Consuming SharePoint Web Services with SAP Web Dynpro using the Adaptive Web Services Model

Applies to:

NetWeaver 7.0 SP 06, SAP Web Dynpro for Java, Microsoft Office SharePoint Server 2007.

Summary

Interoperability between different systems becomes more and more important. In a recently pubished Technical Reference Guide about the Interoperability between SAP NetWeaver Portal and Microsoft SharePoint Technologies we described how to consume SharePoint Web Services within SAP WebDynpro for JAVA with an older Web Services Model.
In this whitepaper we will describe how the Adaptive Web Services Model can be leveraged that offers several advantages such as enhanced administration capabilities.

Authors: Dr. Antje Dziolloß and André Fischer

Company: SAP AG Germany, SAP AG

Created on: 20 May 2008

Author Bio

After her studies of computer science Dr. Antje Dziolloß worked several years at the Technical University as a lecturer. In the 90ties she joined the SRS Company in Dresden where she was involved in different development projects as developer and project manager. Now she works at SAP Germany as a SAP portal consultant with special interest in Web Dynpro Programming, Guided Procedures and UWL.

André Fischer works at SAP AG in the Strategic Alliance Microsoft Team where he addresses various kinds of interoperability topics regarding SAP and Microsoft solutions. Before joining SAP in 2004 Andre has lent his talents as an SAP technology consultant for eight years.

Table of Contents

Introduction

In a recently published Technical Reference Guide about the Interoperability between SAP NetWeaver Portal and Microsoft SharePoint Technologies we described how to consume SharePoint Web Services within SAP WebDynpro for JAVA with an older Web Services Model. In this whitepaper we will describe how the Adaptive Web Services Model can be leveraged that offers several advantages such as enhanced administration capabilities**.**

Challenges Calling Web Services in SharePoint

SharePoint provides different Web Services to make the functionality available externally. For our little sample application we use the search (query) Web Service.

The interfaces of Web Services are described in WSDL files (Web Service Description Language). The model creation is based on these WSDLs. WSDL is a standard and normally there should be no problems if provider and consumer of the Web Service follow the standard. SharePoint provides the WSDL by an URL which points to the SharePoint Server.

Unfortunately the Microsoft Web Service contains elements of type "any" that are not understood by the adaptive Web Service model. Because of this in our recent whitepaper we chose the option to use the older Web Service model rather than the Adaptive Web Services model.

The support of elements of type “any” is planned for an upcoming enhancement package. There is however a way to work around this issue using the currently existing products.

Looking for a Workaround

A little trick should help to overcome the obstacle. If you have a closer look to the WSDL you see that the WSDL contains several operations like "Query", "QueryEx", "Registration" and "Status". The any tag is used in the "QueryEx" operation which we don't want to use. For our needs the "Query" operation is the right one. Just remove all types, operations etc. that are related to the "QueryEx" and save the new WSDL. Using some XML Editor like XML Spy should help you to prevent errors. This new WSDL we can use for the model import during design time.

Our little trick sounds good but we encountered the following “problem” using it which is due to the “adaptiveness” of the Adaptive Web Services Model.

The Adaptive Web Service Model

Adaptive WebService model means that during runtime the interface is checked against the definition for the model for adapting to compatible changes like field extensions.

For this the *Web Service Destinations* are used that allow the configuration of a WS provider system for reading the metadata / WSDL at runtime and execution of the web service

If one configures the Web Service Destination such that it points to the URL of the Web Service provided by SharePoint the model would try to adapt using the original WSDL provided by the SharePoint Web Service.

This obstacle can be circumvented leveraging a feature of the Adaptive Web Service Model. The Web Dynpro Adaptive Web Service model distinguishes the metadata destination and the execution destination:

While the *metadata destination* is used to find the WSDL at runtime from which the Adaptive WS model reads its metadata the *execution destination* is used to execute the web service.

In our case we upload the edited WSDL into KM in the portal and use the URL of this KM document as the metadata destination. The URL of the SharePoint Web Service is still used for the execution destination.

The important point about Web Service destinations is that an administrator can change the above settings without modifying the code or configuration files of the application using Adaptive WS Model; so one should always use WS destinations for applications which are not only meant for test purposes.

This way it would be possible to place the changed WSDL also at any other location which can be accessed via a URL.

The Sample Application

This paper describes a little Web Dynpro for Java application that searches a SharePoint repository by calling a Web Service. The application consists of one single screen where you can enter the search string. After calling the Web Service the number of total available entries is shown and the table is filled with the first ten search results.

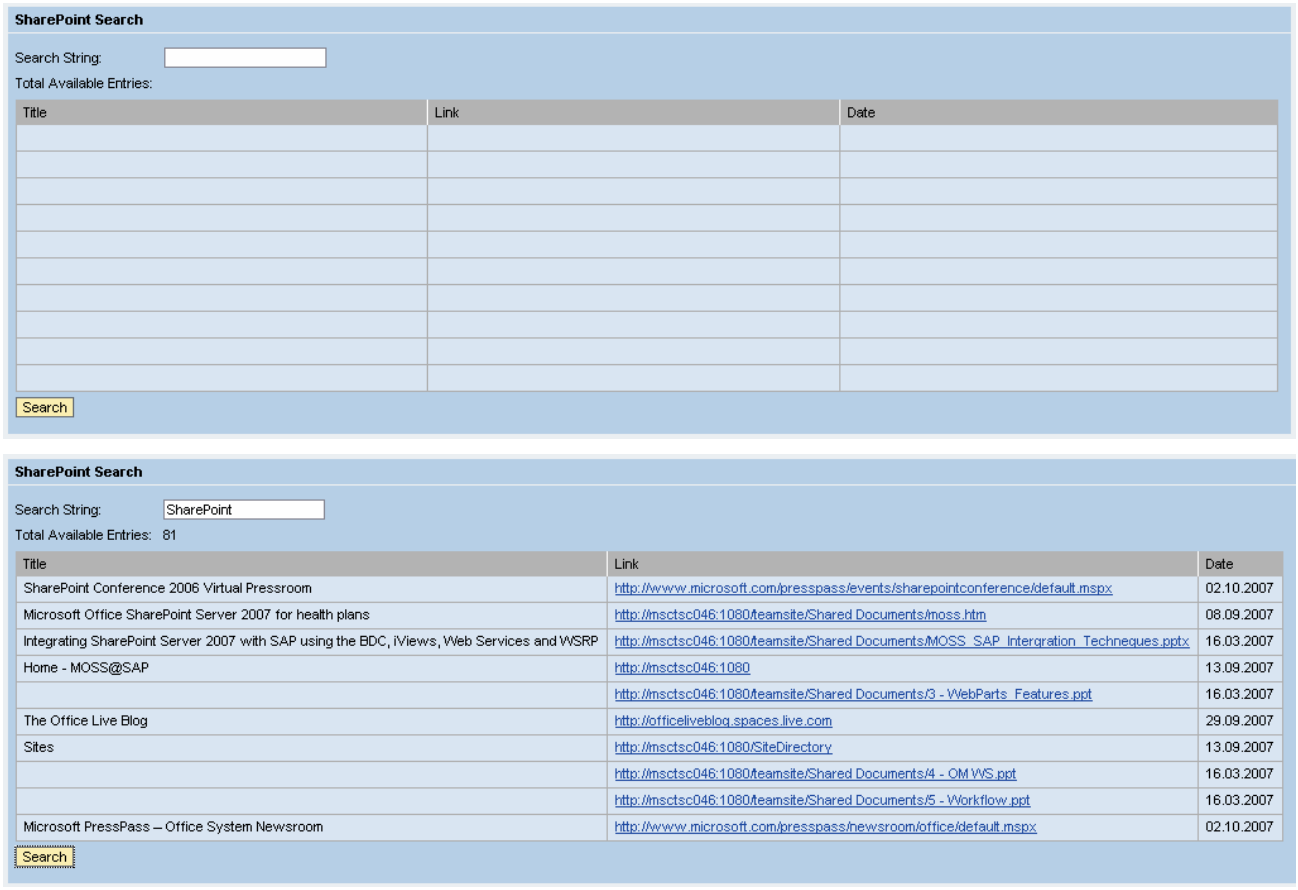

Figure 1 The Search Application

The sample project was created with Developer Studio for NW 7.0.

How to Section

Creation of the Project

The Web Dynpro project was created as a development component. It is generally recommended to work with Development Components because it is easier to integrate other components and libraries (in the sample the SAP XML toolkit).

Click on the menu entry "File-New-Development Component Project". Enter a vendor and a project name and select Web Dynpro as project type.

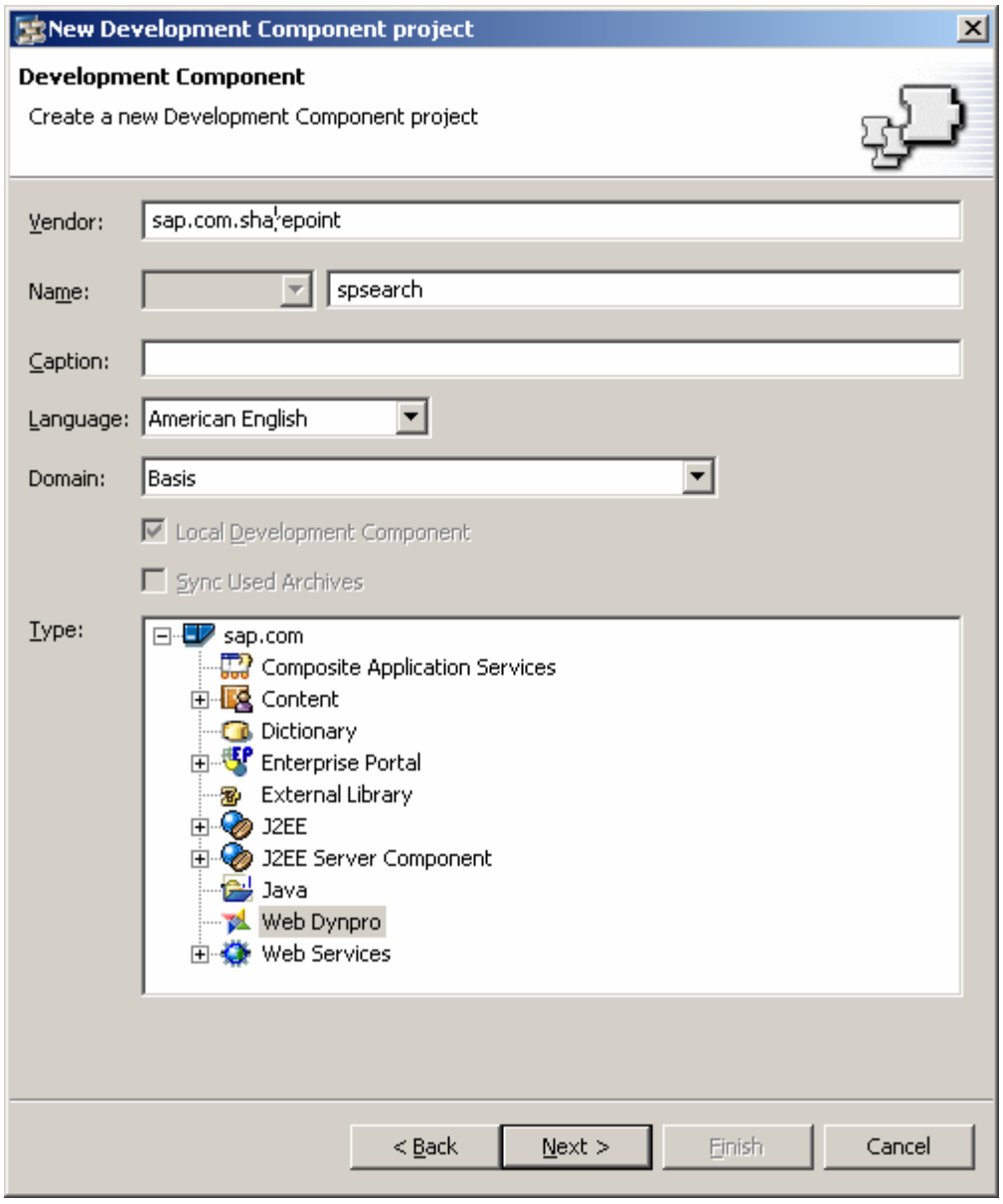

Figure 2 Creation of the Web Dynpro Development Component

On the next screen leave all preset inputs as they are. After clicking "Finish" you see the new project in the WebDynpro perspective of the NWDS.

Create a Web Dynpro Component be right clicking "Web Dynpro Components" on the new project structure. On the following screen enter the component name. Window and view names are generated automatically.

Check them and alter them on your needs. We embed a new view right here because we need exactly one. Enter a package name that fits your naming conventions.

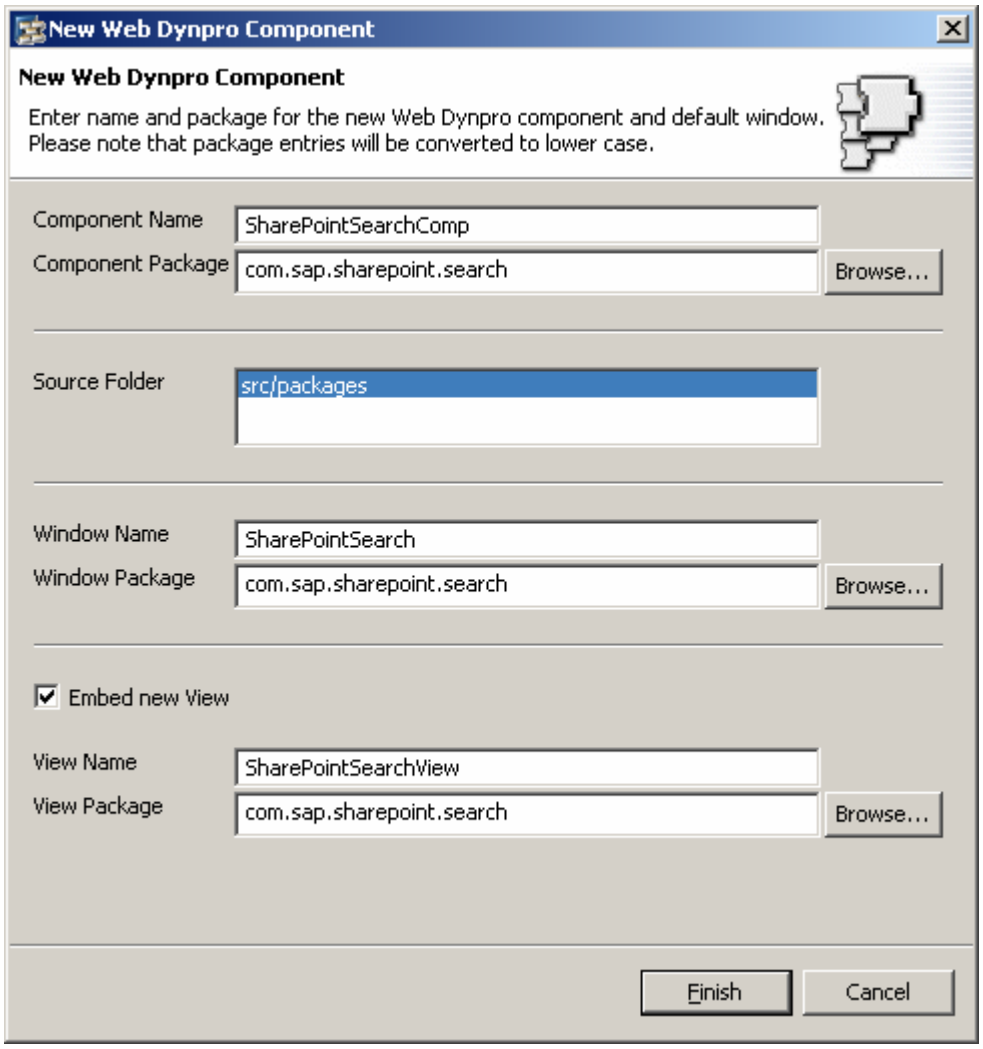

Figure 3 Creation of Web Dynpro Component

In the newly generated project create an application for the previously generated component.

The next picture shows the project structure after component and application creation.

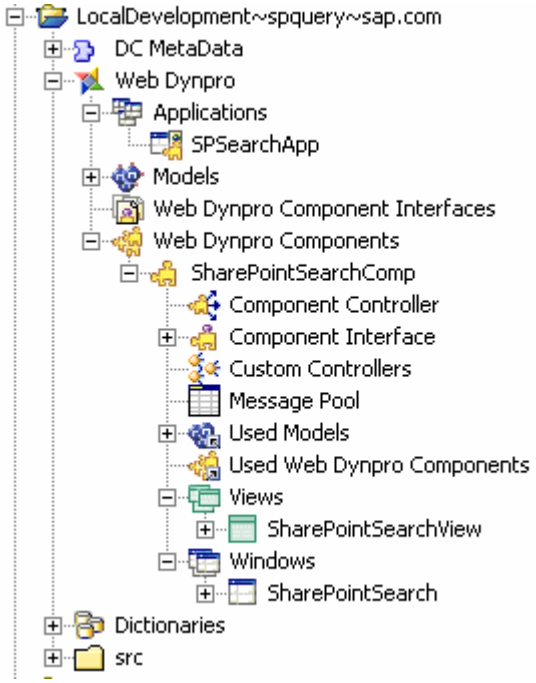

Figure 4 Project Structure

Create the Adaptive Web Services Model

To create the model right click on "Models" in the project structure and select "Import Adaptie Web Service Model".

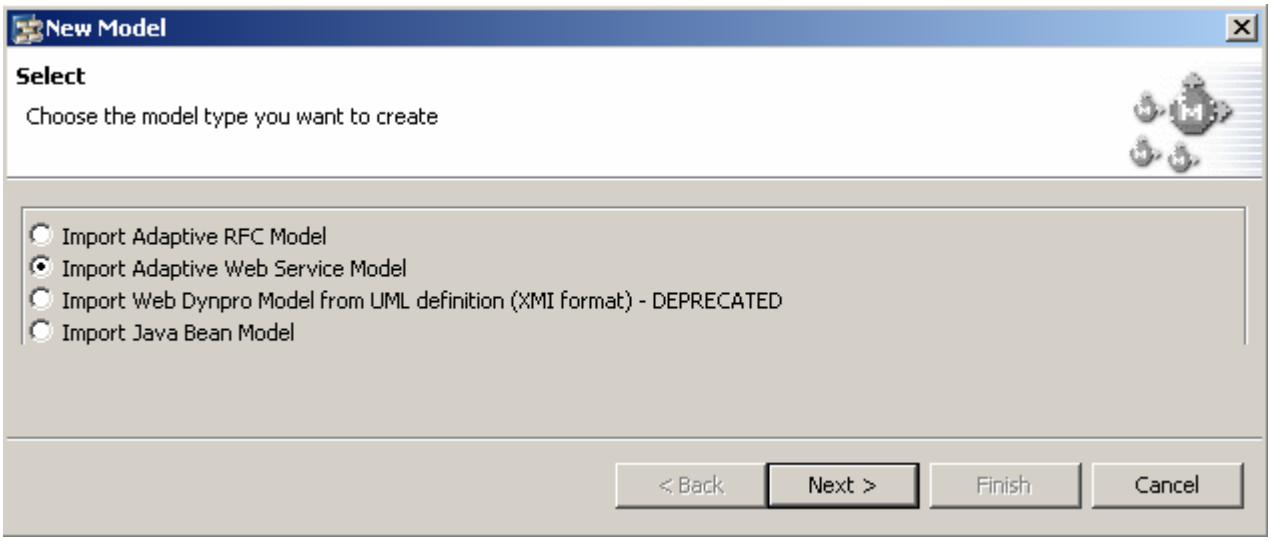

Figure 5 Selecting the Model Type

Enter a name for the model and the package name. For that you can browse the existing ones. Select the previously created namespace and add ".models". Select as WSDL source "Local File System or URL".

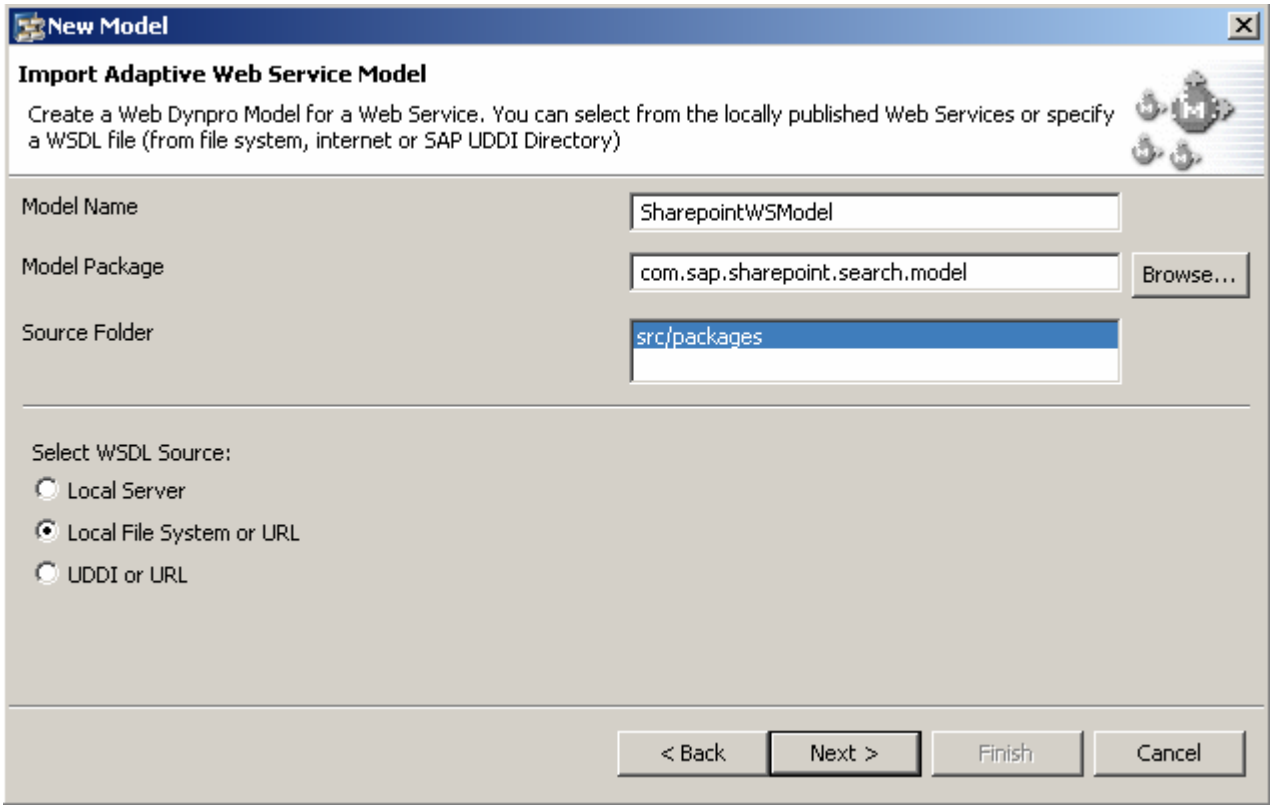

Figure 6 Naming the Model and the Namespace

In the next screen we have to enter the names for the destinations mentioned above. The configuration of the destinations will be done later using Visual Administrator.

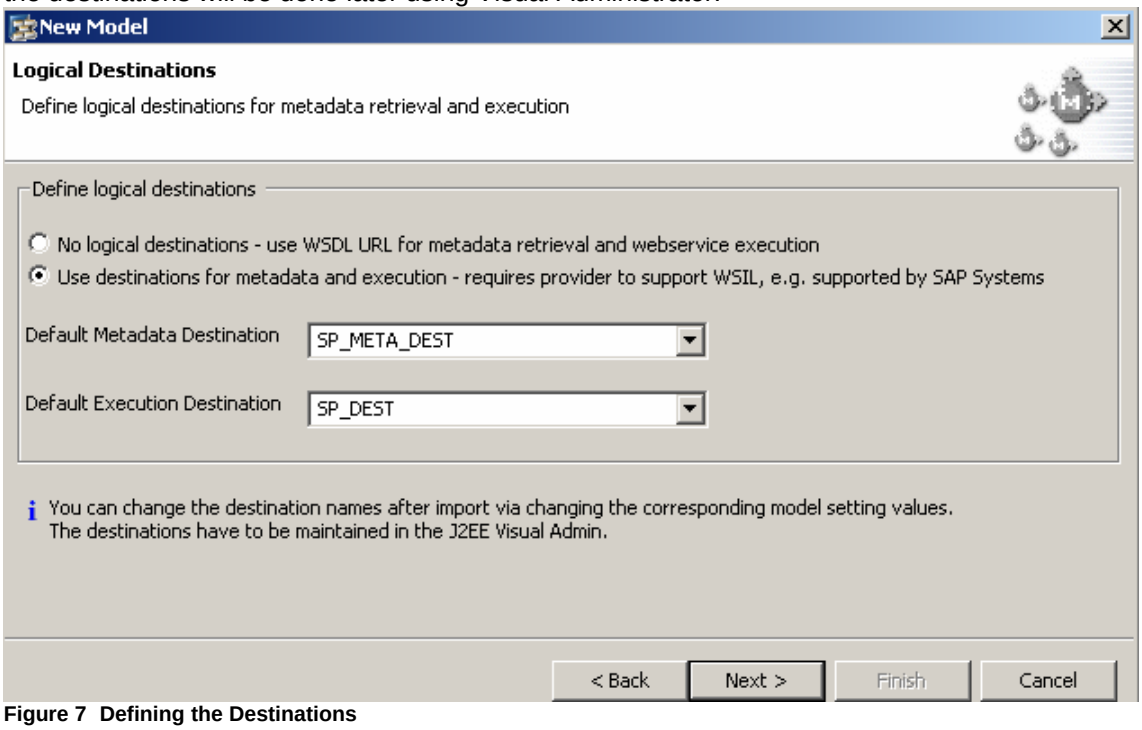

Figure 7 Defining the Destinations

Now we are choosing the newly created WSDL without any tags.

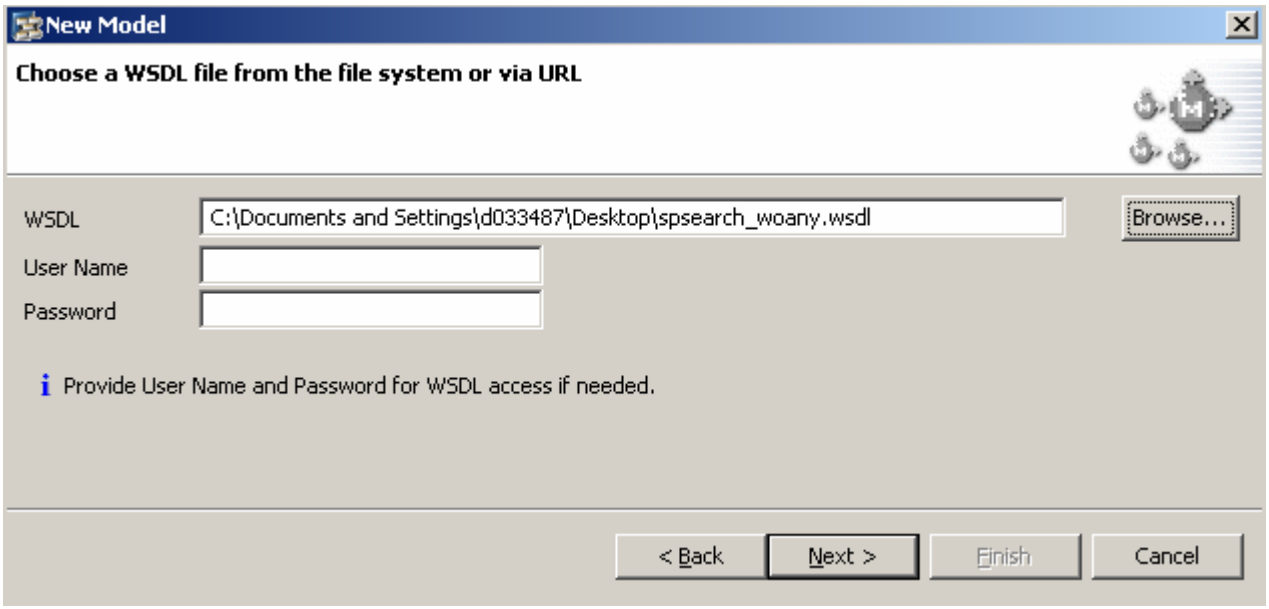

Figure 8 Entering the WSDL Source

On the following screen no renaming of classes is needed. Just click "Finish"

Add the model in the previously created component as used model. In the Project structure we see the generated model classes.

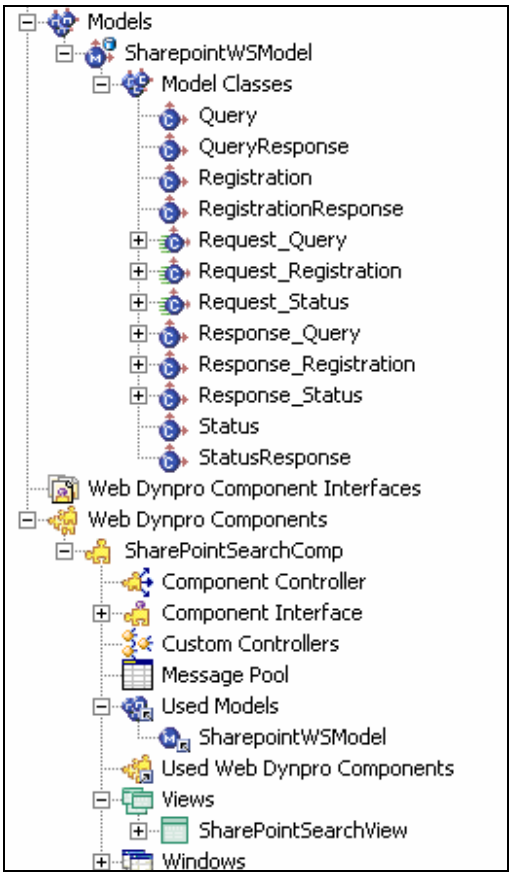

Figure 9 Project Structure with Web Service Model

Calling the Web Service

For calling the Web Service we create a parameterless method in the Component Controller. Right click the component controller to open it and choose the tab methods on the right hand side, where you can create the method.

In the search method the model object for the Web Service is instantiated and executed.

A little bit tricky is the handling of the input and output parameters. There is only one query parameter and one result parameter. Both are xml documents of a certain structure. For details see the SharePoint SDK documentation. So for the query we have to assemble the query xml. This is done by a simple string concatenation.

```
<QueryPacket xmlns='urn:Microsoft.Search.Query'>
  <Query>
    <SupportedFormats>
      <Format>urn:Microsoft.Search.Response.Document.Document</Format>
    </SupportedFormats>
    <Context>
      <QueryText language='en-US' type='STRING'>SharePoint</QueryText>
    </Context>
  </Query>
</QueryPacket>
```

A little bit more effort has to be taken for the parsing of the result that has the structure below. This is done by the method "interpretResult". For parsing the XML structure SAX is used and while parsing the xml data the result is written to the context.

```
<ResponsePacket>
   <Response>
      <QueryID />
      <Copyright />
      <OriginatorContext />
      <RequeryContext />
      <Range>
            <StartAt />
            <Count />
            <TotalAvailable />
            <Results />
      </Range>
      <Status />
      <DebugErrorMessage />
   </Response>
<ResponsePacket>

<Results>
   <Document>…</Document>
</Results>

<Document>
   <Title/>
    <Action>
      <LinkUrl/>
    </Action>
    <Description/>
    <Date/>
</Document>
```

The search method calls the Web Service by instantiating the model object of type Request_Query, setting the query string and calling the method "execute". Note that we have to instantiate the model first in order to pass it to all model classes. We don' t need to bind the model object to a model node because we have to interpret the result before we can use it in the UI.

```
public void search() {
       //@@begin search()
       // assemble the query string
       String queryString = wdContext.currentSearchElement().getSearchString();
       String qXMLString =
              "<QueryPacket xmlns='urn:Microsoft.Search.Query'>"
                     + "<Query><SupportedFormats><Format revision='1'>"
                     + "urn:Microsoft.Search.Response.Document:Document</Format>"
                     + "</SupportedFormats><Context>"
            + "<QueryText language='en-US' type='STRING'>"
                     + queryString
                     + "</QueryText></Context></Query></QueryPacket>";

       SharepointWSModel spSearch = new SharepointWSModel();
       Request_Query modelObject = new Request_Query(spSearch);

       Query query = new Query(spSearch);
       query.setQueryXml(qXMLString);
       modelObject.setQuery(query);

       try {
              modelObject.execute();
              // read the result from the model object
              String result =
                  modelObject.getResponse().getQueryResponse().getQueryResult();
              // parse the xml result
              interpretResult(result);
       } catch (WDWSModelExecuteException e) {
              msgMgr.reportException(e.getNestedLocalizedMessage(), false);
       }
       //@@end

    }
```

Figure 10 The Search Method in the Component Controller

Below you find the SAX parsing code for interpreting the result and filling the context in order to show the results in the UI. The needed class for the SAX parsing SAXTreeStructure was put in the other part of the component controller where own Java code can be placed. Note that we have to work with private member variables like content and document element where the parsing results are saved.

```
public void interpretResult(java.lang.String resultXML) {
	//@@begin interpretResult()
	try {
		// parse the xml return string
		InputStream inputStream = new
java.io.ByteArrayInputStream(resultXML.getBytes("UTF-8"));

		SAXParserFactory factory;
		ClassLoader oldLoader = null;

		oldLoader = Thread.currentThread().getContextClassLoader();
		Thread.currentThread().setContextClassLoader
			(SAXTreeStructure.class.getClassLoader());
		factory = SAXParserFactory.newInstance();

		Thread.currentThread().setContextClassLoader(oldLoader);
		factory.setValidating(true);
		SAXParser parser = factory.newSAXParser();
		parser.parse(inputStream, new SAXTreeStructure());
	} catch (UnsupportedEncodingException e) {
		msgMgr.reportException(e.getLocalizedMessage(), false);
	} catch (FactoryConfigurationError e) {
		msgMgr.reportException(e.getLocalizedMessage(), false);
	} catch (ParserConfigurationException e) {
		msgMgr.reportException(e.getLocalizedMessage(), false);
	} catch (SAXException e) {
		msgMgr.reportException(e.getLocalizedMessage(), false);
	} catch (IOException e) {
		msgMgr.reportException(e.getLocalizedMessage(), false);
	}
	//@@end

    }

//@@begin others
public class SAXTreeStructure extends DefaultHandler {

public void startElement(String namespaceURI,
			String localName,
			String rawName,
			Attributes atts) {
	if (rawName.endsWith("Document")) {
		documentElement =
		    wdContext.nodeDocument().createDocumentElement();
	}
}

public void endElement(String namespaceURI,
			String localName,
			String rawName) {
	try {
		if (rawName.endsWith("Document")) {
			wdContext.nodeDocument().addElement(documentElement);
		} else if (rawName.endsWith("Title")) {
			documentElement.setTitle(content);
		} else if (rawName.endsWith("LinkUrl")) {
			documentElement.setVisibleLink(content);
			// URL encode to replace spaces
			content = URLEncoder.encode(content);
			documentElement.setLink(content);
		} else if (rawName.endsWith("Date")) {
			content = content.replaceAll("\\+", "GMT");
			DateFormat format =
				new SimpleDateFormat("yyyy-MM-dd'T'hh:mm:ssz");
			Date date = format.parse(content);
```

```
                documentElement.setDate(new java.sql.Date(date.getTime()));
            } else if (rawName.endsWith("Description")) {
                documentElement.setDescription(content);
            } else if (rawName.endsWith("TotalAvailable")) {
                wdContext.currentSearchElement().setTotalAvailable(content);
            }
        } catch (ParseException e) {
            msgMgr.reportException("Cannot parse date: " + content, false);
        }
        content = "";
}

public void characters(char[] data, int off, int length) {
        content = new String(data, off, length);
}

}
IWDMessageManager msgMgr;
IPublicSharePointSearchComp.IDocumentElement documentElement;
String content;
//@@end

}
```

Figure 11 The Interpretation of the Result

In order to visualize the search results we need a context structure that can be bound to the UI elements. We create in the context of the component controller a node "Document" with cardinality 0:n for the results that contains all data to be displayed. The elements of the document node and its attributes are created and filled during the SAX parsing process. We don't need a special model node because all processing is done in the search method and the results are not visualized directly.

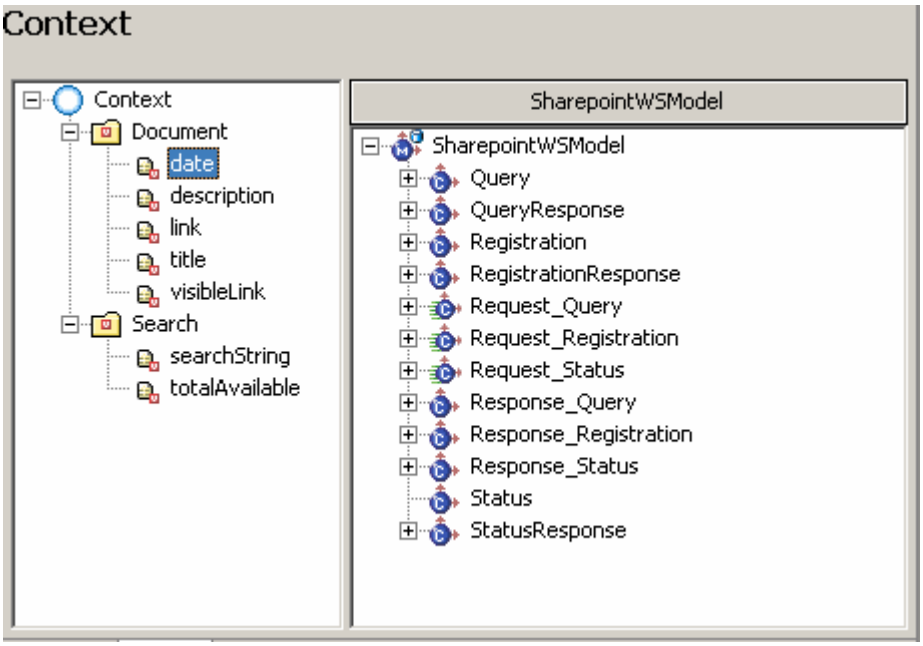

Figure 12 Context of the Component Controller

Creation of the User Interface

Map the context of the component controller to the context of view.

In the Layout the following UI elements are needed:

- An input field for entering the search string
- A text view for showing the number of total available doc
- A table for displaying the results
- A button to perform the search

These UI elements have to be bound to the context in order to transfer/display the values.

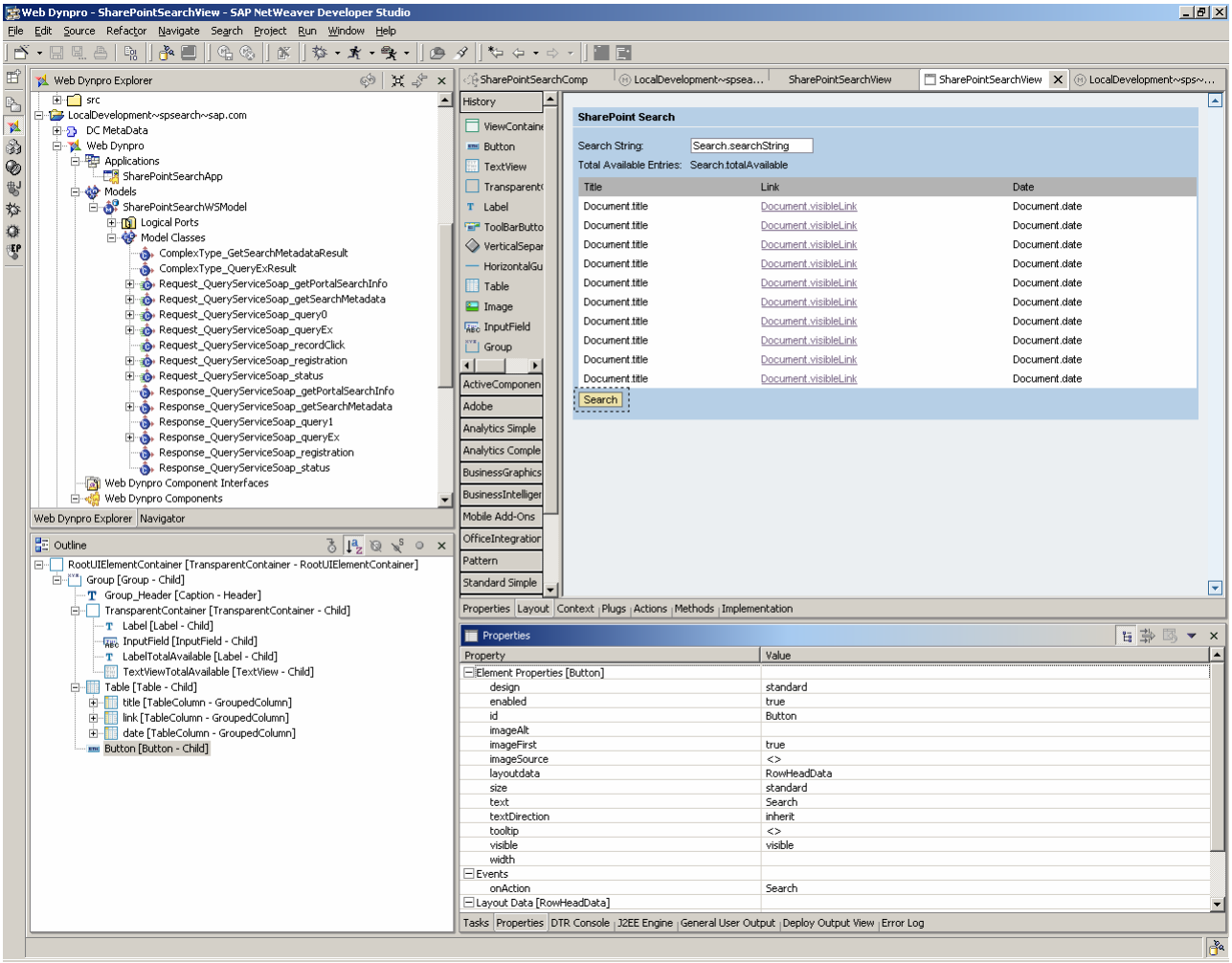

Figure 13 The Layout of the Application

The action Search of the button calls the controller method "Search".

Build and deploy the DC by right clicking the project.

Creating the Destinations

Before running the application the destinations have to be created in the Visual Admin. Go to Web Services Security and create the destinations "SP_META_DEST" and "SP_DEST" below Web Service Clients/sap.com/DynamicWSProxies. We start with basic authentication. In the next chapter we'll show how to work with Logon Tickets.

For the execution destination enter the WSDL URL of the SharePoint Server. Select basic authentication and enter a given user and password.

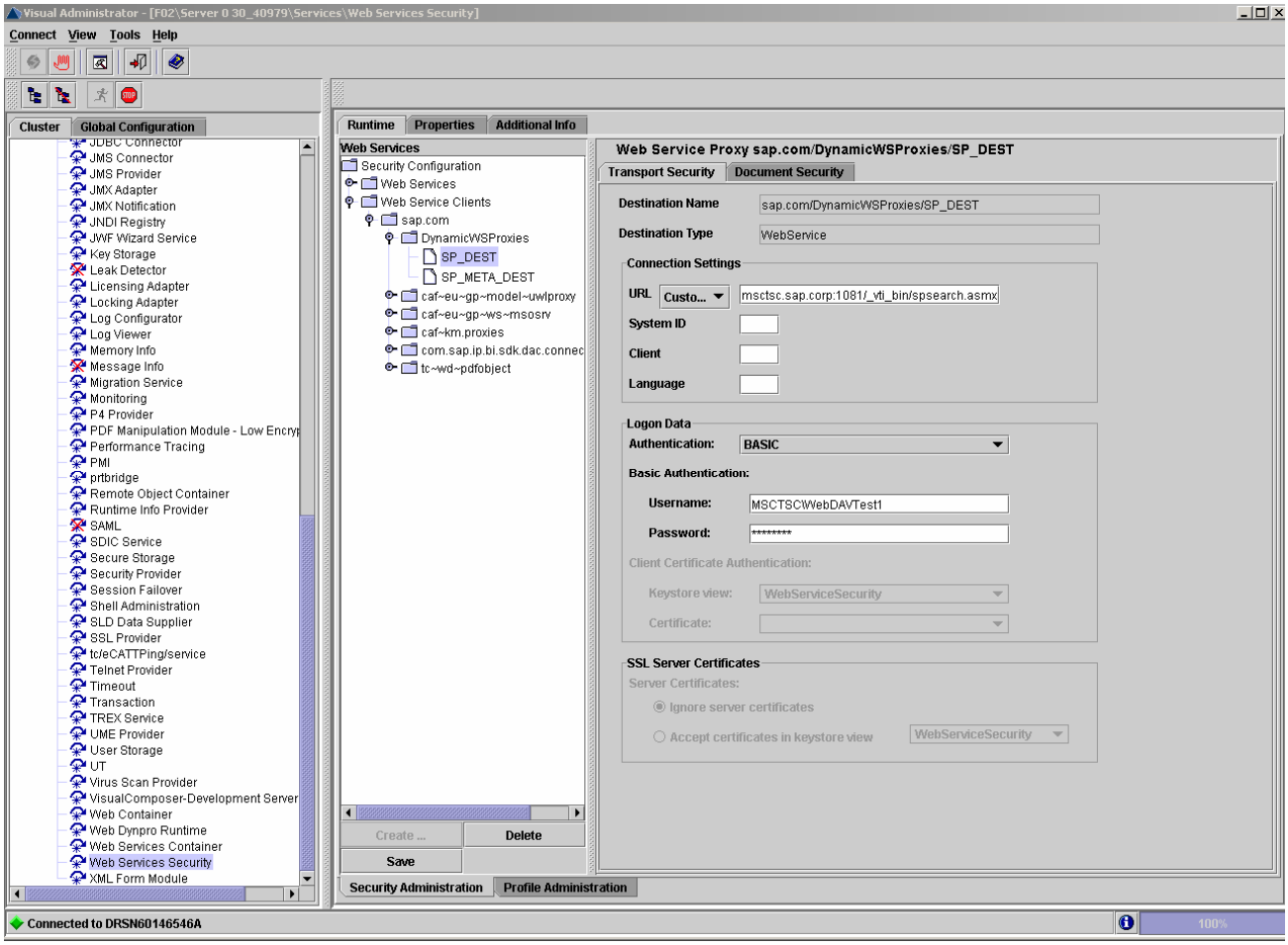

Figure 14 Creation of the Execution Destination

For the metadata enter the URL of the KM where the changed WSDL was stored. For authentication enter a user that has access to the KM document.

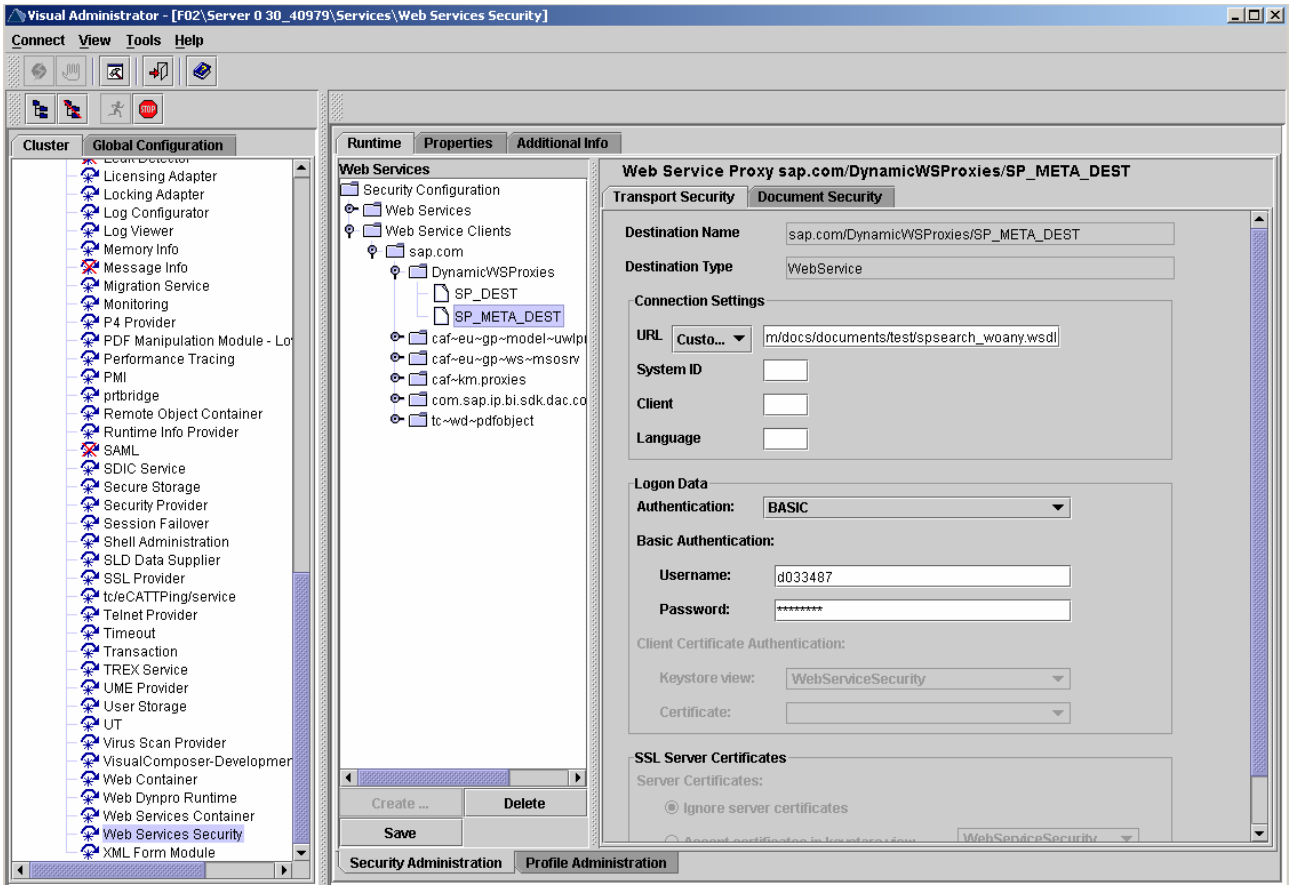

Figure 15 Destination for Metadata

Now our application is ready to run. Right click the application and select "Run".

Authentication and SSO

It is an essential aspect of any sort of interoperability whether a secure and seamless authentication is possible across a WebDynpro based application consuming Microsoft SharePoint based Web Services.

Especially when calling the Search Web Services one would like to be sure that a user that is using the Web Dynpro application does not have access to content he would not be able to see according to his windows credentials.

We face the problem that both SAP NetWeaver Portal and SharePoint reside in two different security domains. While the preferred method of authentication for SharePoint Web Services is Integrated Windows Authentication SAP is currently using SAP Logon Tickets.

Though in the future SAML Tickets will be supported by both vendors a solution is needed for the existing technologies.

Fortunately there is a solution provided by SAP for this: the *SSO22KerbMap Module*.

In order to work with SSO (Single Sign On) some little changes are necessary for our sample application:

1. The application has to work with authentication in order to know for which user to produce the ticket.
2. In the execution destination the authentication approach has to be set to "Logon Ticket".
3. The SharePoint Server has to be prepared for working with SAP Logon tickets

Securing the Application with Authentication

Open the application properties by double clicking the application in NWDS and add a predefined application property "Authentication" with value true.

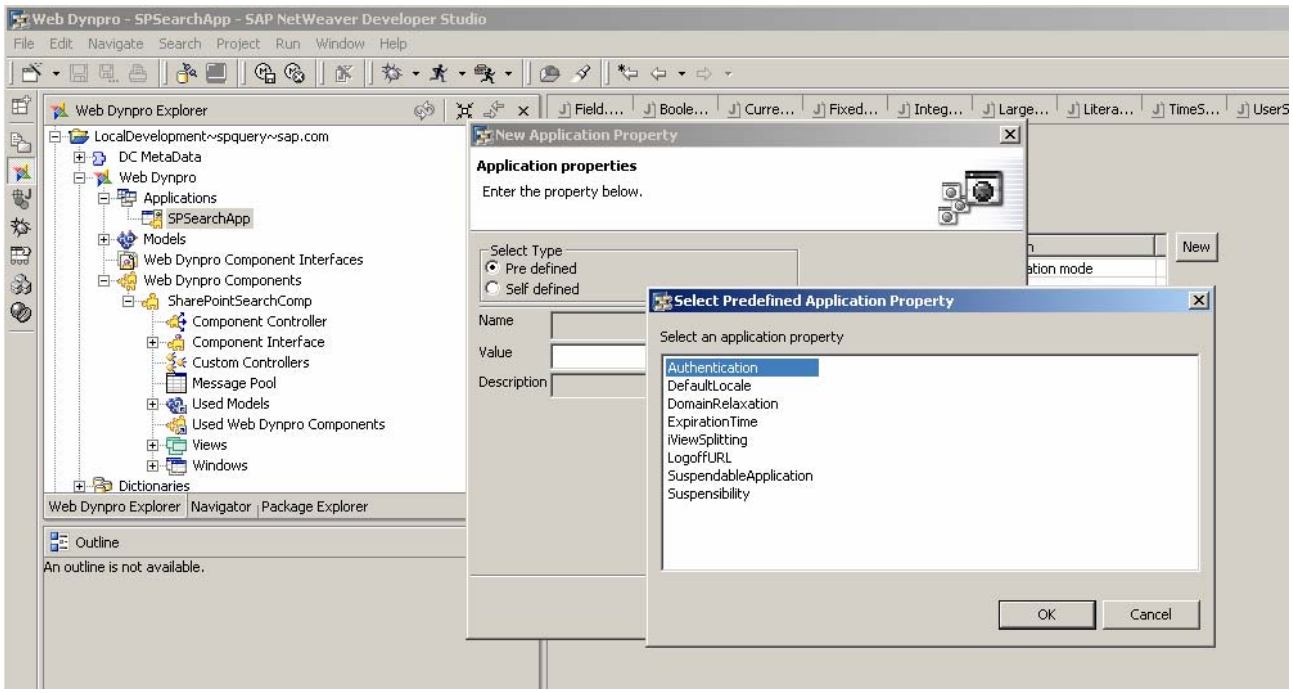

Figure 16 Adding an Application Property for Authentication

Changing the Authentication Mode

The usage of SSO can be easily configured in the execution destination. Just open the previously defined destination "SP_DEST", change the authentication mode in the document security tab to "Logon Ticket" and save the destination.

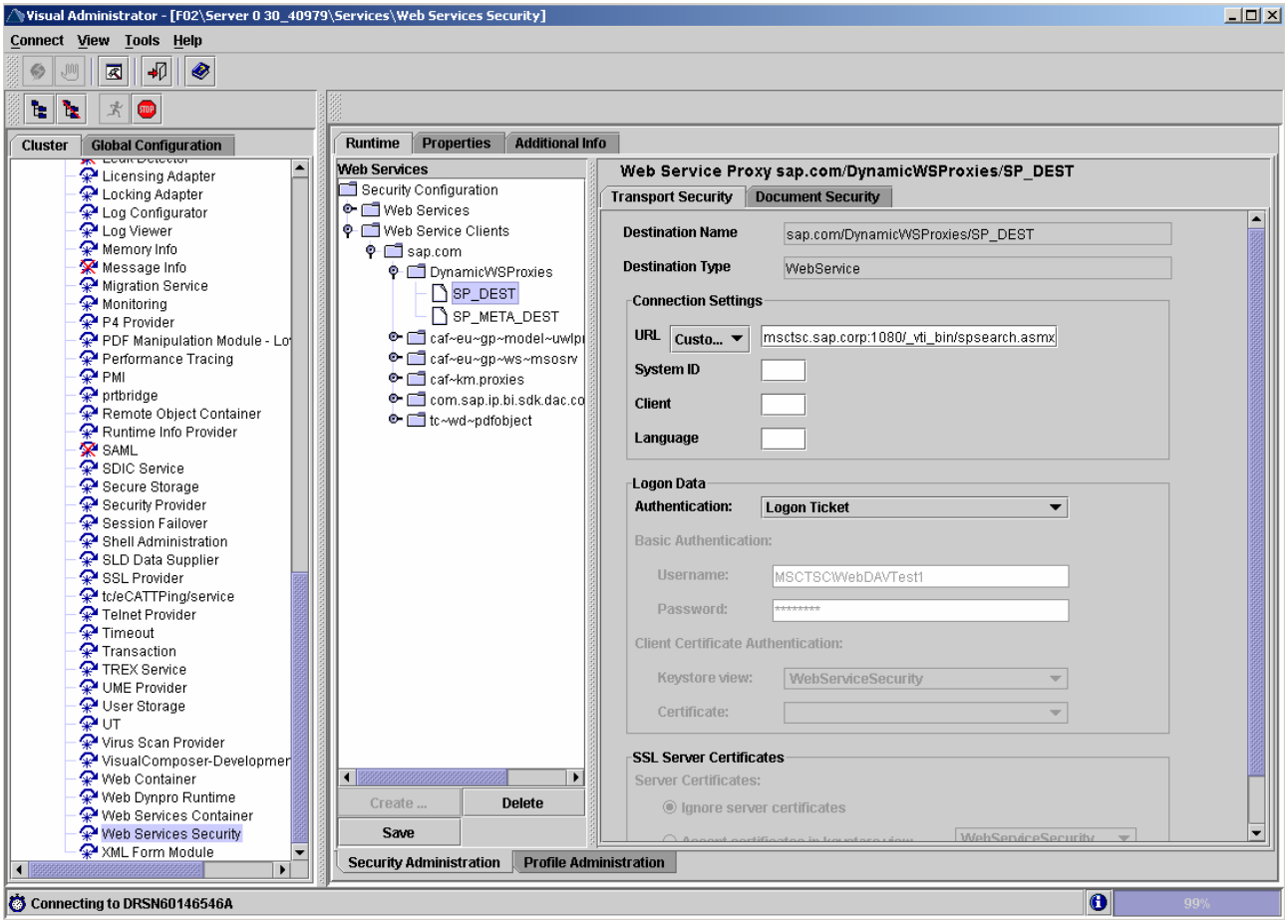

Figure 17 Authentication Mode in Execution Destination

Installing the SSO22KerbMap Module on the SharePoint Server

The installation and configuration of the SSO22KerbMap Module is described in the SDN whitepaper *Using SAP Logon Tickets for Single Sign on to Microsoft based web applications*.

Checking the log

In the Log File that is located in the directory where the SSO22KerbMap.dll is located you will see the following entries showing a successful authentication when calling the Web Service by our application:

```
10:08:20 6876/6880 i OnPreprocHeaders: ----------> Received URL /_vti_bin/spsearch.asmx
10:08:20 6876/6880 i OnPreprocHeaders: Determined account webdavtest1 from cookie
MYSAPSSO2
10:08:20 6876/6880 i OnPreprocHeaders: Running on security context of user NETWORK
SERVICE before impersonation
10:08:20 6876/6880 i LogonAsUser: LsaLookupAuthenticationPackage executed succesfully
10:08:20 6876/6880 i LogonAsUser: LsaLogonUser handle: 10D8
10:08:20 6876/6880 i OnPreprocHeaders: SF_STATUS_REQ_NEXT_NOTIFICATION
```

Related Content

- Joint Technical Reference Guide - Interoperability between SAP NetWeaver Portal and Microsoft SharePoint - available https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/8165
- QueryPacket Element in Microsoft.Search.Query Schema for Enterprise Search http://msdn.microsoft.com/en-us/library/ms545002.aspx
- New Web Dynpro Java Learning Material - Using Adaptive Web Service Models https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/6604
- Using SAP Logon Tickets for Single Sign on to Microsoft based web applications https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/4f209cf3-0201-0010-1db5-d2e33048b6c8

Copyright