Building Web Services the right way using IBM® WebSphere Studio

November, 2001

Setting the record straight

The purpose of this paper is to respond to Microsoft's misleading white paper about creating Web Services with .Net versus IBM WebSphere V4.0 ("WebSphere"). The white paper can be found at http://msdn.microsoft.com/net/compare/webservicecompare.asp. In this white paper, Microsoft claims .Net is a better platform than J2EE for creating Web Services. In their attempt to support this claim, Microsoft hired a supposedly independent consulting firm to develop a Web Service using both .Net, and in Microsoft's own words, the J2EE platform with the most advanced support for XML based Web Services, WebSphere. According to Microsoft, the results of this benchmarking exercise shows the .Net is a superior platform for the development of Web Services, however, this simply is NOT the case.

In this response, we will not only show that IBM offers the superior platform for creating Web Services, but will also point out how Microsoft has attempted to mislead customers. More specifically, we will show that when the benchmark is properly run, (i.e., not in a manner intended to produce a predetermined result), WebSphere created the sample Web Service faster, cheaper, using fewer steps, with less lines of code, and in a heterogeneous environment, not simply a Microsoft environment. In other words, there is no doubt that WebSphere is the superior platform for developing Web Services.

Building Web Services the right way

Before we tell you what Microsoft actually did, lets briefly discuss the right way to build a Web Service…the WebSphere way. There are two important points about building Web Services the right way:

1. Use open standards to ensure portability across platforms
2. Use the most advanced, generally available tool to reduce hand crafted code

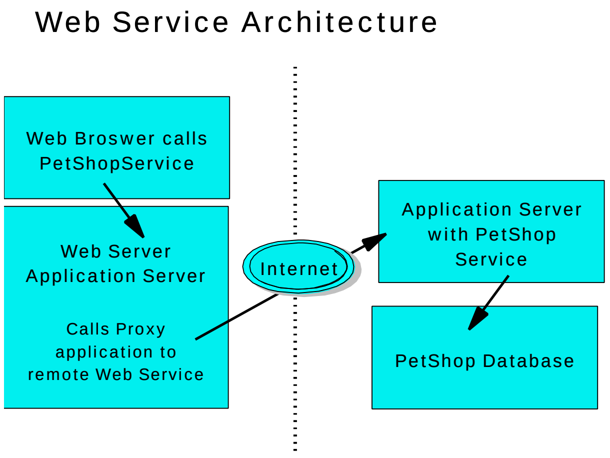
Microsoft did neither of these things.

Web Services represent an important step forward for interoperability by using standard methods to communicate and describe applications and data. Most companies want more; they want to ensure their application investment is protected across multiple platforms and across vendors. By selecting J2EE, companies can create Web Services than run from Linux to zOS and many additional operating systems and middleware environments obtained from multiple vendors.

Using WebSphere, the lines of handcrafted code can be reduced from 106 using .NET to a single line of handcrafted code, a 99% reduction. WebSphere includes advanced Web Services wizards and XML tools to rapidly create Web Services from existing applications. Integrated application servers enable rapid testing and debugging across multiple platforms, including NT, Linux, iSeries and zSeries.

What Microsoft did

Microsoft wanted to show that .NET is superior to J2EE for creating Web Services, so they used a Sun sample called the PetShop. As previously noted, Microsoft “chose IBM WebSphere 4.0 because we believe IBM WebSphere 4.0 has the most advanced support for XML-based Web Services of the major J2EE application servers on the market[1].” IBM appreciates that Microsoft recognizes our industry leading support for Web Services.


Web Service Architecture
Web Broswer calls PetShopService
Web Server Application Server
Calls Proxy application to remote Web Service
Internet
Application Server with PetShop Service
PetShop Database

Microsoft used a beta release of Visual Studio.NET to create a relatively simple Web Service. The Web Service allows a browser to enter an order number into a Web page. Next, the application server uses the order number to invoke a Web Service to backend server. Finally, the backend server runs a stored procedure to look up the order details and return the information. Middleware includes IIS and SQL Server from Microsoft – of course, an all Microsoft implementation. For the IBM implementation, they use WebSphere Application Server V4.0 and Oracle 8i for the database server. Care to guess what Operating System was used?

They made the service significantly more complicated by using a stored procedure, rather than using SQL calls to the database. Possibly they chose this architecture so they could require an additional tool for the IBM architecture? Using SQL calls would have further reduced the time and effort to create the Web Service since the application could have been created using database wizards. Microsoft also made the service more complicated since the stored procedure returns an array of data, rather than simple data structures. Stored Procedures can be written in a portable fashion, but are usually vendor specific in their implementations. Migrating stored procedures is one of the more difficult aspects of migrating between databases.

To create the Web Service for IBM, they used the following tools:

- IBM WebSphere V4 Admin Console
- IBM WebSphere Assembly Tool
- IBM SoapEarEnabler
- IBM WebSphere Studio V4.0[2]
- VisualAge for Java V4.0
- IBM Web Services Toolkit (from AlphaWorks)
- Oracle DBA tool.

[1] “Building Web Services using with Microsoft .NET vs. IBM WebSphere 4.0.doc,” October 2001

[2] The paper does not state whether Professional or Advanced Edition was used.

The summary says 6 tools were used, but doesn't list VisualAge for Java. Possibly they considered it a part of WebSphere Studio, since a copy is included with Studio. It's also not clear why they list standard features of WebSphere Application Server (Admin Console, Assembly Tool, and SoapEarEnabler) as three separate tools. Perhaps it is to mislead you into thinking it is more difficult or costly develop on WebSphere.

The WebSphere Studio V4.0 wizard for creating Web Services from a Java bean only supports simple parameters (integers, strings, etc.) on its methods. So, they wrote the Web Services Description Language (WSDL) file by hand and used the IBM Web Services Toolkit to generate the Java skeleton beans, wrote the contents of the implementation in VisualAge for Java, turned the Java code into a J2EE application using Application Server tools and so on. Naturally their experience was not very positive by using this process and so many different tools.

IBM recommends using WebSphere Studio Application Developer V4.0 ("WebSphere Studio"). Even when the study was done, the beta version could be freely downloaded, and would have allowed them to use one tool to complete the Web Service. The Beta release did not support arrays in the Web Services wizards (used in the PetShop Java bean), so more hand crafted code would be needed than in the analysis below. WebSphere Studio, announced November 5, 2001 already available, provides a complete, integrated environment for developing, deploying, testing, and debugging Web Services.

Microsoft Misdirection

Simplicity to code a Web Service

Microsoft has shown summaries where .NET creates the Web Service in less than 4 hours (238 minutes) with 9 steps, while WebSphere requires over 9 hours (550 minutes) with 14 steps. **But using Application Developer requires 8 steps and just over 3 hours (184 minutes).**

Furthermore, Microsoft .Net required 89 lines of custom code to create the Web Service from the business logic. WebSphere Studio required NO lines (that is, zero lines) of custom code. All code for the Web Service was generated via our wizards.

Microsoft .NET also required 17 lines of code to consume the Web Service. WebSphere Studio only required 1 line (that is, one line) of handcrafted code to consume the Web Service. All but one line of the JSP code could be generated from the wizards.

Total Microsoft .NET handcrafted code to create and consume a Web Service:	106
Total WebSphere Studio handcrafted code to create and consume a Web Service:	1

A 99% reduction in the number of lines of handcrafted code!

The Microsoft analysis of WebSphere used a very different process for building the Web Service, including hand coding of the Web Services Description Language (WSDL) file. The process below uses the same process and completes more in one less step.

Summary of Steps in Development Process

Step	Description	Tool	Time
1	Launch the Web project wizard to create the PetWebServiceEAR project and the PetWebService Web in a single step.	WebSphere Studio	1 minute
2	Create the PetWebService Class with business logic to access Stored Procedure	WebSphere Studio	90 minutes
3	Create WebSphere Test Server Configuration and Instance	WebSphere Studio	1 minute
4	Configure WebSphere Test Server Data source settings (to complete the stored procedure call)	WebSphere Studio	2 minutes
5	Launch the Web Services wizard to generate WSDL files, SOAP deployment descriptor, client proxy code, sample application, deploy to WebSphere 4.0 Test Environment, and deploy to UDDI Registry	WebSphere Studio	10 minutes
6	Test client proxy code and Web Service through to the database (debug Java bean business logic)	WebSphere Studio	30 minutes
7	Use Java Bean Web Creation Wizard to generate input page and results page from Java class files	WebSphere Studio	20 minutes
8	Build and test application, able to debug through all 3 tiers	WebSphere Studio	30 minutes
Total 8 steps			**184**

Open versus proprietary, Generally Available versus beta

The following was our initial response, published on the Web to begin focusing on the important decision criteria that customers should be using as they move forward with Web Services. IBM believes customers want open standards and implementations that run across heterogeneous platforms and middleware. IBM believes in shipping products to customers, not merely beta after beta after beta.

> We appreciate Microsoft's recognition that IBM and WebSphere are leaders when it comes to delivering Web Services capabilities, and want to take a moment to set the record straight. The comparison is misleading in many respects, but should at least have an apples to apples comparison. Instead it has an inconsistent use of alpha, beta and Generally Available technologies.
>
> But more importantly, Microsoft seems to have forgotten the Internet and Java are all about choice - choice of hardware, choice of Operating System, choice of language, choice of development platform - which translates into business flexibility. Businesses need to leverage current and ongoing investments, not rebuild from scratch.

The clear message from Microsoft is their tools fail to run on leading industry products and platforms, like Java, J2EE, Sun Microsystems Solaris, IBM AIX, IBM i-series and IBM z-series. We can all quote benchmarks till we're blue in the face, but at the end of the day it's about proprietary, single vendor versus open and multi-vendor.

Metric	**Microsoft .Net**	**IBM WebSphere**
Choice of Hardware	**Intel**	**HP 9000, Intel, i-series, PowerPC, Sparc, z-series**
Choice of Operating System	**Windows**	**AIX, HP-UX, Linux, OS/400, OS/390, Solaris, Windows**
Homogenous or Heterogeneous environment?	**Homogenous**	**Heterogeneous**
Extend infrastructure or build from scratch?	**Build from scratch**	**Extend infrastructure**
Use alpha, beta or Generally Available products?	**beta**	**Primarily Generally Available products**
Choice of tools providers?	**Only .NET from Microsoft**	**IBM, Borland, WebGain, Macromedia**
Java and J2EE support	**No, Hates Java**	**Yes, delivers industry standard support**
C# Support	**Yes**	**No**

As another proof point on open implementations, IBM has donated to Eclipse.org the Eclipse software, the basis for WebSphere Studio Workbench. Eclipse heralds a new era for development tools by providing the capability to seamlessly integrate tools from multiple vendors.

Ease of learning

Microsoft claims that "Web Services are an integral part of VisualStudio.NET, with ready-made project templates and extensive documentation in the core product. In addition, almost all of the work to create a Web Service is handled automatically by the tool. Hence the learning time is very low."[3] However, in reality, the entire .NET framework is a radically different set of middleware from the COM and DCOM architectures they have been promoting for many years.

In contrast, the Web Services paradigm is only a small enhancement to the J2EE standards for Web Applications.

[3] "Building Web Services using with Microsoft .NET vs. IBM WebSphere 4.0.doc," October 2001

WebSphere Studio has exceptionally good documentation with:

- Scenario sections to explain building complete applications, including Web Services applications, in a step by step fashion. The Hospital scenario shows the user how to create Web Services applications that link into two remote services.
- Concept sections to explain Web Services concepts like UDDI, WSDL, SOAP, and DADX
- Task sections to explain how to perform specific tasks, such as Creating a Web Service from a Java Bean

Customers can easily learn to create a Web Service using WebSphere Studio within a day, just as Microsoft claims users can learn .NET.

Cost of creating this simple Web Service

Microsoft places a dollar cost on creating a Web Service based on the number of hours to learn the new tools and create the Web Service. As we've seen in the last 2 sections, WebSphere Studio reduces the time to create the Web Service, drastically reduces the number of lines of handcrafted code, and has comparable learning costs. The resulting cost comparison now gives the advantage to WebSphere.

	Microsoft .NET	**WebSphere Studio**
Total time required to build the Web Service	11.97 hours (including 8 hours learning time)	11.02 hours (including 8 hours learning time)
Cost for this Web Service	$2,393 (@ $200.00[4] per hour)	$2,203 (@ $200.00 per hour)

After doing the costs for the single case, they extrapolate to a more complete project. In the larger project, WebSphere Studio would look even better, since most of the time spent by developers would be doing coding where we complete the tasks 25% faster with 99% less lines of code to write.

In the Microsoft comparison, they also add the costs to deploy the Web Service and use an incredibly misleading comparison for Server costs.

Costs of Microsoft .Net Framework

Microsoft claims a cost of only $3,999 per server for middleware software, so for 4 8-way processors, the costs are $15,996. However, this does NOT take into account their Client Access Licenses. Microsoft requires an additional charge for each user that is authenticated at the server. In the pet shop scenario, it is unlikely that any user could go to a public web site to request this order information without first logging into the system. Nor would the back end Web Services allow any server to request this information without proper authentication.

[4] The estimated developer cost of course may vary widely based on developer skill sets.

Microsoft offers packages of Client Access Licenses (CALs) at various prices, for example, 200 users for $9,999. An Internet Connector License Option is available for $1,999, which allows unlimited Internet users to access a server. An Internet user is defined as "any person connected to the Internet, other than a person employed by you, or otherwise providing goods or services to you or on your behalf."[5] Therefore, use of this license will depend on the relationship between the Pet Shop and the people requesting the Web Service.

4,000 affiliated resellers or suppliers would require 20 licenses of the 200 user CAL package. These CALs on the two servers consuming the web service would require a payment of an additional $399,960 (2 * 20 * $9,999). Additional CALs may be required on the back end servers, depending on how many authenticated front end servers access the Web Service.

It is also misleading to compare Windows 2000 to a real Web Application Server. According to Gartner Group, Windows 2000 does offer many functions, "however, you must purchase BizTalk Server, Host Integration Server and Application Center 2000, the products that extend their core capabilities into a fully functional application server.[6]" A combined license for BizTalk Server 2000 and SQL Server is $9,749 per processor. To add this to the 4 servers in the Microsoft presentation would add $311,968 to the Microsoft server costs.

It is very difficult to generalize the costs for a Microsoft .Net environment with so many variables. Client Access Licenses and additional servers can easily add hundreds of thousands of dollars to the costs quoted in the Microsoft white paper.

Cost of WebSphere Application Server

Microsoft claims this Web Service application requires Enterprise Edition, priced at $35,000 per CPU. Thus, on the 4 servers with 8 CPU's each, they calculate a cost of $1,112,000 (4 x 8 x $35,000). Yet, Enterprise Edition is certainly not required in the application. In fact, their white paper shows Advanced Edition Single Server, dropping the cost to $256,000 (4 x 8 x $8,000).

Summary

WebSphere Studio is:

- Generally Available
- Based on an open source platform
- Supports open J2EE standards
- Deploys to multiple application servers
- Deploys to multiple server and hardware platforms
- Generates all code required for PetShop Web Service creation
- Requires only a single line of code for PetShop Web Service consumption

The WebSphere software platform is clearly superior for creating and consuming Web Services.

5 Source: http://www.microsoft.com/windows2000/advancedserver/howtobuy/pricing/changes.asp

6 Source: Gartner Group report, Microsoft: An Application Server Vendor?, 23 October 2001

Appendix

Step by step process to create Pet Shop Web Service with WebSphere Studio using the same process as Microsoft would use for .NET

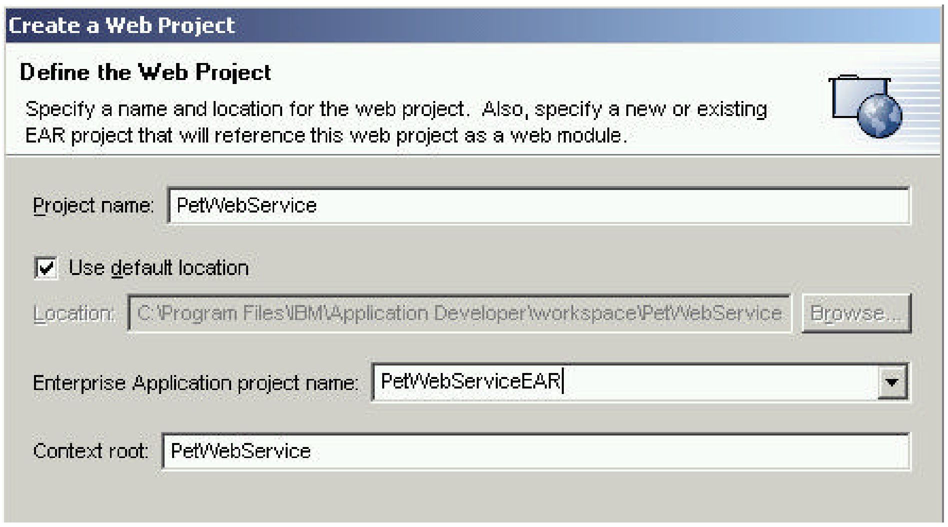
WebSphere Studio Application Developer:

Application Developer is a complete, integrated environment for building, deploying, testing, and debugging Web Services and J2EE applications. Application Developer also includes support for publishing and discovering Web services in UDDI Registries. Web services tools wrap existing Java bean or EJB components as SOAP-accessible services and describe them in the Web services description language (WSDL). Database developers can also use SQL as a programming language to quickly build data-aware Web services. Application Developer includes tools to wrap database access directly into a Web service using the DADX specification file. DAD Extension (DADX) is an extension of the Document Access Definition (DAD) file for IBM DB2 XML Extender. A DADX document enables the creation of Web Services that store and retrieve XML documents managed by XML Extender.

The web services in the comparison study are based on the .Net Pet Shop and the Java Pet Store application developed by Sun. The business requirement for the web service is to return detailed information about an order that has been created through the web application. The IBM implementation of the Web service was done via a Java bean that used a stored procedure call to retrieve order data. There are two ways this Web service could be implemented in WebSphere Studio. The first is to follow the same architecture used in the Microsoft comparison. The second is to describe the Web service as a stored procedure call using the DADX specification mentioned above. In the latter case, the Web service would make a stored procedure call directly and no Java code needs to be written.

The following describes how the Java based implementation would flow in Application Developer while contrasting to the comparison study experience.

1. Launch the Web project wizard. Specify the name of the Web project - PetWebService - and also the name of the Enterprise Application project - PetWebServiceEAR. This **one step** creates the PetWebServiceEAR project and the PetWebService Web project, while Microsoft required two steps.


Create a Web Project

Define the Web Project

Specify a name and location for the web project. Also, specify a new or existing EAR project that will reference this web project as a web module.

Project name: PetWebService

[✓] Use default location

Location: C:\Program Files\IBM\Application Developer\workspace\PetWebService

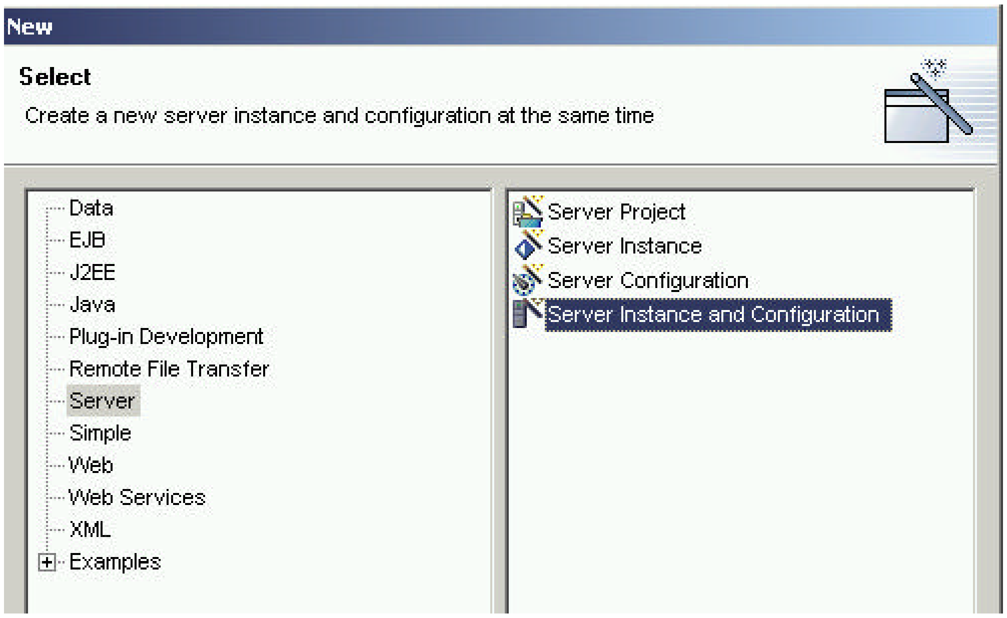
Enterprise Application project name: PetWebServiceEAR

Context root: PetWebService

2. Create the Java class implementation on which the Web service is based in the PetWebService Web project. The Java class would be essentially what is provided with the comparison study (it opens a datasource connection to the database and performs a stored procedure call to retrieve order information based on an order id). *This is fundamentally different from how the study was done because they started by handwriting the WSDL files.*

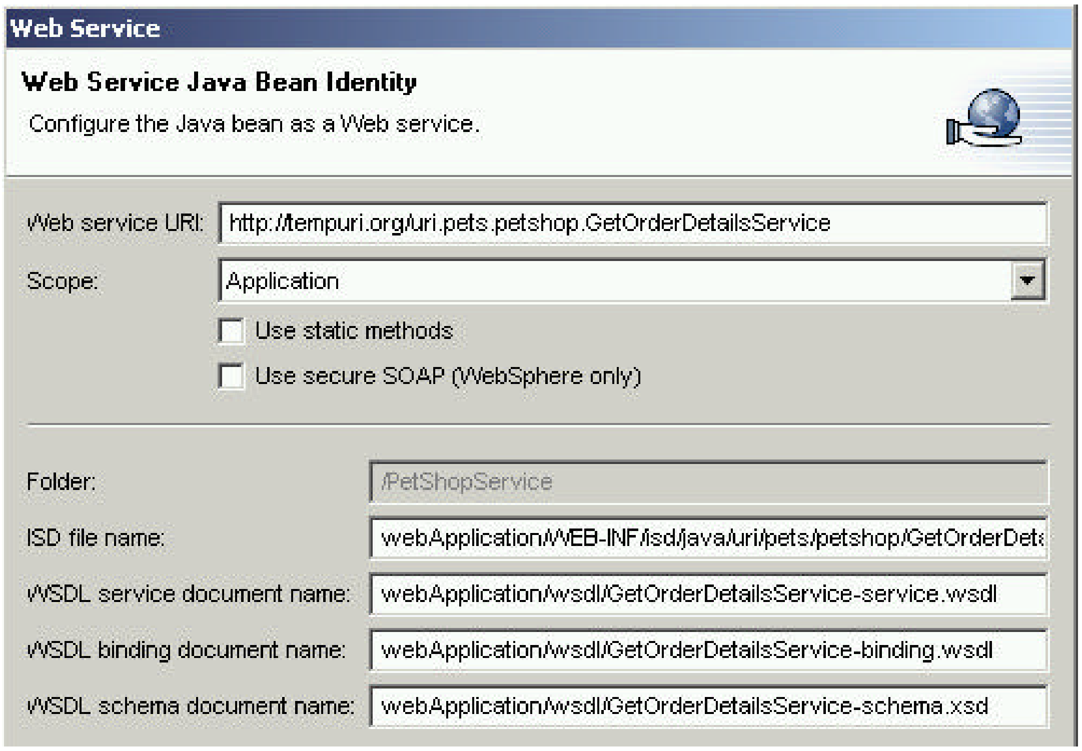
3. Create WebSphere Server Instance and Configuration. This step creates an instance of the WebSphere Application Server to be used in the test and debug steps.

4. Configure the data source for the PetShop stored procedure in the WebSphere Configuration. Simply double click on the WebSphere Configuration, select the Data Source tab and fill in the data source information for the PetShop stored procedure.



Create a new server instance and configuration at the same time

5. Launch the Web Service wizard, selecting the Java file created in step 2. The wizard takes the user through the pages where they can select which methods to expose in the Web



Configure the Java bean as a Web service.

Web service URI: http://tempuri.org/uri.pets.petshop.GetOrderDetailsService

Scope: Application

☐ Use static methods

☐ Use secure SOAP (WebSphere only)

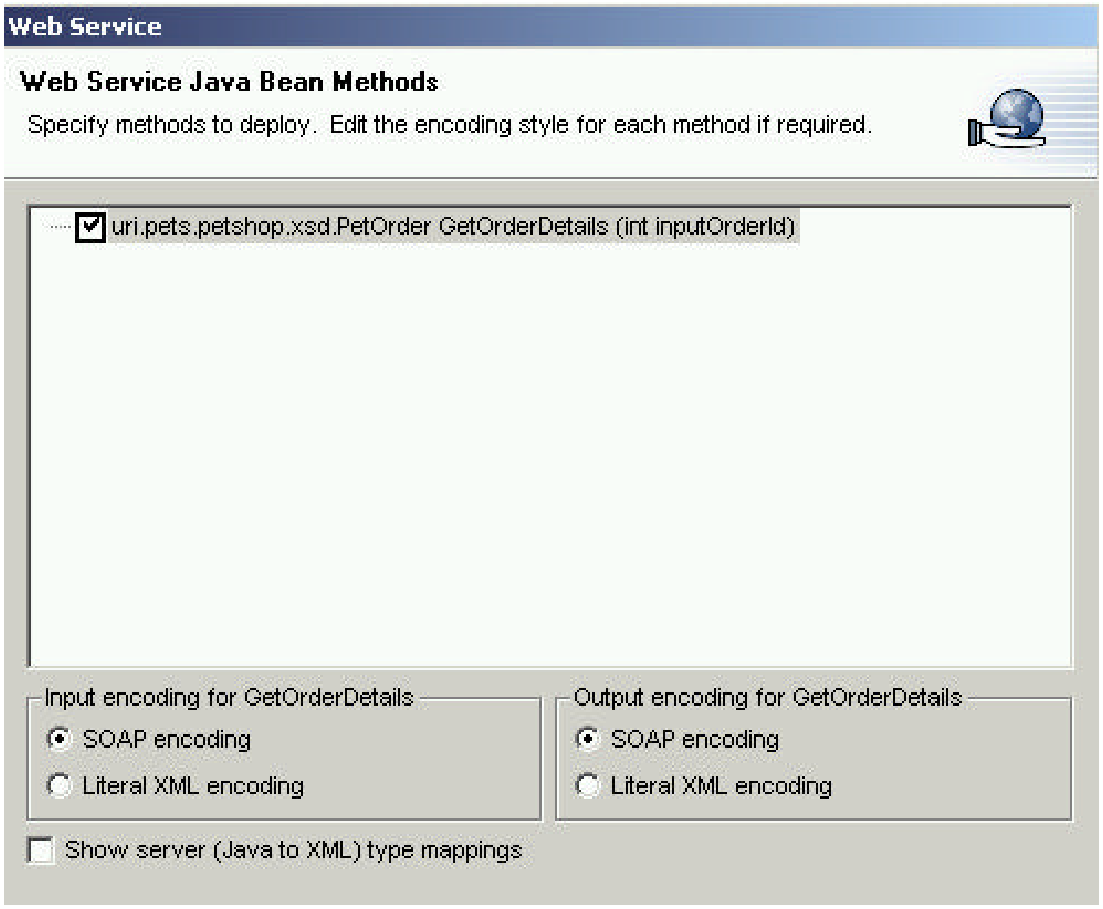
Folder: /PetShopService

ISD file name: webApplication/WEB-INF/isd/java/uri/pets/petshop/GetOrderDet

WSDL service document name: webApplication/wsdl/GetOrderDetailsService-service.wsdl

WSDL binding document name: webApplication/wsdl/GetOrderDetailsService-binding.wsdl

WSDL schema document name: webApplication/wsdl/GetOrderDetailsService-schema.xsd service, perform custom Java/XML mappings on parameters, if necessary. The wizard generates WSDL files, a SOAP deployment descriptor and deploys the Web service to the WebSphere 4.0 Test Environment (included with WebSphere Studio). *Since WebSphere Studio Application Developer GA and Site Developer Beta 2 handle Java classes whose methods contain complex type parameters, there is no need to handcraft the WSDL files, as was done in the comparison study. Also there was no need to run the WebSphere Application Assembly tool, the soapearenabler tool, soap admin tooling, or the WebSphere deployment tooling. All the steps necessary to deploy the Web service application are performed by the*



Web Service Java Bean Methods

Specify methods to deploy. Edit the encoding style for each method if required.

[✓] uri.pets.petshop.xsd.PetOrder GetOrderDetails (int inputOrderId)

Input encoding for GetOrderDetails: [● SOAP encoding] Literal XML encoding

Output encoding for GetOrderDetails: [● SOAP encoding] Literal XML encoding

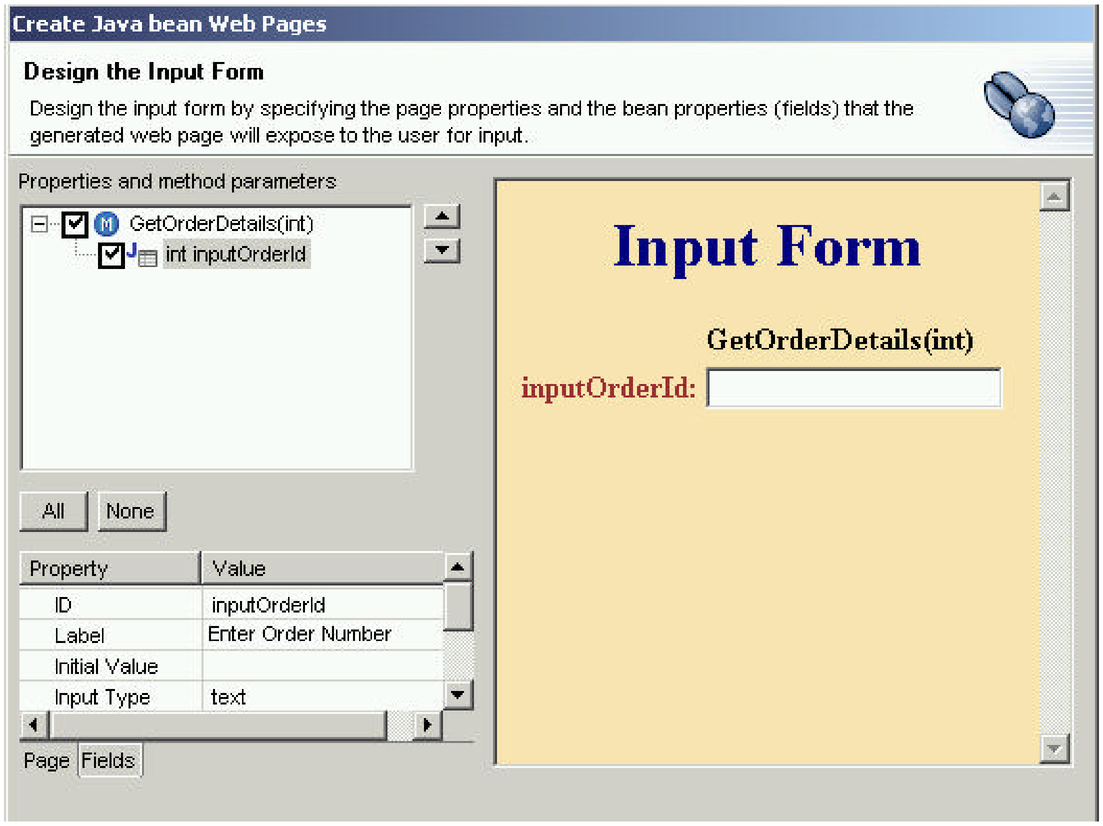
☐ Show server (Java to XML) type mappings

Web Service wizard. At the user's choice, the wizard then proceeds to generate the Web service Java client proxy code, launch a Test client, generate a sample application, and finally optionally publish the Web service to a UDDI Registry.

6. Test and debug client proxy code and Web Service through to the database. To reproduce the client usage of the PetShop Web service, launch the Web Services Client wizard and select the PetShop-service.wsdl file. The wizard takes the user through the pages where they can perform custom XML/Java mappings on parameters (if necessary), creates Java client proxy bean (which includes creating Java beans corresponding to the complex parameter and return types specified in the WSDL document) and launches the Test client. Since WebSphere Studio provides a complete test environment for testing application running in WebSphere Application Server, the developer can debug their Web service application in WebSphere Studio. Since the Web service application is a J2EE application (contained in an EAR file),

it is deployed to WebSphere 4.0 using the Test Environment inside WebSphere Studio or alternatively the EAR file can be exported and deployed to J2EE Server. There is no need to run WebSphere Application Assembly tool, soapearenabler tool or soap admin tooling. Advanced capabilities like incremental compile and swipe and execute make this an extremely productive test environment.

7. After testing the service, the user can use the sample application generated in the previous step to build a Web based UI using the Web tooling in Application Developer. The sample application is a set of JSPs that demonstrate how to call the Web service proxy client. Using the Create New JSP from a Java Bean wizard, and selecting the Java class files created earlier allows the user to merely specify input and output fields to display. Both an input page (not done by Microsoft) and a results page are generated. By using the more complex

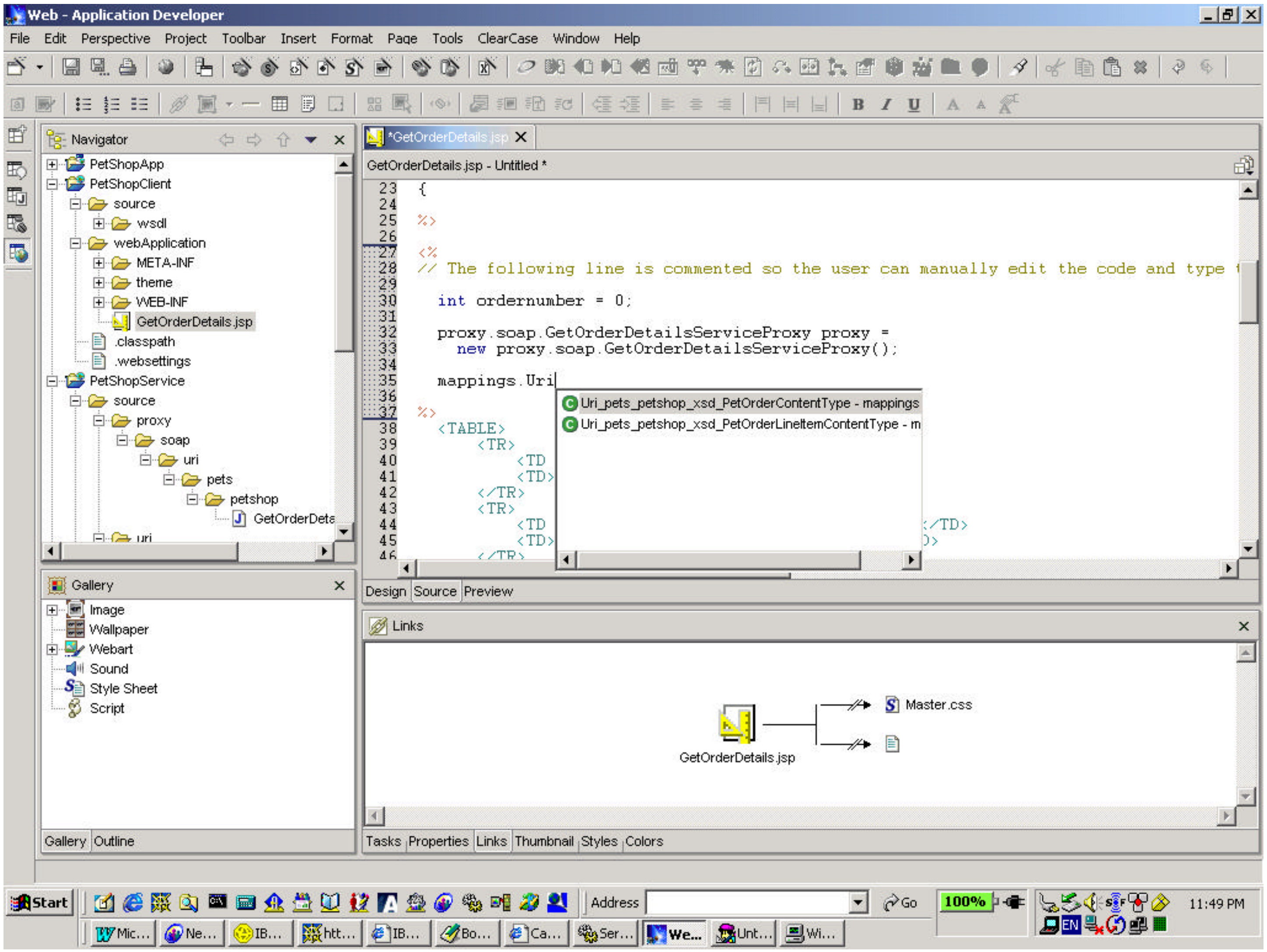

Create Java bean Web Pages
Design the Input Form
Design the input form by specifying the page properties and the bean properties (fields) that the generated web page will expose to the user for input.
Properties and method parameters
GetOrderDetails(int)
int inputOrderId
All
None
Input Form
GetOrderDetails(int)
inputOrderId:
Page
Fields

Property	Value
ID	inputOrderId
Label	Enter Order Number
Initial Value	
Input Type	text

array generated by the Stored Procedure, this step is slightly more complicated than if a traditional application were written simply using SQL standards to call the database. The wizard must be called twice and two pages created. The first page displays results for the order number, status, shipping and billing addresses. The second page displays results for the array. These JSP code fragment from the second page must be cut and pasted into the first page. Finally, this line of code must be handcrafted to associate the JSP to the proxy Web Service application. However, even this is made simpler by using code assist (crtl space) to select the URI and proxy methods.

```
mappings.Uri_pets_petshop_xsd_PetOrderContentType
orderdetails = proxy.GetOrderDetails(ordernumber);
```

By using wizards, and reducing the number of lines of handcrafted code down to one line, this task took 20 minutes, rather than 30 minutes for Microsoft.



```
{

%>

<%
// The following line is commented so the user can manually edit the code and type t

  int ordernumber = 0;

  proxy.soap.GetOrderDetailsServiceProxy proxy =
    new proxy.soap.GetOrderDetailsServiceProxy();

  mappings.Uri

%>
  <TABLE>
      <TR>
          <TD
          <TD>
      </TR>
      <TR>
          <TD                                              </TD>
          <TD>                                             )>
      </TR>
```

8. Build and test application, with the ability to debug through all 3 tiers. WebSphere Studio includes the ability to start a debug operation on a remote Application Server. It also allows for debugging of JSP code, including the ability to set breakpoints. Since most of the debugging of the application was done in step 4, we reduced this time from the Microsoft example to merely 30 minutes for debugging the new JSP code generated in step 5.

Step by step process to create Pet Shop Web Service with WebSphere Studio using a simpler process

Application Developer includes tooling that can wrap database access directly into a Web service. The Web service used in the comparison study could also be implemented using a DADX specification file, which would perform the stored procedure call directly and not involve a hand-written Java implementation at all.

1. Launch the Web project wizard. Specify the name of the Web project - PetWebService - and also the name of the Enterprise Application project - PetWebServiceEAR. This **one step** will create the PetWebServiceEAR project and the PetWebService Web project, while Microsoft required two steps.

2. Create a DADX Group using the Web Service DADX Group configuration wizard in the PetWebService Web project. Specify the JDBC source, database name, and other database connection specific information.

3. Create a PetWebService.dadx file that contains the definition of the stored procedure call. An example of a dadx file that performs a stored procedure is included with Application Developer.

4. Follow steps 3 through 6 from the Java based scenario above, but instead of selecting the Java file in the Web Service wizard, the user would select the PetWebService.dadx file.

The complexity with this particular DADX scenario is not in creating or deploying the service, but in handling the complex response from the stored procedure. The response contains multiple parts, and this presents a problem for the SOAP runtime on the client. The SOAP runtime can only handle one return type, not many, as is the case with this stored procedure. This whole issue is bypassed with the Java bean implementation of the service because it collects the return parts into one complex type and returns that from the Java service. So, we could create Java clients for the HTTP Get or HTTP Post protocol (where the DADX stored procedure service returns XML schema), but some processing will have to done on the client side to extract the information out of the returned XML.

WebSphere Studio's strength is not only well integrated end-to-end Web services tooling, but also that it is a complete J2EE development/test environment which includes Web tooling, EJB tooling, Database tooling, XML tooling, WebSphere test environment, and of course Java IDE.