Integrating Mainframe and Mid-Range Host Applications with SOA and the .NET Framework

By Peter Havart-Simkin

As Microsoft has rolled out the .NET framework, it has become apparent to many organizations that the Java/.NET debate does not have to end with only one winner or the alternative of an internal architectural debate stalemate. It is now clear that both development frameworks have a role to play in any organization that may have previously backed one or the other as a corporate standard. However, a key attribute of any development framework—if it is to participate in an existing IT environment—is going to be its ability to integrate with what is there already. That generally means hooking into what we unkindly call “legacy systems” so that existing business processes and transactions can be leveraged when new applications are rolled out on new platforms.

A few years ago, general opinions expressed that existing host systems (mainframes, AS400s and the like) were going to be replaced with new applications built on new platforms. That often meant taking, for example, COBOL programs off the mainframe and reinventing them as Java programs on an application server. In so doing, many companies were not really looking at how to get best use from what they had on their host systems because it was going to go away.

Many have discovered, however, that this “rip and replace” approach has severe downsides—like time and budget overruns—and has more complexity than first thought. The whole momentum of this type of IT strategy has stalled for several reasons, not the least of which is that most IT budgets have been cut or restricted.

This whole cauldron of opposing views has now been stirred even further by the specter of a new architectural approach called Services Oriented Architecture (SOA). Since this is an enterprise-level initiative involving fundamental change, it has a major impact on every aspect of a company’s IT strategy. SOA, derived from XML and Web Services, offers the opportunity for a company to move from process-driven services to event-driven services.

SOA involves making processes or transactions available as business services with known interfaces using a standard representation. Potential consumers of these services can find them using a standardized method and then connect to them and use them. Service providers and service consumers can be developed completely independent of each other, which makes SOA different from DCOM or CORBA for those of you who think you have heard this before!

In due course, a full SOA implementation will also be dependent on another fundamental emerging technology called the Enterprise Service Bus (ESB). The ESB (see FIGURE 1) will carry XML-based documents of a known standardized schema (i.e. purchase orders, invoices) and will route them according to content—not by headers on packets. Routing and orchestration will be a fundamental part of an ESB along with other services such as management, guaranteed delivery and security.

It is hard to imagine that an organization can consider implementing an SOA without taking their existing systems with them in the overall architecture. Unlike the rip and replace mentality of a few years ago, an SOA approach insists that you bring your existing host applications, business processes and transactions with you so they cannot be ignored. They have to be part of SOA since SOA is, by definition, an all-encompassing approach. If you cannot afford to, or do not want to, throw your existing systems away and replace them, then you have no choice but to consider how they can be brought into the brave new world of SOA and connected to the Enterprise Service Bus. For this to happen, existing host processes and transactions have to be presented or represented as XML and Web Services. Both IBM and Microsoft will be bringing products to market that will deliver the intelligent messaging infrastructure that will be the backbone of SOA implementations. In Microsoft’s case, this will be the “Indigo” component of the forthcoming “Longhorn” Windows operating system.

SOA: JAVA OR .NET?

So where does this leave the Java/.NET debate? In fact it makes it moot! Both frameworks have their strengths, and both can play a role within an SOA implementation (see FIGURE 2). The key is the Web
Services loosely coupled architecture. It is the lingua franca of both .NET and Java because both can deliver and consume Web Services. There is no reason, therefore, not to consider using the right framework for the right job. The Java J2EE framework, for example, is ideal if you are building high-performance transaction systems (à la BEA Tuxedo) and want to run them on high performance UNIX hardware. The .NET framework will win out if you are building user-facing applications. The .NET framework has the built-in ability to easily handle virtually any user presentations on anything from PCs to PDAs and from SmartPhones to SmartWatches. This is partly because the framework extends right down to, and includes, these sorts of devices.

In a slight twist to the story, the .NET framework has another advantage in an SOA implementation. It is natively XML and Web Services, and is therefore closer to SOA than J2EE is right now. Therefore, .NET is actually ahead in the SOA and ESB race!

**BRINGING TOGETHER .NET AND HOST SYSTEMS**

Because of Web Services, SOA, and the ESB, and the fact that the .NET framework is now a full-fledged player in this space, enterprise integration for .NET has become an extremely important element of the overall picture. A few years ago, enterprise integration for Windows was a second-string issue. If bringing your existing enterprise systems with you is crucial when you move toward implementing SOA, then integrating .NET as an SOA platform with existing host systems is a corporate imperative. This means “wrapping” the existing host application transactions or business processes in such a way that they can participate in an SOA implementation.

“Wrapping” generally means generating XML Web Services that represent, for example, host transactions that can be discovered and subsequently connected to and consumed by a .NET framework application. What is really happening here is that the service represents a business process that is a wrapping of existing business logic. Therefore, it is more accurate to call these XML Web Services “Business Services” instead of simply “Web Services.”

All this, of course, assumes that you are dealing with a loosely coupled world. Web Services is a loosely coupled technology. However, in many cases, you may not want the overhead of the protocol stacks involved with open and loosely coupled systems. If this is true at your site, it is entirely reasonable that certain elements of an SOA can legitimately use tightly coupled technology. In the Java case, this would be JavaBeans or EJBs. In the .NET world, it would be .NET Assemblies, which are DLLs that are part of the new Microsoft .NET Common Language Runtime.

One other notable advantage of the .NET framework over the Java framework is the choice of languages available for application development. At last count, VisualStudio.NET supported 23 different development languages!

**SO WHAT ARE THE OPTIONS FOR LEVERAGING LEGACY SYSTEMS INTO .NET AND SOA?**

This depends entirely on the type and age of your host applications. There are essentially three different levels at which you can access a host application (see Figure 3). The first is at the data level, which usually means direct access to the database on the host. In most cases, this is not the ideal. For the majority of host systems, the interpretation of the data in the database is done by the application sitting on top of it; the data itself does not contain the value you are looking for. It is in the business process above it: the order processing business logic, for example. If the database is accessed directly, then the business logic that already exists on the host has to be reproduced off the host to deliver the same result. Potentially, there could be two systems to maintain. One exception to this would be, for example, if the database contained customer records. Then, the data itself clearly has the intrinsic value you are looking for.

The second level is the business logic level, or the ability to access the business process transactions directly by connecting into the code on the host. This is possible if the host application has either some sort

©2004 Technical Enterprises, Inc. Reproduction of this document without permission is prohibited.
of programming interface (API) that you can access or elements of the application that can be exposed for external access. Older host applications will normally not have any APIs. Even if they do have them, they will probably not be useful for modern-day integration. They were probably added in for very different reasons when the application was originally built.

In the case of more modern IBM mainframe CICS-hosted applications, it is possible to access the transactions in the application through program-to-program communication (using connection protocols like LU6.2 APPC). In other words, a .NET application could talk directly to a CICS application by spoofing another CICS system and operating in a peer-to-peer fashion. The two systems would swap transactions transparently.

In an SOA environment, it makes more sense to expose host transactions as XML Web Services rather than continue with proprietary communication architectures. Accessing host systems at the business logic layer, and subsequently exposing those transactions as Web Services, would be the first choice if that facility were available. Unfortunately, this is only the case in a small percentage (12% at last count according to Microsoft) of host systems applications.

The third level is the presentation layer. Here, a host application is accessed through the same screen mechanism used by terminals and terminal emulation software. In the case of IBM systems, this would be the 3270 data stream for a mainframe or the 5250 data stream for an iSeries or AS400 system. This, of course, connotes “screen scraping,” which is regarded by some as an undesirable, brittle and low-performance way of doing things. That view is out of date and is generally based on experience with older client-side screen integration using such technologies as HLLAPI. Modern, server-based presentation-layer integration technologies are many times better, notably more reliable and considerably more scalable. In addition, they support high-performance applications.

There are two levels at which a host system can be accessed through the presentation layer. The first is after the screen layout has been created, in which case user interactions—screen sequences with inputs and outputs—are captured. Think “virtual user.” The second is before the screen is created. In some IBM applications, it is possible to intercept the transactions as they transit between CICS and the presentation layer via BMS maps. This interception uses a technology called 3270Bridge. In this approach, the screen has not yet been constructed. As a result, input and output fields that would normally end up on a screen and be referenced by screen x: y co-ordinates are seen, instead, as name/value pairs. Naturally, name/value pairs translate very easily into XML tags within a Web Service.

Most older host applications can only be accessed through the presentation layer. They are either too old to have modern APIs or they cannot support products like 3270Bridge.

As we have seen, the way existing applications are accessed is governed by a number of factors. The final two are time and ease. If you want to build access into an existing host system quickly, it will be faster at the presentation layer. If you want to build a non-intrusive solution that requires no changes to the host system applications, then you can choose techniques like 3270Bridge or APPC. There is a high probability that any one host system environment will have a mixture of application types and thus require a mixture of access types to bring those applications into SOA. It is unlikely that one method of access will satisfy all of the needs to expose existing business processes as Business Services in SOA.

To join the Windows world with existing host systems, Microsoft has a product that allows the integration of host applications with .NET. Called Host Integration Server (HIS), this Windows server-based product provides a range of options for bringing host transactions into .NET. Originally delivering COM objects and MTS transactions from business processes on IBM mainframe and midrange systems, the latest incarnation now delivers XML Web Services and .NET Assemblies. While it can access mainframe databases such as DB2, its key capability is to access applications using program-to-program communications like LU6.2 APPC. Consequently, HIS accesses host systems at the data layer or the business logic layer.

The latest Microsoft products are all targeted to integrate with the Microsoft development environment, VisualStudio.NET. It is therefore possible, with HIS, to interact with host applications from within VisualStudio.NET. This makes developing new applications that derive part of their business process from existing applications that much faster and easier.

The .NET framework can easily incorporate business processes or transactions from host systems when new applications or services are being built. However, integration is not simply about building new applications. It is also about bringing additional value to existing applications. Microsoft has recognized this, and it is clear that the new Office 2003 product is as much a development platform as it is a delivery mechanism for word processing and spreadsheets.

Office 2003 incorporates many of the initiatives associated with XML, Web Services and SOA. It also has development tools and SDKs that allow it to be brought closer to the enterprise infrastructure. One of these is called Niobe. It allows Web Services to be called from inside Office 2003 or to be triggered by events from within Office 2003.
WHERE TO GO FROM HERE?

SOA will deliver major benefits to the organization in terms of ease of integration. Once components are SOA and XML Web Services capable, bringing together disparate systems into an integrated system will be much easier. Host systems and existing host applications will be key elements in SOA and will need to be hooked into the ESB.

The Microsoft .NET framework, along with Web Services enabled applications, such as Microsoft Office 2003, will be key delivery mechanisms for the SOA based enterprise. By enabling .NET for XML, Web Services and SOA, Microsoft has renewed their importance to the enterprise as a strategic component alongside existing J2EE initiatives.

In the future, there will be three pillars supporting the SOA organization: existing host-based applications exposed as services, new initiatives built on the Java platform and end user delivery of new services through .NET enabled applications. This should truly end the Java/.NET debate and enable application developers to take full advantage of the new and evolving SOA and Web Services technologies.

NaSPA member Peter Havart-Simkin is the Senior Vice President of Strategic Development at NetManage, Inc. Reach him at peter.simkin@netmanage.com.