WDJ: Adaptive Web Service Model – Controller Coding Explained

Applies to:
Web Dynpro for Java UI Development, SAP NetWeaver 04 SP Stack 17, SAP NetWeaver 04s SP Stack 6

Summary
This article de-mystifies the magic relationship between a Web Dynpro model and a Web Dynpro controller context at runtime, so that you really understand how to correctly implement the model-specific controller code within your own Web Dynpro applications: how to instantiate model objects, how to create a model object graph, how to bind executable model objects to a context node, what’s going on technically when executing a model object and why to invoke the IWDNode.invalidate() method of the response context node object.

In addition to these principle aspects this article describes some technical details which are specific for the Adaptive Web Service Model like tracing Web Service invocations and programmatically modifying Web Service invocation settings with the IWDWSInvocationModifier-API and the Service Extension Interfaces of the Web Service Runtime.

This article is an addition to the Web Dynpro tutorial Creating an E-Mail Client Using Web Dynpro’s Adaptive Web Service Model. Please read this SDN tutorial first to understand the principles of the Adaptive Web Service Model. Both articles are based on the same Web Dynpro sample application.

Author: Bertram Ganz
Company: SAP AG
Created on: 19 February 2007

Author Bio
After his studies in mathematics, physics and computer science Bertram Ganz finished his teacher training at a German grammar school stressing technical sciences. He has been a member of the Web Dynpro Java Runtime development team (SAP NetWeaver ESI Foundation UI) since 2002. The main focus of his work is on knowledge transfer, rollout, and documentation. Bertram regularly publishes articles on Web Dynpro in the context of the SAP NetWeaver Application Server. Bertram is co-author of the books “Maximizing Web Dynpro for Java” and “Java Programming with the Web Application Server”.

© 2006 SAP AG
# Table of Contents

WDJ: Adaptive Web Service Model – Controller Coding Explained .................................................. 1

Applies to: ........................................................................................................................................ 1

Summary .......................................................................................................................................... 1

Author Bio ........................................................................................................................................ 1

Table of Contents ............................................................................................................................ 2

Introduction ...................................................................................................................................... 3

Creating the Model Object Graph and Binding the Context to it ..................................................... 4

  Creating a new Model Instance ................................................................................................... 5

  Creating the Model Object Graph .............................................................................................. 5

  Binding the Executable Model Object to the Context ............................................................... 5

Implementing the Method sendEmail() ............................................................................................ 6

  Executing the Model Object of Type Request_SendEmail ......................................................... 6

  Invalidating the Model Node Response ...................................................................................... 7

  Supplying Relation Roles ............................................................................................................. 8

Programmatically Modifying Web Service Invocation Settings ....................................................... 9

  Code Sample ............................................................................................................................. 10

Tracing Web Service Invocations .................................................................................................. 10

Related Content ............................................................................................................................. 11

Copyright ........................................................................................................................................ 12
Introduction

Web Dynpro consequently follows the Model-View-Controller design principle (MVC Pattern) to separate different concerns in different entities. In this article I want to give a detailed explanation of a very fundamental relationship between two of the MVC layers:

«The relationship between Model and Controller at runtime»

or more precisely:

«The relationship between model objects and context elements at runtime»

In the Web Dynpro terminology this relationship is generally named context-to-model binding relation and from experience I know, that it is one of the most challenging topics every Web Dynpro application developer must tackle.

On the one hand the context-to-model binding is defined with the Web Dynpro Tools at design-time. That’s very easy and you can rapidly define complex nested context structures which reproduce the relationships between model classes in the controller context (see screenshot). At design-time there are terms like model, model usage, model class, executable model class, relation, model node, model attribute, supplying relation role, relation target or relation source.

On the other hand there is the context-to-model binding at runtime. That’s not so easy to understand because it refers to the context and to the model on an object instance level. This means you must deeply understand what’s going on in and between controller context and model at runtime. And here there are additional terms like model object, model object graph, context model node object, context model node element object or supply function and on Java source code level methods like execute() or invalidate().

In this article I want to de-mystify the magic relationship between a Web Dynpro model and a Web Dynpro controller context at runtime, so that you really know how to correctly implement the model-specific controller code within your own Web Dynpro applications.

In addition to these principle aspects this article describes some technical details which are specific for the Adaptive Web Service Model like tracing Web Service invocations and programmatically modifying Web Service invocation settings with the IWDWSInvocationModifier-API and the Service Extension Interfaces of the Web Service Runtime.

Note

Although this document explains model-specific controller coding in respect of the Adaptive Web Service Model the principle ideas are also applicable for other Web Dynpro models, like the Adaptive RFC model, implementing the Common Model Interface.
Creating the Model Object Graph and Binding the Context to it

EmailAdWSComp.java

```java
/** Hook method called to initialize controller. */
public void wdDoInit() {
    //@@begin wdDoInit()
    // create a new model instance (1)
    model = new EmailAdWSModel();
    // create model objects (2, 3)
    Request_SendEmail requestMO = new Request_SendEmail(model);
    SendEmail emailMO = new SendEmail(model);
    // associate model objects (4)
    requestMO.setSendEmail(emailMO);
    // bind executable model object to context node (5)
    wdContext.nodeRequest_SendEmail().bind(requestMO);
    //@@end
}
//@@begin others
private EmailAdWSModel model;
...
//@@end
```

The diagram above illustrates the technical details of the above Java code (the numbers in the comment lines refer to the numbers in the diagram) which can be divided in three tasks:

- Creating a new model instance
- Creating the model object graph
- Binding the Executable Model Object to the Context
Creating a new Model Instance

The Adaptive Web Service model instance is not maintained by the Adaptive WS model runtime. Consequently we must explicitly create a new Adaptive Web Service model instance of type EmailAdWSModel in the first step (1). This class was automatically generated during model import. To access the same model instance within all methods of the component controller it is stored in a private member variable. Its declaration is implemented within the final user coding area.

Creating the Model Object Graph

In steps 2-4 we instantiate and relate new model objects. In contrast to the former (non-adaptive) Web Service model or to the Adaptive RFC model the model instance must be explicitly created and passed to the model object constructors. Like EmailAdWSModel the model classes Request_SendEmail and SendEmail were automatically created with the Adaptive Web Service Model import. In addition to model object creation you must implement model object relation by calling the corresponding setter method: requestMO.setSendEmail(emailMO).

Important Note

Contrary to the Adaptive RFC Model or to the former Web Service Model the new Adaptive WS model object creation always requires a model instance. The model instance is not created and maintained by the Adaptive Web Service model itself. When creating new model object instances the model must be always passed to the corresponding constructor.

Binding the Executable Model Object to the Context

In step 5 we bind the executable model object requestMO to the context, or more exact, to the context model node Request_SendEmail:

wdContext.nodeRequest_SendEmail.bind(requestMO).

Binding a model object to a context model node means, that a new context model node element referencing the passed model object requestMO is added to the model node Request_SendEmail. With this reference together with the defined supplying relation roles of the inner context nodes the Web Dynpro Runtime automatically transfers data between the model and the context.
Implementing the Method sendEmail()

The Web Service invocation is implemented in the public method `sendEmail()` of the component controller `EmailAdWSComp.java`. The most important lines of code are the execution of the Web Service model object and the invalidation of the context node `Response`.

Executing the Model Object of Type Request_SendEmail

```java
public void sendEmail() {
    //@@begin sendEmail()
    ...
    // call Email Web Service and update dependent model node 'Response'
    wdContext.currentRequest_SendEmailElement().modelObject().execute();
    wdContext.nodeResponse().invalidate();
    ...
    //@@end
}
```

The actual Web service call is executed via calling the `execute()` method of the executable model object (1) which is currently referenced in the corresponding context model node (2). This model object is associated with the model object of type `SendEmail` (3) containing the e-mail data entered by the user (4) through data binding, context mapping and model binding. The instantiation of the model object (1) and its binding to the node `Request_SendEmail` (2) was implemented before in the method `wdDoInit()`.
Invalidating the Model Node Response

EmailAdWSComp.java

```java
//@@begin javadoc:sendEmail()
/** Declared method. */
//@@end
public void sendEmail() {
  //@@begin sendEmail()
  ...
  wdContext.currentRequest_SendEmailElement().modelObject().execute();
  wdContext.nodeResponse().invalidate();
  ...
  //@@end
}
```

After calling the `execute()` method of the executable model object (1), the Web Service returns two corresponding response objects (2 and 3). The relation (6) of the executable model object (1) thereupon points to the model object (2). However, the context model node `Response` (4) has not yet a node element (5) pointing automatically to the corresponding response object (2) in the model object graph.

Due to this fact, you as an application developer, have to explicitly invalidate the model node `Response` (4) contained as an inner node below the node `Request_SendEmail` in the context. This is executed with the code line `wdContext.nodeResponse().invalidate()`. Thus, the Web Dynpro runtime environment is told to automatically track the corresponding model relations (6 and 7) during access to the response node elements (8 and 9). In doing so, the references of the response node elements (8 and 9) in the context are adjusted to the response model objects (2 and 3) returned from the Web Service in the model.
For that purpose, the Web Dynpro runtime uses the *supplying relation roles* (see figure below) – defined in the model node *Response* and *SendEmailResponse* –, which is equivalent to the relations between the model classes Request_SendEmail to Response_SendEmail (6) and Response_SendEmail to SendEmailResponse (7).

**Supplying Relation Roles**

This figure also illustrates the dependency between *model class relations* and *inner context model nodes*. The relation between the executable model class Request_SendEmail and Response_SendEmail is named *Response*. In the context this relation is represented by the inner model node *Response* (same name like the model class relation) which is a child of the model node Request_SendEmail. A runtime this inner model node points to the model object of type Response_SendEmail, which is returned by the Web Service.

The returned result (in the example application this is just a BigDecimal object in the model object of type SendEmailResponse) is then displayed in an appropriate message text in the *message bar* of the Web Dynpro application.
Programmatically Modifying Web Service Invocation Settings

Web Service invocation can be influenced by implementing callback methods defined in the interface com.sap.tc.webdynpro.model.webservice.api.IWDWSInvocationModifier.

IWDWSInvocationModifier

This interface can be implemented by an application using the WS model in order to modify the web service invocation object just before it's execution.

Invocation modifiers need to be registered with the executable model class for which invocation is to be modified.

Additional public Methods based on Declaration

```java
void doModifyInvocation(Object port)
```

- Parameter `port`: The WS-Runtime invocation port. This parameter is passed by the Web Dynpro Java Runtime.

Modification of the invocation is done through service extension interfaces exposed by the Web Service Runtime where instances are created through factories using the `port` parameter passed to this method.

Service extension interface instances created in this method are only valid for one invocation and must not be refused for subsequent invocations.

```java
void doModifyAfterInvocation()
```

This method can be implemented to call methods on service extension interfaces retrieved in `doModifyInvocation(Object)` after invocation of the web service.

This API should only be invoked if really needed as it allows modifications on a technical level!

From the `port` parameter passed by the Web Dynpro Runtime a Web Service Runtime Service Extension Interface can be created which allows typed access to invocation settings.

The current Service Extension Interfaces (SEIs) and factories in package com.sap.engine.services.webservices.espbase.client.api are:

<table>
<thead>
<tr>
<th>SEI</th>
<th>Description</th>
<th>Interface and Factory</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPControlInterface</td>
<td>API that provides fine control over some http connection properties that the web service client uses to access the web service.</td>
<td>HTTPControlInterface&lt;br&gt;HTTPControlFactory</td>
</tr>
<tr>
<td>SOAPHeaderInterface</td>
<td>API provided for setting and getting SOAP headers by consumer applications. Methods for getting headers can be invoked after operation call, while setting output headers should happen before operation call.</td>
<td>SOAPHeaderInterface&lt;br&gt;SOAPHeaderIFactory</td>
</tr>
<tr>
<td>SessionInterface</td>
<td>API provided to manage HTTP sessions when using web services that support &quot;cookie&quot; sessions.</td>
<td>SessionInterface&lt;br&gt;SessionInterfaceFactory</td>
</tr>
</tbody>
</table>

SAP DEVELOPER NETWORK | sdn.sap.com
BUSINESS PROCESS EXPERT COMMUNITY | bpx.sap.com

© 2006 SAP AG
Code Sample

The `com.sap.tc.webdynpro.model.webservice.api.IWDWSInvocationModifier` interface can be implemented by an application using the Adaptive Web Service model in order to modify the Web Service invocation object just before it's execution.

Invocation modifiers need to be registered with the executable model class for which invocation is to be modified:

```java
public void doModifyInvocation(Object port) {
    HTTPControlInterface httpItf = HTTPControlFactory.getInterface(port);
    // timeout for this WS invocation is 120 sec.
    // don't use global config. setting of WS-Runtime
    httpItf.setSocketTimeout(120000);
}
```

Tracing Web Service Invocations

When an Adaptive Web Service application is started in debug mode for problem analysis purposes it is important to enrich the tracing information with additional exception details and the request/response trace of an invoked Web Service.

This is done by implementing the private controller method `traceWSInvocation()` which is called within the catch block of method `sendEmail()`. In case you also execute an Adaptive Web Service model class in other controllers, this method must be declared so that it is public.

The executable Web Service model object only comprises these details when the invocation log is enabled:

```java
requestMO.wdSetInvocationLogEnabled(logger.beDebug() ? true : false);
```

This line of code is implemented in method `sendEmail()`.

```java
public void sendEmail() {
    // call Email Web Service and update dependent
    // model node 'Response'
    wdContext.currentRequest_SendEmailElement().modelObject().execute();
    wdContext.nodeResponse().invalidate();
    ...
    try {
        //@@begin javadoc:sendEmail()
        /** Declared method. */
        //@@end
    }
}
```

```
public void sendEmail() {
    //@@begin sendEmail()
    IWDMessageManager msgMgr = wdComponentAPI.getMessageManager();
    Request_SendEmail requestMO =
        wdContext.currentRequest_SendEmailElement().modelObject();
    requestMO.wdSetInvocationLogEnabled(
        logger.beDebug() ? true : false);
    try {
        // call Email Web Service and update dependent
        // model node 'Response'
        wdContext.currentRequest_SendEmailElement().
            modelObject().execute();
        wdContext.nodeResponse().invalidate();
        ...
    } catch (CMIException ex) {
        msgMgr.reportException(ex.getLocalizedMessage(), true);
        this.traceWSInvocation(requestMO, ex);
    }
```
}  
}  

// declare member variable for model instance of type EmailAdWSModel  
/**  
* Enrich tracing information with exception details on an  
* invoked Web Service  
*/  
private void traceWSInvocation(  
  WSTypedModelClassExecutable requestMO,  
  Exception ex) {  
  logger.traceThrowableT(  
    Severity.ERROR,  
    wdComponentAPI.getApplication().getDeployableObjectPart().getName(),  
    ex);  
  if (logger.beDebug()) {  
    logger.debugT("Adaptive Web Service call request");  
    // trace Web Service request (header + SOAP request)  
    logger.debugT(requestMO.wdGetRequestLog());  
    // trace Web Service response (header + SOAP request)  
    logger.debugT("Adaptive Web Service call response");  
    logger.debugT(requestMO.wdgetResponseLog());  
    logger.debugT(  
      requestMO.associatedModelClassInfo().getModelInfo().toString());  
    // model object tree in XML format  
    logger.debugT(requestMO.toString());  
  }  
}  
//@end

Related Content

SDN Web Dynpro Tutorial: Creating an E-Mail Client Using Web Dynpro’s Adaptive Web Service Model

SDN Article: Using the Adaptive Web Service Model / https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/b2bc0932-0d01-0010-6d8e-cff1b2f22bc7
