Web Service Technology for SAP NetWeaver

Web services will likely hold out great promise for transactions that are inherently independent of platforms and programming languages: the ever-elusive combination of interoperability along with greater flexibility and adaptability in building, maintaining, and changing the connection points in your applications. And because Web services are easily executed over the Internet, the resulting business scenarios have an expanded user reach. But how do you go about replacing traditional services — especially the most resource-intensive ones, based on contracts or proprietary protocols — with Web services?

This article outlines how in just a few simple steps — and with the help of tools and wizards built into the design-time environment — developers can quickly and easily adapt current Java and ABAP applications to a service-based approach. But first, let’s look at the benefits of a Web services-based design and SAP’s new level of support for Web services with SAP NetWeaver and SAP Web Application Server 6.40.

Building Web Services for Enterprise-Scale Applications

Protocols for connecting enterprise applications have been around for quite some time, but developers have typically encountered obstacles when designing according to these models. For example:

- Proprietary protocols force developers to think in a product-specific way and learn special programming APIs in order to achieve interoperability. This is the major issue with, for instance, RFC technology. While RFC is a proven, robust technology with an easy-to-use programming model that has been widely used to connect SAP systems, developers have found it more difficult to interoperate SAP and non-SAP environments when relying on software development kits on top of the RFC API.

- Interface description languages leverage programming to express the contract of a business service in a language-neutral way. CORBA/DCOM used that approach, but bridging the two environments turned out to be a major undertaking.

Web services promise to overcome the limitations of these technologies for a number of reasons:

- Web services are defined independently of programming platforms and languages. All major technology vendors support Web services.

- Web service definitions are expressed in XML syntax, making it easy to develop and offer the tools for defining services based on XML.

- Web services, as their name suggests, can easily be executed over the Internet, making it possible to establish distributed business scenarios in an ad hoc fashion.

- Web services can be developed in any programming language, including Java, ABAP, or C#.

- Web services can be published in a common directory based on the UDDI (Universal Description, Discovery, and Integration) standard.

- Web services have been standardized in committees including WS-I and W3C.

Quick adaptation of business processes is crucial for the overall success of a company. Consequently, you need a highly flexible architecture that offers enterprise application content in a service-oriented fashion and that allows you to reconfigure business processes in a model-driven way.

Karl Kessler, SAP AG

Enterprise Services Architecture (ESA) is SAP’s blueprint for helping developers design and implement such service-oriented business solutions.¹ A service-oriented approach, as the name suggests, requires a thorough technical foundation to develop, publish, and consume services — and ideally, will operate in a platform-neutral and standardized way. SAP NetWeaver, SAP Web Application Server (SAP Web AS), and more and more new SAP solutions follow ESA, so they include tools and content that help developers quickly build new business scenarios — especially cross-application scenarios (xApps)² — on top of SAP’s existing enterprise applications.

The advantages of Web services technology are the driving force behind moving to service-oriented enterprise applications and adopting ESA. As a result, Web services will play the dominant role in the implementation layers of various integration scenarios — not just cross-application frameworks, but A2A (application-to-application) and B2B integration as well.

The Web Services Programming Paradigm

With Web services, the world is divided into providers and consumers (see Figure 1):

- The Web service provider develops a Web service in a certain programming language and deploys it to its own server runtime environment. The service is described in the Web Services Description Language (WSDL) using special XML tags. The service description is published in a common service directory (see Figure 1). Web services directories are generally organized following the UDDI specification. Major software vendors such as SAP, Microsoft, and IBM operate their own public UDDI directories. You’ll find SAP’s directory at http://uddi.sap.com.

- A developer on the Web service consumer (client) side can browse the UDDI directory and look for applicable services (see Figure 1). The consumer may then download the WSDL of a selected Web service and trigger the execution of the Web service over the communication link that is established between the consumer and the provider. Web service invocations are standardized using SOAP, while a SOAP request contains the name of the Web service plus its actual parameters. A SOAP response contains the result parameters based on the signature that is exposed in the WSDL. Note that in a Web service scenario, the use of the directory is optional; if you know where a Web service runs and you obtain the description directly from the Web service provider, you can invoke the Web service without using the directory.

Figure 1 The Basic Steps for Creating and Executing Web Services in an Application

Web Service Provider

Figure 2 shows the Web service provider environment in more detail. SAP Web AS 6.40 offers a development and runtime environment for the provision of Web services. The development environment supports the implementation and configuration (security settings, etc.) of Web services, along with WSDL generation and the service’s publication in UDDI.

The runtime environment offers a SOAP engine capable of executing requests and producing responses under a given Web service configuration that controls security settings and the behavior of different features.

Although you can develop Web services from scratch, there are also a couple of good starting points in your current applications — traditional services that can be transformed into Web services. These include:

- Existing RFC-enabled function modules and BAPIs on the Web Application Server’s ABAP stack
- On the Java stack, session beans or ordinary Java classes

¹ See www.sap.com/esa.
² Cross applications (xApps) are composite applications that can span departments and organizations, using solutions from SAP and certified SAP partners. For more information see www.sap.com/solutions/xapps/index.asp.
These services will need some refinement in a specific Web service’s design-time environment before they can be exposed directly to the Web service layer. This environment is now built directly into the ABAP Workbench and the SAP NetWeaver Developer Studio.

**Web Service Consumer**

Figure 3 shows the Web service consumer environment in more detail. From the development environment, the client-side developer is able to locate existing Web services in UDDI directories. The WSDL of a given Web service is then used to generate suitable proxy classes for either the Java or ABAP environment. The Web service proxy configuration is distinct from the actual code, so you can reconfigure the Web service without actually changing the code.

At runtime, the call to the proxy class triggers the SOAP runtime on the client side to issue a SOAP request and translate its response in the format expected by the proxy classes.

As on the server side, the Workbench and the Developer Studio help implement the client-side logic of a Web service scenario.

**A Quick Walk Through the SAP Web Services Tools for Java and ABAP**

The support for Web services in the Developer Studio is a real highlight of the SAP NetWeaver 2004 release. Without any detailed knowledge of WSDL, SOAP, or UDDI, developers — whether they are working with Java or ABAP — can develop a Web service and immediately test drive it. Here’s how:

**Developing a Web Service in Java**

Rather than start from scratch, you can derive your Web services from existing code. For Java, session beans offer a good starting point. Since they are responsible for performing session-level functions and services and serve the UI layer (Web Dynpro, JSP, etc.) on the server side, session beans are the natural entry points to business logic. They expose an interface with business methods that you can use to define the operations of the Web service, and are derived from the underlying session bean types.

To create a Web service from a session bean, SAP provides a Web Services Creation wizard. Simply go to the Developer Studio. There, when you open any J2EE program, you will see it separated into three different projects:

- The Enterprise Application project, which contains items such as the deployable enterprise archive (FlightApp.ear).
- The EJB project, which contains Enterprise Java Beans and their corresponding deployment...
A session bean (FlightBookingProcessorBean) is selected in Figure 4.

- A Web project, which contains things such as JSP, servlets, style sheets, and other Web content that is not relevant from a Web services perspective.

In Figure 4, you’ll see that by right-clicking on the FlightBookingProcessor session bean, you can open a context menu. Start the Web Services Creation Wizard by selecting **New → Web Service**. Based on the properties of the session bean, a popup then appears (see Figure 5). The only thing you have to do is to specify a Web service name (WSFlightBooking in our example).

Next, you select the business methods you would like to expose as Web service operations. There is no need to expose the full interface of an existing session bean; you simply expose the methods that are required for your business scenario. In our example, the Web service will include two business methods, getConnections and save_booking (see Figure 6). For each selected method you can rename parameters for better readability, hide parameters, change their types, or provide default values.

Finally, you build and deploy your enterprise application project to publish the Web service onto the J2EE engine as you would do with any other J2EE project.

That’s it. You’re done. Easy, isn’t it?

All that’s left is to check and test the Web service. You can inspect the WSDL file that was generated automatically for you, however, I personally prefer to test-drive a Web service immediately after deployment. The Developer Studio provides a Web service view that you can open from the **Windows → Show View** menu. Here, the Web Services Navigator lists all services currently available (see Figure 7).
Note!

No matter what language you develop your Web service in (i.e., ABAP or Java), it always has three parts:

- A virtual interface, the selected business methods and potential parameter adjustments
- The Web service definition, which enumerates selected features such as authentication or session settings
- The Web service configuration, which indicates, for example, the correct transport protocol to use

As you can see, Web services are highly modular — one of the benefits that ESA builds on. Changing the configuration of an existing Web service does not involve digging into any code.

If you double-click a displayed service in the Web Service Navigator (bottom right in Figure 7), the home page for that Web service is created automatically. From the home page, you can also select the Test tab, which will display the fields for every input parameter for a particular method — in this case, the getConnections method (see Figure 8). Fill in these fields, hit the Send button, and a page containing the output parameters is shown, similar to the input form.
Web Services Support in the ABAP Workbench

Web services development in the ABAP Workbench — with its own version of the Web Services Creation Wizard — is also quite easy. From the Object Navigator (transaction SE80), select Enterprise Services → Web Services Library and right-click for the drop-down menu (see Figure 9). In this wizard, the individual steps are nicely documented, which will be helpful if you want to make changes to your Web service in the future.

You must first specify a name for the virtual interface and select a given endpoint — an ABAP development object that you can derive a Web service from. In our example, we use function modules (see Figure 10).

After specifying a function module name, the system will generate a suggested name for the underlying Web service definition (see Figure 11).

Then, as shown in Figure 12, pressing Complete releases the Web service, which means that the service can be executed.

To test your Web service, from the SAP GUI OKCode field, start transaction WSADMIN, which provides an overview of the available Web services. From there you can start the Web service's home page, provide input parameters, and execute the Web service just as you would in the Java scenario. The result will look exactly like the view back in Figure 8. Once again you see the power of Web services: you develop a service in one environment (say, ABAP) and easily invoke its functionality in another environment (Java, in our case).

As you can see in this example, a running J2EE engine is necessary to execute the Web service, since the home page functionality was implemented as a Java servlet. However, this function is able to connect to any Web service runtime environment.
Advanced Web Service Features

With SAP Web AS 6.40, a couple of tools and wizards are available to investigate the client side of the Web services programming paradigm: web service client proxy generation and tools for creating and accessing UDDI directories.

Creating Proxies for Web Services

Developer Studio includes a project wizard that lets you create a deployable or standalone proxy for a given Web service (see Figure 13). First, specify a name for the proxy and a Java package name to store the generated proxy classes. Then, select a Web service by specifying its WSDL or URL, or by picking one from the Web Service Navigator. Finally, deploy the proxy to the J2EE engine in an enterprise archive.

Later you can develop a J2EE project and let it reference the Web services client API that was generated in the Web service proxy creation process. The online documentation at http://help.sap.com provides detailed steps for how to work this out.

Setting Up Your Own UDDI Client and Server

Major software vendors currently operate their own public UDDIs, but any SAP Web Application Server can be configured to offer UDDI client and server capabilities. The Visual Administrator in SAP Web AS provides the means to set up the UDDI client and UDDI server:

- The UDDI client in the SAP Web AS is a Web application. You will find its main screen on the SAP Web AS homepage after installation. It lets you browse existing UDDI registries and add new Web service entities.

Conclusion

Web services offer an easy yet powerful way to transform your business software into service-oriented applications, and will form the foundation for semantic interoperability that supports enterprise-scale applications. SAP Web Application Server 6.40 now offers a rich set of tools and wizards to explore the potential of Web services from a development perspective to build new applications on top of SAP NetWeaver’s Web services portfolio. Not only do these tools reflect an Enterprise Services Architecture framework, but they also allow developers to more easily design custom applications built with a standardized Web services-oriented approach.

For details and documentation on building Web services in the SAP NetWeaver Developer Studio, visit the online documentation inside the Developer Studio, or at SAP’s Developer Network (http://sdn.sap.com). For more information on Web services in the ABAP stack, visit http://help.sap.com. And for an overview of ESA, please see www.sap.com/esa.

Karl Kessler joined SAP in 1992. He is the Product Manager of the SAP NetWeaver foundation including SAP Web Application Server, Web Dynpro, ABAP Workbench, and SAP NetWeaver Developer Studio, and is responsible for all rollout activities. You can reach him via email at karl.kessler@sap.com.