



# Jumpstart SOA

*Pragmatic approaches to integrating .NET and Java components within WebSphere Portal*

by Laurence Moroney



**Laurence Moroney** is a senior architect and the director of technology evangelism for Mainsoft Corporation, where he is responsible for counseling customers about their interoperability and porting challenges. Previously, he worked in several fields, designing mixed architectures for financial services systems, airports, casinos, and professional sports.

[ljpm@sportstalkny.com](mailto:ljpm@sportstalkny.com)

**T**he struggle to integrate business assets across the .NET - J2EE technology divide is legendary. So it should come as no surprise that the emergence of portal applications as an enabler of Service Oriented Architecture is forcing enterprises to revisit interoperability challenges in a user-centric environment.

The aim of a portal is to integrate all enterprise data and applications into a coherent whole, and it is wasteful if the portal becomes simply another silo, limited in what it can aggregate due to back end technology constraints. For the portal to be an effective collaboration and productivity enhancement tool, it needs to be populated with enterprise content, irrespective of whether the applications were written in C#, Visual Basic.NET, or Java. This article will survey various technology options that are available for integrating content on the .NET platform into IBM WebSphere Portal Server.

## Portals as Enablers of SOA: Options for Integration

Consider the scenario where a number of business-critical appli-



cations are running on Microsoft's .NET Framework-based technology. The business already has WebSphere Portal running portlets that offer common sign-on and branding. End users want assets that are currently implemented in .NET on the portal, and IT management has the job of figuring out how to make this work.

## Rewrite .NET Apps in Java

One approach is to rewrite .NET applications in Java, which is a good strategy if 1) the intention is to

migrate all development and infrastructure to Java; 2) the applications are small and there aren't very many of them; and 3) ample Java developers are on hand.

If the .NET application is written using tiers, with a typical nTier architecture offering resource, service, business logic, and presentation tiers – a partial rewrite may be an option. One retailer I recently encountered favored this approach, rewriting the presentation tier as Java portlets, and then consuming their .NET middleware using SOAP Web services. Following this approach, the retailer could achieve its end user requirements such as single sign-on using WebSphere Portal's end-user facilities. A partial rewrite provides all the benefits of a pure Java portlet implementation at the cost of disposing some of the existing work and skills. However, the retailer would face the risk of potential interoperability problems between the .NET and Java Stacks across SOAP Web Services. As the original architecture was .NET end-to-end, it is highly likely that used complex data types such as the .NET DataSet, which interoperate

“...it is wasteful if the portal becomes simply another silo, limited in what it can aggregate due to back end technology constraints”

nically on a .NET stack, will fail on a mixed stack, and as such may necessitate considerable re-engineering of the middleware tiers.

### Web Services for Remote Portlets

A second option for a multi-platform scenario is to use a relatively new standard called “Web Services for Remote Portlets” (WSRP). This is a powerful yet limited solution. The concept is simple: Web services typically only communicate data, and not the presentation. With WSRP, the standards-based technology of Web services is expanded to deliver HTML markup in the payload. This allows for applications to run on their native platform as WSRP “Producers.” Portals that want to integrate them to make WSRP calls to them to get their UI as WSRP “Consumers.” It’s possible to make modifications to applications that run on the .NET Framework, and expose them as WSRP “Producers,” which could then be consumed by a Java portlet running on WPS. However, Microsoft has not yet produced a facility for applica-

tions to be exposed as WSRP producers in either its SharePoint product suite or its upcoming one-stop API for integrated applications called the “Windows Communication Foundation.” Nevertheless, WSRP can be used to integrate non JSR-168 applications into portals. While WSRP largely preserves the existing .NET development model and the Windows runtime, it does require the amendment of the presentation layer to make it a WSRP producer.

One of the limitations of WSRP is the brittleness it introduces into the environment. If an application is consumed using WSRP from a portal server that requires a sign-on, a portlet that runs on the portal server needs to be written that can read from its credential store, and those credentials are then used to sign onto the WSRP producer using a WSRP call. It would then have to parse the SOAP message containing the WSRP code, including the UI and generate the UI from that, overriding the existing WSRP functionality on the portal server.

### Re-hosting ASP.NET Applications as JSR-168 Portlets

A third option for .NET-Java integration – available exclusively for WebSphere Portal – is Mainsoft’s cross-platform software, Visual MainWin for J2EE, Portal Edition. This technology recompiles .NET applications to run as JSR-168 portlets that can run natively on WebSphere Portal. The approach delivers a tight integration between .NET and J2EE on the Portal container, providing all the advantages of Java-written portlets, such as single sign on, universal branding, inter-portlet communications, and easy management, without having to reengineer existing .NET components.

Visual MainWin is similar to WSRP in that it preserves existing .NET components and development models. The primary difference is that the runtime is WebSphere rather than Windows. Visual MainWin works by using a patent-pending technology that cross compiles the Microsoft Intermediate Language (MSIL) code generated by the .NET Framework compilers into Java Byte code, and it provides a Java-based implementation of the .NET Framework runtime support classes on which the application will execute. The Portal Edition targets WebSphere Portal, allowing .NET developers to port ASP.NET Web forms applications, ASP.NET Web Services, or .NET class libraries into JSR-168 Portlets, J2EE-based Web services, and J2EE class libraries, respectively. Some reworking is typically necessary to address the paradigm shift between a Web application and a Portlet, but in the majority of cases, this is simply removing hard-coded visual styles and replacing them with ones that the portal server uses to brand portlets running within it.

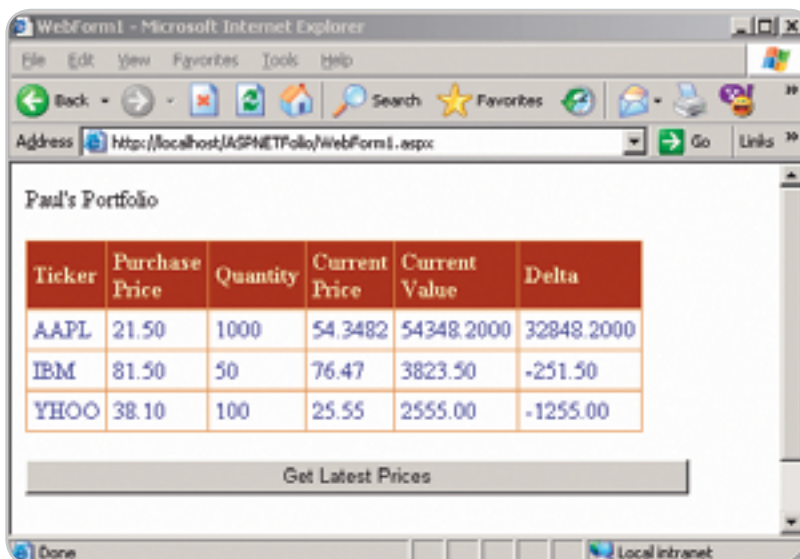


Figure 1 Running the portfolio on IIS

Visual MainWin also includes a plug into the Visual Studio .NET IDE and enables C# and Visual Basic. NET developers to continue using Microsoft's popular IDE to develop and maintain ASP.NET applications running on WebSphere Portal. The cross-platform tool enables code sharing between development teams using .NET and Java technologies, including the facility to directly consume Java-based assets such as WebSphere Portal APIs and EJBs from within the Visual Studio. NET development environment.

Take, for example, a label on an ASP.NET Web forms application that has been developed using Visual Studio, with font and size attributes that are appropriate for its native environment. In a portal environment, the label's attributes are set by the portal container rather than by Visual Studio so the label is consistent with all other applications in the Portal. When re-hosting the ASP.NET Web forms application, .NET developers remove the code used to set the font type and size properties. This does not suffer from the architectural brittleness of the WSRP approach. Whenever the styles used to brand the application change, the portal container sends them to the ASP.NET application.

The ability for Visual Studio developers to retain control over the enterprise applications was a primary reason that a Fortune 100 personal insurer pursued this approach. This insurance provider resulted from two insurance providers that merged. One of the providers was built on IBM technologies and the other was a Microsoft shop. Post merger, the insurer had over 1,000 Visual Studio developers, many of whom have been supporting a homegrown Windows portal comprised of about 700 ASP/VB6 applications. Mainsoft and Prolifics, a premier IBM business partner, demonstrated the ability to replace a Windows-based portal with a WebSphere Application Server and WebSphere Portal. The team required five staff-days to migrate a sample ASP/VB6 application to WebSphere Portal.

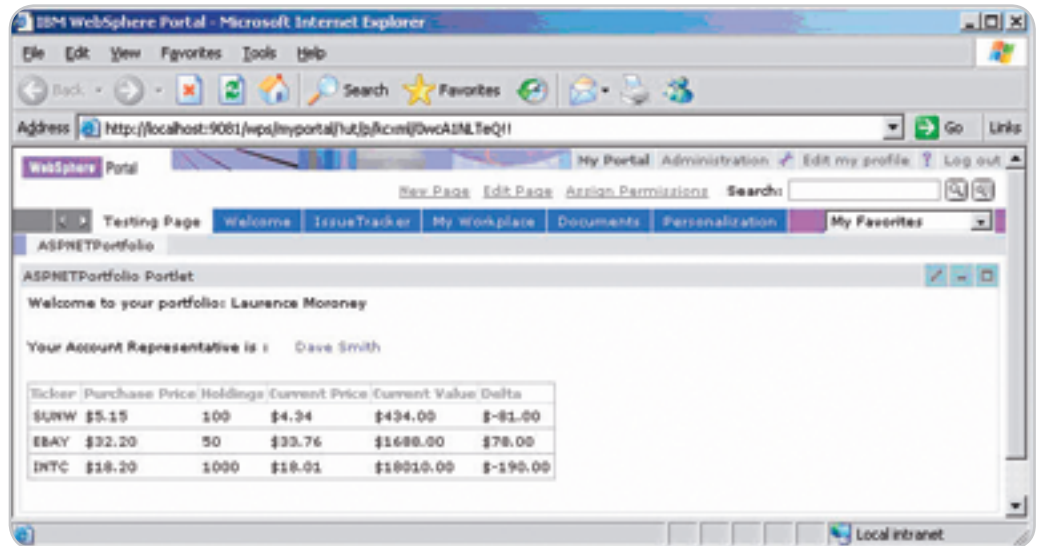


Figure 2 Running the portfolio on WPS

### Sample ASP.NET Portlet Implementation on WebSphere Portal

Take a simple application written as an ASP.NET Web forms application that provides a user interface layer onto middleware implemented using Web services. In this case, the UI layer provides the portfolio holdings. The application is a single Web form that contains a data grid. This data grid uses ADO.NET data binding to an XML data source to present the information. The data source is built from an XML file (in the real world, it would be a middleware service that wraps a database) and a public Web service that provides delayed stock quotes. The overall presentation is defined by hand coding it on the Web forms level. Sign-on is custom written using Windows forms authentication. You can see the application, running on Windows with IIS in Figure 1.

When porting this application to run on WPS, some minor modifications were needed. Many of these are business-driven as opposed to technology shortfalls. For example, the application used its own custom sign-on instead of integrating with the portal sign-on. Thus the code for custom sign-on should be removed, and replaced with code that integrates with the WPS credential store and user management APIs. This is simpler than it sounds, because Visual MainWin enables the

ASP.NET application to consume the Java APIs in the same way any other Java application can handle them. Additionally, the hard-coded visual styling should be removed and replaced with the CSS tags that the WebSphere Portal server defines, so that the application can be branded and styled properly.

Once all this is done, it's simply a matter of recompiling the application into Java code, and deploying it to WebSphere Portal using Visual MainWin. Deployment is automatic to the development server, and a WAR file can be generated for deployment to staging and production servers.

The re-hosted application can be seen in Figure 2. This shows tight integration with WPS, providing a welcome message to the user by retrieving her name from the WPS credential cache as well as demonstrating integration with Java-based assets such as the Workplace client people awareness functionality.

The code for this portlet is available for download here at <http://dev.mainsoft.com/Portals/0/Downloads/ASPNETPortfolio.zip>.

Of course, re-hosting is not limited to existing ASP.NET applications to WPS using this toolkit. Visual MainWin also enables .NET developers to create new applications as Java portlets on WPS that can take advantage of all the underlying functionality for WPS that was previously

	Integration with WPS	Integration with Other Portlets	Manageability	Coding Effort	Preservation of .NET Skills
Full Rewrite in JSR-168	Excellent	Excellent	Excellent	Extensive	Poor
Partial Rewrite in JSR-168	Excellent	Excellent	Good	Moderate	Good
Use WSRP	Poor	Poor	Difficult	Moderate	Excellent
Use Visual MainWin	Excellent	Excellent	Excellent	Minor	Excellent

only available to Java developers (WCT People Awareness, Edit Mode, Portlet-to-Portlet Integration, and “wiring,” to name just a few).

**Conclusion**

Should you have a complex data center where applications run on varied technology stacks, and you want to use the benefits of WebSphere Portal for “on the glass” integration, there are three options available to you:

- Rewrite applications, or their presentation layer, as JSR 168-compliant Java portlets, which can integrate fully into WebSphere Portal. This is ideal in scenarios with limited .NET code in which

demand for integrated end user experience is high. It’s best suited for a transformation to a Java only shop.

- Use WSRP to consume .NET applications in WebSphere Portal, preserving .NET components and developers. However, this approach has limitations in what it can support. WSRP may require an extensive rewrite of applications to get them to work, and functionalities such as single sign on, and consistent branding can introduce architectural brittleness.
- Visual MainWin for J2EE, Portal Edition runs ASP.NET applications natively on WebSphere

Portal, without having to rewrite the application in Java. It also extends WebSphere Portal’s rich end user functionalities across C# and Visual Basic components. This approach is useful when implementing a mixed portal environment quickly; when dynamic business requirements exist for .NET applications; and when retention of the current development model is preferred.

Ultimately, the extent of integration requirements, business requirements, and end user needs will define the best solution to use with WebSphere Portal. ☺



**Mainsoft Corporation**

226 Airport Pkwy • Suite 250  
 San Jose, CA 95110  
 800-MAINWIN  
[www.mainsoft.com](http://www.mainsoft.com)