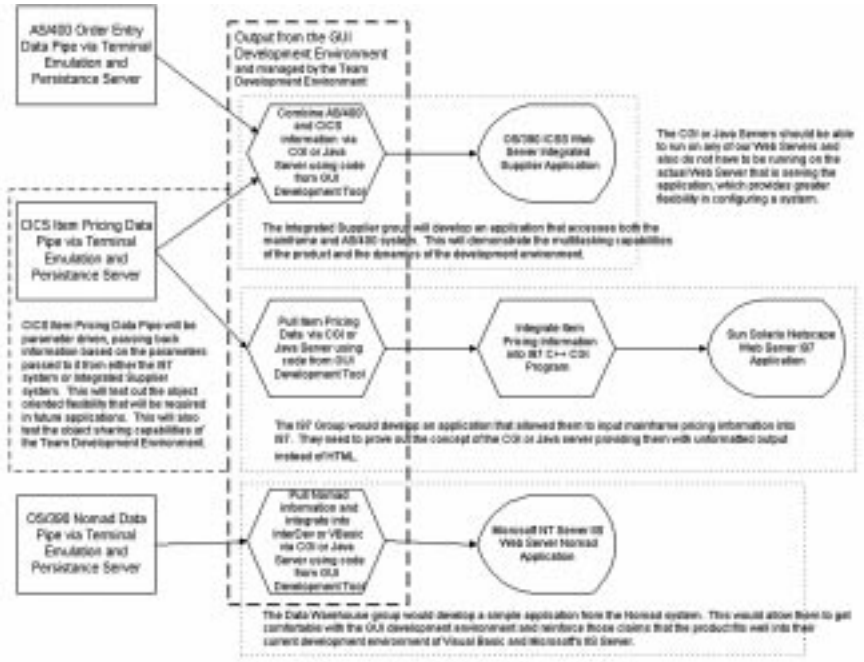
- OO applications consist of interacting software components that reflect subsets of the real world.
- Data and related functions are modeled together and allow a natural design and implementation.
- Simplicity, modularity, versatility and a focus on commonalities drastically improve code reuse.
- Communication of software components through defined interfaces allows them to be implemented in different ways and languages."

It is important to understand that I'm not proposing that implementing object technology be the short-term solution to development requirements. Object technology takes numerous experienced staff members, millions of dollars and more than a couple of years to implement successfully before any tangible results are attained. However, we have to get a grip on the realities of the future and be positioned to begin working on a technology strategy that will be scalable to the future demands of the enterprise. The first step in our journey is to use proven open technology standards in the design of all systems. I will discuss how we get started in Part II.

The most important and valid comments that I've received so far on this topic are from NaSPA member and technical editor, Fred Schuff. I'm sure the majority of readers will share his views and for good reason:

- "My biggest concern is the apparent contradiction between what exists and what is proposed. Systems are poorly designed with few standards, little structure, little planning for future growth or new technology implementation, little documentation, confused programming styles, etc. The new paradigm order will see "standards" adopted and implemented not only throughout a company but across companies, industries and the world economy.
- The real question: Is this massive standards implementation to be done by the same leadership that has created the current environment which is struggling with the Year 2000 changes because there is no order, structure, organization, documentation and coordination within small business units?
- Yes, OO is a great idea. However, it requires even more stringent rules and standards because it is more complex technology and is dependent upon other components. Currently, rules and standards do not work in the IS industry. I can't even transfer a spreadsheet across versions from the same vendor without problems, much less consider transferring $50 million using an OO object designed and implemented somewhere else. If you knew that large banks manually check each large money transfer by hand, multiple times, even though they are electronically transferred, you would understand that OO is farther away than we think. However, the discussion has to begin in order to make some improvements, and this is probably a very good start. Especially if the technology can be explained well."

Figure 1: Example of Concept Requirements



Protecting our investment in application development is a primary reason for using our legacy systems in our Internet initiatives. This protection should continue to be a priority in the development of future systems.

Fred's concerns are real and are being addressed by numerous companies. The introduction of Java to the computing environment has finally given CORBA the vehicle it needs to be implemented in the real world. For a more in-depth discussion of the CORBA and Java advantages, refer to *Client/Server Programming With Java and CORBA*, second edition by Robert Orfali and Dan Harkey. Also, IBM's San Francisco project is a first step toward a global object technology effort and is a sophisticated undertaking that may result in one of the first widely used, powerful business object frameworks. San Francisco is a collection of business process components and services — an application framework — that is designed to reduce both time and expense when developers are building new business applications (also refer to www.ibm.com/Java/Sanfrancisco/). Obstacles are now being overcome on a daily basis, but I feel the biggest obstacle is summed up in Fred's "real question." How do we convince management to invest in long range technology when they have been conditioned by the "I want it now generation"? This series will address these issues and more.

Internet enablement of legacy systems has generated questions that also pertain to current and future application development

in general. It is only proper that a solution for legacy systems be a solution that also encompasses any new development effort. The solution that is provided in this series is a proven methodology, the framework for the future of application development. It will answer the following questions:

- How do we Internet-enable legacy systems to take advantage of newly invented and future technology without creating systems that will soon become obsolete by technology?
- How do we create new systems that contain legacy system logic that can evolve and grow with advancing technology, thus allowing us to capitalize on what has already been done in past systems without reinventing the wheel?
- How do we create systems that allow the use of the best tools for the job vs. a generalized tool that tries to do it all?

All of these questions must be addressed if we are to take advantage of the wonders that lie ahead.

PROTECTING OUR INVESTMENT

Protecting our investment in application development is a primary reason for using our legacy systems in our Internet initiatives. This protection should continue to be a priority in the development of future systems. If a solution to a current objective is imple-mented in "throw away" code, then a primary objective of legacy system enablement has been lost. While the ultimate solution should be to eliminate the overhead associated with legacy system enablement, that solution, too, should be based on the same open architecture principles, thus providing a framework for all future systems.

Since legacy system enablement normally involves taking advantage of existing application logic in the mainframe and midrange environment, legacy system enablement should also take advantage of all existing application logic, not just a derivative portion. This is especially important for those applications that are becoming just as critical to the company's bottom line as the legacy applications. A solution implemented to take advantage of all application logic will greatly enhance the possibility that the same solution will provide the basic requirements for growth and evolution as technology advances.

There currently exist hundreds of solutions for getting legacy systems incorporated into our Internet initiative, and it seems as though new solutions appear daily. Unfortunately, each attempt is to incorporate the final result produced by the legacy system into the new technology being implemented, rather than recovering the completeness of the application's logic. This is normally done with an application development tool that allows for extracting any information necessary from the legacy system after processing is finished.

If we use a technology that is not CORBA compliant, that technology should be used with the understanding that the technology is a short-term fix to be remedied by a CORBA-compliant technology in the near future.

Creating systems can involve many separate technologies and disciplines and normally requires the use of development tools specifically suited to the situations at hand. It may be possible to use one tool to do all development, but locking ourselves into one technology when there are so many to take advantage of just doesn't make good business sense. Using tools that communicate with each other while we take advantage of the knowledge base provided by each tool's technological strengths gives us the best of both worlds. Creating flexibility within the company to allow individuals to use the best tool for the job also makes the ability to find and attract employees much easier.

An important lesson to be learned from our legacy systems is to avoid continually rewriting the same routines and procedures. While source libraries contain numerous reusable routines, they are far from a true object-oriented implementation. Even when success is achieved, copy libraries are only applicable to specific systems or, at best, a single enterprise. Today's object technology advances encompass a global standard, not a company standard.

There usually comes a point when technology, business practices, or simply the passage of time dictate that our systems have become obsolete. Obsolescence is evidenced nowhere more strongly than by our existing legacy systems' problems with the Year 2000 and the struggle to provide them with Internet access. The lack of global standards in the development of these systems is the very reason there are so many legacy system enablement solutions.

A solution to the majority of the aforementioned questions is readily available, it is CORBA. There are other standards in the works but they all follow and incorporate what CORBA establishes. Microsoft's Distributed Com-ponent Object Model (DCOM), Remote Method Invocation (RMI), Hypertext Transfer Protocol/Common Gateway Interface (HTTP/CGI), and Peer-to-Peer communication that hides the details of the network commonly referred to as simply sockets, Extensible Markup Language (XML) and others all have bridges or gateways that do or will communicate with CORBA. I will discuss these standards in more detail in future articles, but in the event that we do want to adopt another standard in the future, at least we will have a standard to convert from if we insist on CORBA as the foundation.

If we use a technology that is not CORBA compliant, that technology should be used with the understanding that the technology is a short-term fix to be remedied by a CORBA-compliant technology in the near future. There may be sound reasons to use a non-CORBA compliant technology to produce quick results, but I see no sound reason to embrace that technology in the long run.

A special thanks to NaSPA member and technical editor, Dwight S. Miller for his help with this article. ts

NaSPA member Richard B. ViPond is a senior consultant for Ciber Network Services, Inc.