Migration Guide: Web Service Model to Adaptive Web Service Model

Applies to:
Netweaver Web Dynpro, 700 SP8 onwards, NetWeaver04 SP17 onwards.

Summary
This document is intended to describe the reasons of Web Service Model deprecation and the process to be followed in adopting Adaptive Web Service model. The document acts as migration guide for the developers of Web Dynpro application to help switching to Adaptive Web Service Model (AWS) from Web Service model. This document does not cover all the possible combination for Web Service model but acts as a hint about the steps involved in migration.

Author: Dhawal Joshi, Nidhi Rajshree, Raghuvira Bhagavan
Company: SAP Labs India Private Ltd.
Created on: 02 July 2007

Author Bio
All the three authors are associated with SAP Labs and work for the Web Dynpro Tools team.

Dhawal Joshi works as a Development Lead and has been associated with SAP Labs since Nov 2003. His areas of expertise include CMI, Web Dynpro components and Mobile Web Dynpro.

Nidhi Rajshree works as Development Specialist and has been associated with SAP Labs since January 2006. Her areas of expertise include Web Dynpro Models.

Raghuvira Bhagavan works as Development Specialist and has been associated with SAP Labs since August 2004. His areas of expertise include Web Dynpro View Designer.
Introduction

The deprecation of Web Service Model means that SAP recommends the entire Web Dynpro application programmer community to avoid using Web Service model for their applications. By deprecating this model type SAP wants to hint that Web Service Model import will be removed in future releases. Thus model import wizard will not have an option to import Web Service model but instead will have option to import Adaptive Web Service model. Although all the existing application which already has the model in form of Web Service will continue to work in similar manner as before, but in case if new enhancements has to be done then that will no longer be possible. For enhancements all the applications has to migrate to Adaptive Web Service.

Motivation

Main motivations behind the adoption of Adaptive Web Service Model are the short coming of older Web service model. The feature differences between the models are as follows

<table>
<thead>
<tr>
<th>Old Web Service Model</th>
<th>Adaptive Web Service Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>It does not support advanced security settings like non-SAP SSO authentication tickets, WS SOAP header.</td>
<td>Such security settings are supported by Adaptive Web Service Model</td>
</tr>
<tr>
<td>No support to import URL which requires authentication</td>
<td>Allows importing WSDL through URL which requires authentication (This requires the use of WS destinations which themselves currently require support of WSIL by the WS provider system; SAP WS provider systems – ABAP and J2EE - support WSIL)</td>
</tr>
<tr>
<td>Proxy settings are stored at design time and were not configurable at runtime.</td>
<td>Proxy settings are configurable at runtime using J2EE Engine NetWeaver Administrator; AWS model will use settings of WS-Runtime</td>
</tr>
<tr>
<td>Logical Ports are available at design time</td>
<td>Logical Ports are not available at design time</td>
</tr>
<tr>
<td>No direct support for nested structure, thus application programmer has to manually bind the</td>
<td>Direct support of nested structure, no such manual binding is required</td>
</tr>
</tbody>
</table>
### Migration Guide: Web Service Model to Adaptive Web Service Model

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Works on typed generation of standalone WS proxy classes</td>
<td>Typed wrapper on top of generic AWS model not requiring standalone WS proxy class generation.</td>
</tr>
<tr>
<td>Limited set of XML schema definition support</td>
<td>Improved support for XML schema type constructs in WSDL (for further details see Adaptive Web Service documentation)</td>
</tr>
<tr>
<td>Multi-dimensional Arrays were not supported</td>
<td>Full support for nested complex types / multidimensional arrays in XML schema types defined for WS.</td>
</tr>
<tr>
<td>No support of anonymous/ local types</td>
<td>Full support for anonymous / local types.</td>
</tr>
<tr>
<td>Based only on code generation at design time → every bug fix requires recompilation and re-deployment of application</td>
<td>Typed proxies based on generic base and/or delegation classes, bug fixes possible without new code generation</td>
</tr>
<tr>
<td>Metadata is read once at design time and was reused at runtime. No possibility of adoption of changed metadata</td>
<td>Adaptive in the way that metadata is read at runtime. This allows adoptions (in limited sense) to changed metadata</td>
</tr>
<tr>
<td>Limited support of data-types</td>
<td>Data type support at runtime improved (see Adaptive Web Service model documentation for details).</td>
</tr>
<tr>
<td>Does not allow renaming of model classes</td>
<td>Allows renaming of the model classes.</td>
</tr>
<tr>
<td>Does not require model instance to be passed for request execution</td>
<td>Creation and management of model instance(s) to which model objects are assigned have to be handled in application code.</td>
</tr>
</tbody>
</table>

### Description

There are several ways to model a Web Dynpro application. Although there is no perfect modeling way which fits to all the requirements but there are some general patterns and recommended way to model an application. One of such way is as follows; multiple Views within a component have View Context bound to Controller Context which in turn is bounded to Model. The migration steps described below is targeted for such context mapping (for details look at Appendix A – “Old WS Model Project Structure”). The concept which can be derived out of the steps can be applied with some minor changes to any of the context mapping pattern.

### Assumptions

- The user involved with migration has good working knowledge of Web Dynpro.
- The old WS model is bound to WD component controller context.
- These steps should be tried out and adapted as needed with one or several Web Services imported as old WS model in a separate project (if possible).
- Depending on project structure / components and the complexity of the model other ways to proceed may be easier. So the following steps can only be hints.
- To keep the migration guide, simple to understand – non-dc Web Dynpro project is used. These steps holds good even for dc Web Dynpro project in writable conditions.
Prerequisites

- Make sure the Web Dynpro project having old WS model is in properly compiling and running stage.
- Proxy Settings are maintained in the Preference page, if required. (Windows >> Preference >> Workbench >> Proxy Settings)
- Developer Studio is configured to an engine which supports Adaptive Web Service runtime (NetWeaver04 – sp17, 700 – sp8, CE 7.1)

Migration Steps

   - Right Click on Model
   - Select Create Model
   - Model Type selection Wizard pops-up
   - Select Import Adaptive Web Service Model
   - Press Next
2. Specify new Model Name and Model Package.
   a. Model Name -> AwsModel
   a. Model Package -> com.sap.migration.demo.model
   a. Press Next

3. Specify the destination settings. If user selects
   - “No logical destinations” - Then WSDL URL specified for importing will be used for metadata retrieval as well as for web service execution at runtime. WSDL URL can be changed after import via changing the corresponding model setting value. In case WSDL file imported is loaded from local file system this is mandatory as otherwise a runtime exception will be thrown as an attempt to load the WSDL at runtime from the given local directory of J2EE Engine.

   - “Use destinations for Metadata and execution” – This option allows user to configure the destination for metadata and for execution separately. By default Metadata Destination is mapped to “DEFAULT_WS_METADATA_DEST” and Execution destination is mapped to “DEFAULT_WS_EXECUTION_DEST” which can again be configured in J2EE engine settings for runtime. (See Appendix B – Configuring WS Logical Destination) You might want to define specific destinations according to the application area your WD application belongs to.

   - Default destinations selected for both metadata and execution
     a. Press Next

4. Choose a WSDL to be imported from the Remote Location / File System or Service Registry. Else simply enter the WSDL URL. URL used for migration is http://www.dataaccess.com/webservicesserver/numberconversion.wso?WSDL
     a. Press Next

5. Adaptive Web Service Model allows renaming the Model Class to be created. User can specify the alias for a namespace to be attached to all the Model Class as suffix or even can change the name of individual Model Class. Renaming also helps in removing any Model Class name clashes.

   a. Default Model Class names used
   a. Press Next
6. Press finish and model gets created.

7. Structurally the Adaptive Web Service model differs from Web Service model. Following fig shows the difference.
Structural differences between the two model types are:

- Adaptive Web Service does not have separate node as Logical Port to specify Http port, Proxy, Authentication data. These settings are now can be configured in J2EE engine, which are picked at runtime.

- The numbers of model class created are different in both the model types.
  - There are two request and two response classes for the old web service model
  - Whereas the Adaptive Web Service model has a model class extra for each of the request and response class as a complex type.

As a general rule the AWS model creates special classes named "Request_<operationName>" and "Response_<operationName>" which act as containers for input and output parameters of the corresponding web service operation. These classes have no direct correspondence to complex types defined in the WSDL. Relations of the request and response model class are created to all model classes corresponding to complex types of WS operation parameters. In addition there is a relation from the request to the response model class for assigning the response model object to the request model object after invocation of the WS.

The old web service model changes its structure e.g. according to the number of parts of the message element in the WSDL. For this sample the message only contains one part so that the request model class is identified with the message; if adding an additional part to the message the logic for creation of model classes is similar to the AWS model.

The relation between the model classes has also changed.

- For example each request class in the adaptive web service model has a relation to each complex type model class apart from a relation to its corresponding response class.

8. Identify the component and its views which have to be adopted.

   - DollarsView, WordsView and the Component Controller have to be adapted to the Adaptive web service model

9. Add the Adaptive Web Service model under "Used Models".
o Right click on "Used Models" node with in a component.

o Select “Add Existing Model”.

o Select Adaptive Web Service model AwsModel

o Press “Ok”

10. Open Data Modeler for the identified component.
    o Double click on ConversionComp under Web Dynpro Components

11. Create data link between Component controller and Adaptive Web Service Model.
    o Click on Create Data Link (Arrow) on the menu bar in the Diagram view.
    o Click on the Component Controller and drag and drop on the Adaptive Web Service Model

12. Due to data link binding, Model Binding Wizard pops-up.
    o Model Binding Wizard pops-up

13. Drag and drop the request Model Class(s) from model context to controller context. Select model class property and related model class as per the requirement on views.
    o For example, drag and drop Request_NumberToDollars from the left hand side to the context on the right hand side.
    o Edit Context Mapping wizard pops up.
    o Check all the attributes under the Request_NumberToDollars class. Rename context node Response to DollarResponse (WordResponse for the other request classes).
    o Press OK.
    o Similarly drag and drop Request_NumberToWords.
    o Press Finish
    o The mapping should finally look like this
14. Create the context structure in parallel to the existing context structure. For each view dealing with model (via context mapping) either adapt the existing mapping as described above or remove the model context mapping and map via component controller context (create context mapped to model nodes in controller context).

   o In the Diagram view of the Component, double click on the link between DollarsView and the Component Controller.
   o Edit Context Mapping wizard pops up.
   o Drag and drop Request_NumberToDollars from the new model to the context on the left hand side.
   o Check all the attributes under the class and press OK.
   o Press Finish.
   o Similarly drag and drop the request classes for mapping between WordsView to the Component Controller.
   o Save the Project.

15. Adapt UI element property to context bindings

   o Double click on DollarsView. Click on the Layout Tab.
   o Click on the Input Field.
   o Map the value to DNum.
   o Click on the TextView Field.
   o Map the text property to NumberToDollarsResult.
o Double click on WordsView. Click on the Layout Tab.

o Click on the Input Field.

o Map the value to UbiNum.

o Click on the TextView Field.

o Map the text property to NumberToWordsResult.

o Save All MetaData.

16. Make the implementation changes (if any) for example in wdinit()

   o Double click on ConversionComp under Web Dynpro Components.

   o Click on the Implementation Tab.
The `wdDoInit()` method should look like this.

```java
wdContext.nodeRequest_NumberConversionSoapType_numberToDollars().
    bind(new Request_NumberConversionSoapType_numberToDollars());

wdContext.nodeRequest_NumberConversionSoapType_numberToWords().
    bind(new Request_NumberConversionSoapType_numberToWords());
```

Delete the above lines and add the following:

```java
AwsModel model = new AwsModel();
Request_NumberToDollars req1 = new Request_NumberToDollars(model);
NumberToDollars dollars = new NumberToDollars(model);
req1.setNumberToDollars(dollars);
wdContext.nodeRequest_NumberToDollars().bind(req1);

Request_NumberToWords req3 = new Request_NumberToWords(model);
NumberToWords words = new NumberToWords(model);
req3.setNumberToWords(words);
wdContext.nodeRequest_NumberToWords().bind(req3);
```

As you can see the model object has to be passed to the request class and we have to explicitly initialize the complex types as shown above.

And the implementation of the execution method in the Component Controller of each operation should like this:

```java
try {
    wdContext.currentRequest_NumberConversionSoapType_numberToDollarsElement().modelObject().execute();
    wdContext.nodeResponse().invalidate();
} catch (Exception e) {
    //...
}
```

Comment the above two lines and add the following:

```java
try {
    wdContext.currentRequest_NumberToDollarsElement().modelObject().execute();
    wdContext.nodeResponse().invalidate();
} catch (Exception e) {
    //...
}
```

Similarly change the implementation for the other operations.

17. Build the complete project
18. Create an archive, deploy and run on the engine.
19. After successful runtime testing, delete obsolete entities from the project.
20. Delete the references of the old web service model from the data binding and context binding of the Component Controller. Delete the methods for the old WS Model execution. Delete additional
implementation that is specific for the old WS Model. Delete the 'Used Model' reference for the WD Component. And finally delete the old web service model.

21. Migration is completed.
Appendix A – Old WS Project Structure

The Above migration guide has been written with the existence of a current working application created using old web service model.

1. A web service model (WSModel, package: com.model) is created with the WSDL URL: http://www.dataaccess.com/webservicesserver/numberconversion.wso?WSDL

2. An application (WSApplication, package: com.sap) with default Component (ConversionComp), Window (HomeWindow) and view (HomeView) is created.

3. The WSModel is added to the Used Models in the ConversionComp component.

4. The request classes are bound to the component controller with all the attributes checked. A HomeContainer is embedded in the HomeView. Two views DollarsView, WordsView are created and mapped to the corresponding attributes in the component controller. A SelectionView is created with two radio buttons to navigate to each of the view. The latter three views are embedded in the HomeContainer with inbound and outbound plugs. The window should look like this:
5. `wdDoInit` method of the Component Controller implementation class for dollar operation should look like this and similarly for other operations.

   Request_NumberConversionSoapType_numberToDollars reqDollars =
   new Request_NumberConversionSoapType_numberToDollars();
   wdContext.nodeRequest_NumberConversionSoapType_numberToDollars().
   bind(reqDollars); 

   and the action button implementation should have this piece of code in it:

   ```java
   try {

   wdContext.currentRequest_ConversionsSoapType_numberToDollarsElement().mode
   lObject().execute();
   wdContext.nodeDollarsResponse().invalidate();
   }
   catch (Exception e) {

   }
   ```

   Similarly the other view should have the corresponding code for initialization and execution.

6. You should be able to successfully deploy and run the application.
Appendix B – WS Logical Destination

WS destinations are used to configure the WS provider system to read metadata of the WS and to execute the WS (if not using the “pure WSDL URL” scenario without destinations). The name of the destination is defined for the WS model during model import and the importer UI allows to use different WS destinations for reading metadata (metadata destination) and for execution (execution destination).

Use of WS destinations requires that the WS provider system supports WSIL (WS inspection language) for WS lookup – this is the case if the providing system is a SAP J2EE or an ABAP based system.

There are two logical destinations of which the names can be maintained for a WS model:

- Metadata destination: The destination used to retrieve metadata for the WS from at runtime
- Execution destination: The destination used to execute the WS at runtime

Both destination names are resolved at runtime by the WS-Runtime as underlying layer of the WS model. The above described destinations are to a big extent analogous to JCO destinations defined for and Adaptive RFC model.

Options for setting logical destinations

1. No logical destination: Both the destination for reading metadata and execution of the WS are taken from the WSDL used at design-time (settings are the empty String). This is almost the same as the current WS model. The difference will be that there is no lports.xml which can be maintained using the IDE to set internet proxy to be used, target address etc.; this now needs to be done via code if necessary.

2. Different Metadata and execution destination: Both can be entered. Default values: "DEFAULT_WS_EXECUTION_DEST" and "DEFAULT_WS_METADATA_DEST". At runtime these two destinations have to be configured. The execution destination is used to execute the WS, the metadata destination is used for metadata retrieval. Both destination names must be different for this option.

The option selected by default is 2.
After the WS is imported developer studio still allows user to configure WSDL URL, default Execution destination and default Metadata destination. This can be achieved by editing the model setting designTimeURL. This can be done by opening the model class (double click on model class) and switching to properties view with in the Web Dynpro perspective. You can also change the default destination names for metadata and execution destination.

You maintain a WS destination (e.g. the WS provider system for one of the destinations) using the NetWeaver Administrator of the J2EE system where the test bed is running as shown in the following screenshots

1. Select service “Web Services Security” and create destination with the name ref’d in the importer UI (e.g. “DEFAULT_WS_METADATA_DEST”)

2. Maintain the destination properties where URL is required for all destinations; for ABAP destinations additional properties (system name, client, language and user/pwd) have to be set.