Teknoda "**SAPping**" http://www.teknodatips.com.ar

Notas técnicas – Tips de SAP Netweaver ABAP – JAVA

Tip en detalle Nro. 25

(Lo nuevo, lo escondido, o simplemente lo de siempre pero bien explicado)

Los nuevos escenarios de programación con SAP Netweaver (serie de varios tips)

"Cómo consumir un Web Service desde una WebDynpro JAVA"

Tema: Web Dynpro, Netweaver, SAP Netweaver Application Server, WebServices, J2EE

Descripción: El objetivo de esta serie de tips es **recorrer y ejemplificar el nuevo escenario de desarrollo que SAP ofrece a partir de Netweaver**. En este tip, explicamos en detalle los pasos básicos para el consumo de un Web Service desde un aplicativo Web Dynpro JAVA y desarrollamos un ejemplo detallado **paso a paso**.

Nivel: Avanzado

Versión: SAP Netweaver Application Server 6.40 (motor JAVA) en adelante

Fecha pub: Febrero de 2009

NOTA: Para entender este tip es necesario haber leído los tips anteriores de la serie**: "Entendiendo las Web Dynpro: un caso práctico paso a paso"** y **"ESA y Web Services en SAP Netweaver: Introducción"**. Consulte nuestro sitio web para accederlos:

http://www.teknodatips.com.ar/sap-netweaver.html

Tabla de contenido

II. Modelos de datos en el patrón MVC.
III. Modelos de datos en WebDynpro – Web Service como Modelo de Datos.
IV. Caso práctico.
V. Para tener en cuenta
VI. Dónde obtener información adicional

1. Introducción

En los nuevos escenarios de programación con SAP Netweaver, los **Web Services** cumplen un rol preponderante dentro del concepto de eSOA (Enterprise Services Oriented Architecture), y conforman el nuevo paradigma para el desarrollo de aplicaciones en SAP. (Referirse al *Tip Nro. 16 – "**ESA y Web Services en SAP Netweaver: Introducción"***, para comprender la arquitectura eSOA y el importante concepto de Web Services.)

Por otro lado, en este mismo escenario, SAP propone el desarrollo de **Web Dynpros** como la interfaz de usuario Standard (UI) para las aplicaciones, que facilitan el acceso a la funcionalidad provista por SAP, desde un entorno Web. (Se recomienda la lectura del *Tip Nro. 13: **"Entendiendo las Web Dynpro: un caso práctico paso a paso",*** para una mayor comprensión del presente tip).

En este tip seguiremos por medio de un **detallado** ejemplo, todos los pasos necesarios para poder **consumir un Web Service desde un aplicativo Web Dynpro JAVA**. (**Nota:** en la jerga de **Web Services "consumir"** un web service es **utilizarlo** en una aplicación**.)**

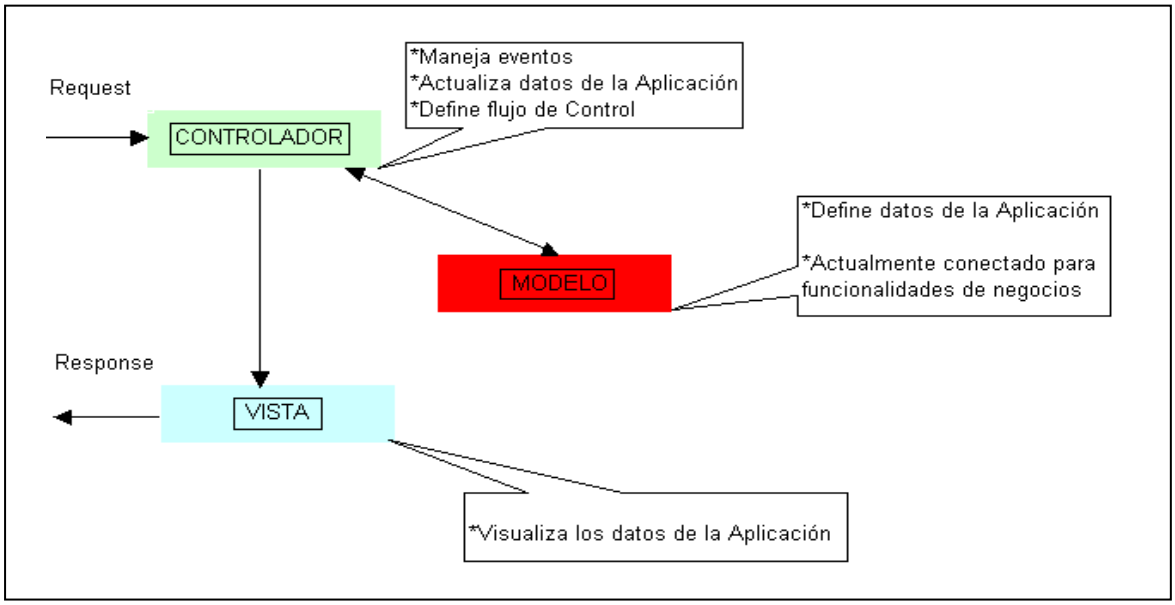
Si bien en este tip nos enfocaremos en el consumo de un web service desde una **Web Dynpro JAVA**, los conceptos aplicados también son válidos para el caso de desarrollar una Web Dynpro ABAP. Una de las diferencias radica en la utilización del **ABAP Workbench** como entorno de desarrollo para el caso **ABAP**, y el **SAP Netweaver Developer Studio** para el desarrollo de una **Web Dynpro JAVA**.

Antes de empezar a especificar los pasos detallados del ejemplo, se hará un repaso de los conceptos claves que abarcan el desarrollo de una Web Dynpro (por ejemplo, el concepto de Model View Controller, utilizado tanto para las WD JAVA como para las WD ABAP), y en particular aquellos que tienen que ver con los **modelos de datos**, debido a que los Web Services son considerados modelos de datos por este framework.

2. Modelos de datos en el patrón MVC

En un tip anterior (Tip Nro. 13: **"Entendiendo las Web Dynpro: un caso práctico paso a paso"**) se han mencionado las características más importantes del patrón MVC (**Model View Controller)** y se han desglosado, además, las distintas partes de las que consta este modelo.

En el presente tip vamos a especificar en detalle el **Modelo**.



En el patrón MVC, los **modelos de datos** son las entidades que representan los datos con los que va a trabajar una aplicación y es una de las capas vitales del patrón, ya que es la responsable de proveer el **contenido completo** en cuanto a los datos propiamente dichos.

En forma genérica estos modelos de datos pueden estar representados por **clases**, **diagramas** o **cualquier tipo de descriptor** que proporcione un perfil de la estructura que tendrán los datos con los que se va a trabajar.

En Web Dynpro los veremos como un componente más dentro de la aplicación, pero que es el encargado de proveernos datos de persistencia y funcionalidad para trabajar sobre ellos. El modelo encapsula la funcionalidad del negocio real. Sirve como fuente de datos para cualquier clase de *visualización* y provee un único punto para actualizar o recuperar información. Esto puede ser utilizando en JAVA a través EJB (enterprise Java Beans), ó ABAP a través de RFC y BAPIs, o como se explica en este tip, **a través de Web Services.**

3. Modelos de datos en WebDynpro – Web Service como Modelo de Datos.

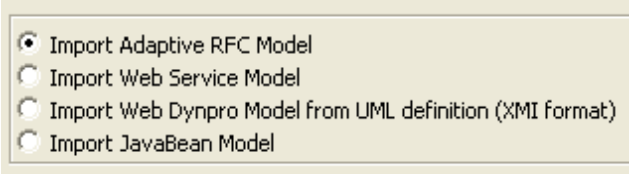
En el caso de WebDynpro se cuenta con varias posibilidades para el uso de modelos de datos.

Como se ha visto en el *Tip en detalle Nro. 13: "Entendiendo las Web Dynpro: un caso práctico paso a paso"*, el framework está volcado hacia el desarrollo visual y en ese sentido se dispone de una representación visual para el uso de los modelos de datos que se agreguen en un componente dado.

Lo interesante de este enfoque es que se podrán incorporar distintos tipos de modelos de datos, y se podrá manejarlos en forma análoga en todos los casos. Se verá en el ejemplo, que al incluir un modelo de datos en un componente, lo que se hace es incorporar un nuevo elemento del tipo "caja negra" que tiene puntos de entrada y puntos de salida con los cuales trabajar, pero quedan encapsulados los detalles del tipo de modelo que se esté usando.

De esta manera se podrá hacer un uso muy parecido de un modelo de datos generado a partir de un WebService o desde una RFC a través del SAP Enterprise Connector.

A continuación mostramos el menú que se despliega al crear un modelo dentro de un componente de WebDynpro. Se pueden ver las distintas opciones que éste ofrece, en lo siguiente:

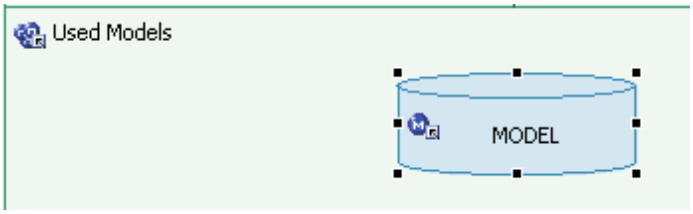


En primer lugar se encuentra **Adaptive RFC**, que es en general bastante utilizado porque utiliza el SAP Enterprise Connector como base para encapsular en varias clases proxy, una conexión JCo (Java Connector) para poder realizar una llamada a una RFC en un sistema SAP ERP.

Luego la opción que nos ocupa en este tip, **el Web Service Model**. También, en este caso como en los otros, se generan clases que encapsulan la llamada, pero aquí particularmente, la tecnología subyacente es HTTP, SOAP y XML.

Las últimas dos opciones que se muestran son para la generación de un modelo de datos a partir de una definición UML, para lo cual se utilizan archivos en formato XMI, y los conocidos JavaBeans, (que son clases de Java con atributos miembro, que representan "Objetos", lo cual obviamente puede usarse como perfil de un modelo de datos). La funcionalidad que se le agregue a estos objetos también podrá ser accedida a través del modelo.

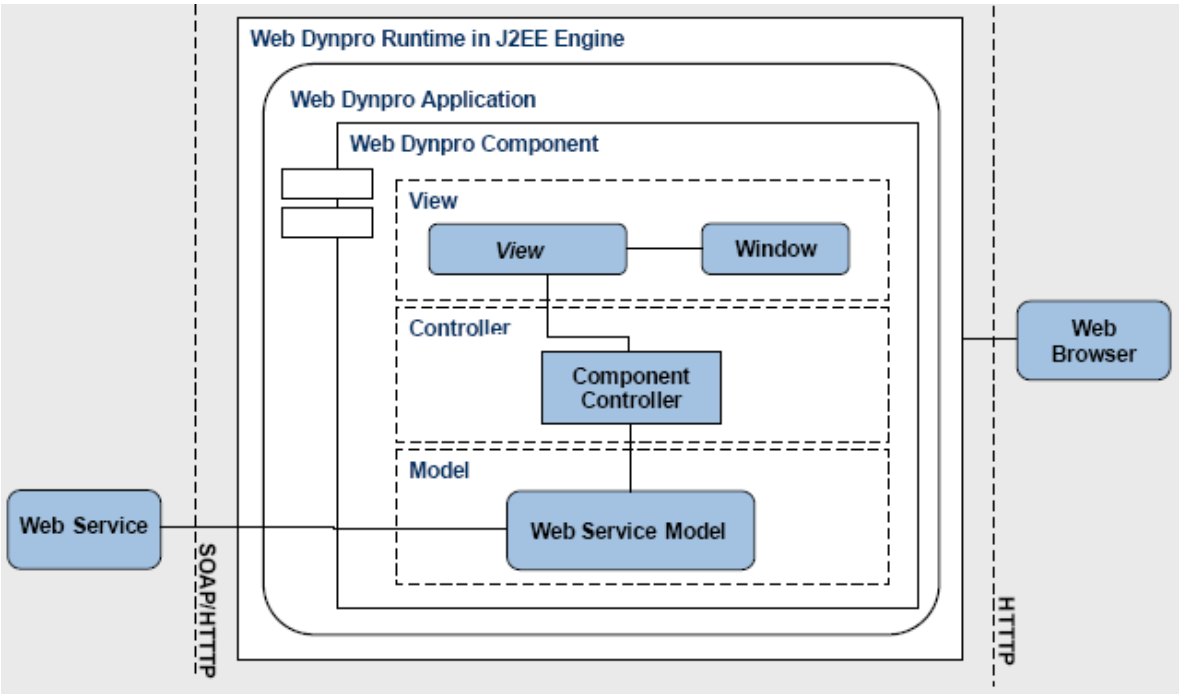
Como mencionamos anteriormente, lo más interesante de la creación del modelo, es que **no importa cuál de estas opciones se elija**, el framework permitirá manejarlos en forma análoga. Veremos a continuación lo que el framework genera en forma gráfica para trabajar con el modelo.



Este modelo gráfico encapsula todos los datos y funciones que modifican datos persistentes en un sólo modelo. Nuestro trabajo será simplemente enlazar este componente con los **controladores** correspondientes, cuyo trabajo consistirá en consumir los datos y funciones que este modelo provea.

Por ejemplo, desde el **Component Controller,** podremos tomar el contexto que representa este modelo, allí asignar valores a los parámetros de entrada de los servicios que el modelo preste, y luego ejecutar las llamadas para finalmente leer los resultados. Con ésto estaremos abarcando dos capas importantes del patrón MVC: **Model** y **Controller**, y además, para poder desplegar la información en pantalla, tendremos que incorporar la tercera capa denominada **View** que es la encargada de la interfaz visual.

En el siguiente esquema se puede ver la estructura típica del escenario que se acaba de describir. Se cuenta con un **web service a consumir** y se lo consume a través del modelo generado en el framework de Web Dynpro. Una vez generado el modelo, quedan encapsulados los detalles de la llamada propia al Web Service. Luego, el manejo queda supeditado al **Component Controller** que deberá poblar su contexto y realizar la llamada al servicio con métodos automáticamente generados por el entorno de desarrollo.



4. Caso práctico

El caso práctico que desarrollaremos ejemplifica **cómo se utiliza un WebService en una Web Dynpro JAVA**. Para este ejemplo tomaremos un Web Service de la web, en este caso de la dirección http://coeservice.en.kku.ac.th/samples/. En este sitio, elegimos, como ejemplo, un web service que va a permitir realizar **la conversión de un valor de temperatura dado en grados Celsius a grados Farenheit**. En dicha dirección de Web, hay otros Web Services disponibles, que también se podrían utilizar.

Para el **desarrollo de la aplicación Web Dynpro JAVA** completa que haga uso de web services como modelo de datos, es necesaria la ejecución de los siguientes pasos, que van a ser desarrollados más abajo en este tip: :

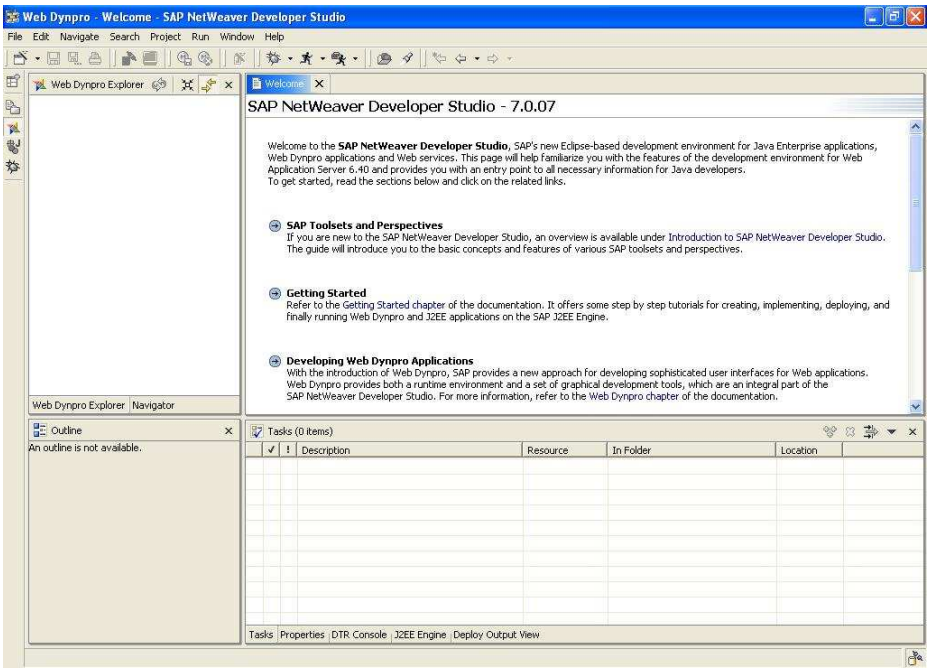
- Pasos para la creación de un proyecto Web Dynpro
- Pasos para la creación de una componente Web Dynpro
- Pasos para la creación de un modelo de datos
- Pasos para la creación de una o más vistas
- 1era. Parte: Pasos para la creación de "inbound" y "outbound plugs"
- 2da. Parte: Pasos para la conexión de los "plugs" por el uso de links de navegación
- Pasos para la creación de actions (acciones) e implementación de la navegación
- Pasos para la estructura de los contextos
- Pasos para la realización del diseño de la vista *FormularioView*
- Pasos para la realización del diseño de la vista *ResultadoView*
- Pasos para la implementación de la lógica de negocio en el Component Controller
- Pasos para agregar la llamada al Servicio en FormularioView
- Pasos para la creación de una Web Dynpro Application
- Ejecución de una Web Dynpro Application

La aplicación desarrollada se llamará: **"ConsumoWebServiceWD**" y va a estar compuesta por **2 vistas**. Al ser ejecutada la aplicación, la primera vista debería tener el siguiente aspecto:

Y el de la segunda vista, como sigue:

Para comenzar a construir una Web Dynpro, lo primero que debemos hacer es abrir el producto cliente **SAP Netweaver Developer Studio (**el entorno de desarrollo para JAVA**)**, que debemos tenerlo instalado en nuestro puesto de trabajo (PC).

Su ruta de acceso **es** ***Inicio → Programas → SAP Netweaver Developer Studio → SAP Netweaver Developer Studio*** y realizamos doble click sobre la misma. Luego, aparecerá una pantalla como la siguiente:



A partir de aquí, lo primero que se necesita es seleccionar la **perspectiva** con la cual vamos a trabajar. Para ello se elige desde el menú principal: ***Window → open Perspective → Web Dynpro***. Luego crearemos el **Proyecto** que contendrá la Web Dynpro a desarrollar.

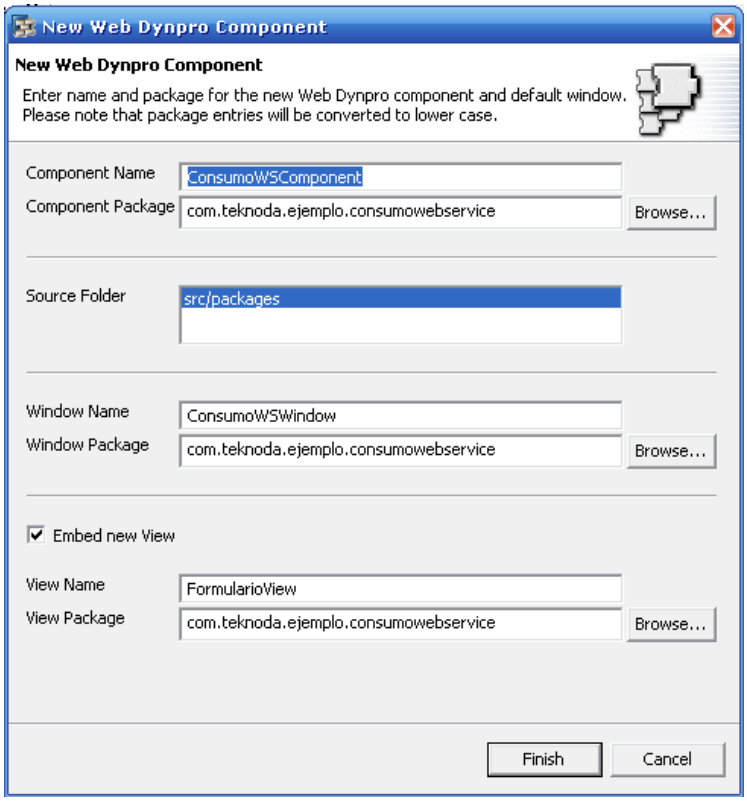
Pasos para la creación de un proyecto Web Dynpro

1. Seleccionar ***File → New → Project***.
2. Aparece el asistente de ***New Project***
3. En el panel izquierdo seleccionar ***Web Dynpro*** y luego en el panel derecho ***Web Dynpro Project*** y presionar ***Next***.
4. En la siguiente pantalla de propiedades del proyecto, colocar el nombre del proyecto, que llamaremos **ConsumoWebServiceWD**, y dejar las especificaciones por defecto del ***Project contents*** y el ***Project language***.

Con los 4 pasos realizados previamente, se ha creado la **estructura del proyecto**, pero aún no se han definido las vistas y su layout, los controladores, ni los eventos. Para ello, debemos definir previamente una **componente Web Dynpro** que los encapsule.

Pasos para la creación de una componente Web Dynpro

1. Expandir el nodo ***Web Dynpro*** y posicionarse sobre ***Web Dynpro Components***. Presionar Botón derecho del mouse y seleccionar del menú contextual: ***Create Web Dynpro Component***
2. Se visualiza, entonces, el asistente para **"New WebDynpro Component "**. Luego, se debe ingresar el nombre del *Component Name:* **ConsumoWSComponent** y especificar el paquete (tal como **com.teknoda.ejemplo.consumowebservice**) .



3. Corregir ConsumoWSComponent en la caja de texto de W*indow name* por el nombre **ConsumoWSWindow.**

4. Posteriormente, se debe corregir ConsumoWSComponentView en la caja de texto de V*iew name* por el nombre de la primera vista a crear llamada **FormularioView**.

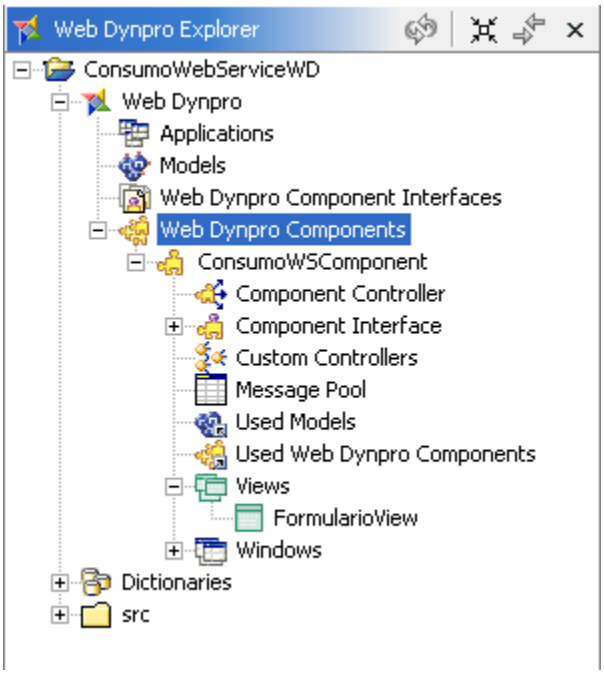
5. Dejar todas las especificaciones por defecto que aparezcan. Cerciorarse que se encuentre tildado "**Embed New View"**.

6. Presionar ***Finish***.

7. Presionar el dibujo del "diskette" que se encuentra en la barra de herramientas, para que el proyecto quede guardado.

Una vez realizados estos pasos, en el panel derecho, se desplagará automáticamente el ***Data Modeler view***, (que no va a ser usado por el momento), mientras que a la izquierda de la pantalla, en el **Web Dynpro Explorer**, al expandir el nodo **Web Dynpro Components** se podrá visualizar la nueva componente creada con nombre: **ConsumoWSComponent**..

La figura siguiente muestra cómo quedaría el panel **Web Dynpro Explorer**:

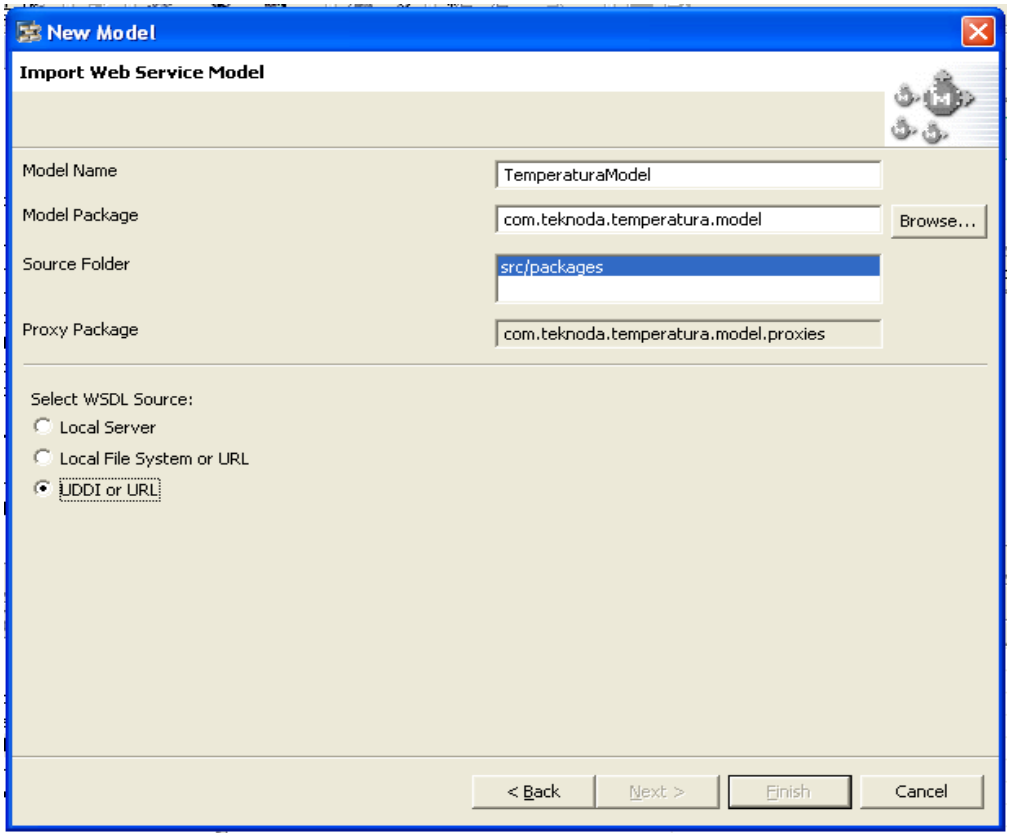


Pasos para la creación de un modelo de datos

1. Expandir el nodo ***Web Dynpro*** en el panel Web Dynpro Explorer .
2. Presionar botón derecho del Mouse sobre el nodo **Models**.
3. Hacer "click" sobre "**Create Model**".
4. Seleccionar el radio button correspondiente a "**Import Web Service Model**" y luego **Next.**
5. Luego, como se muestra en la figura de más abajo, se debe ingresar un nombre para el modelo de datos,. En este caso ingresamos el nombre "**TemperaturaModel"**
6. Como package del modelo ingresar "**com.teknoda.temperatura.model**".
7. Es importante seleccionar la fuente de generación del Web Service, en este caso se debe seleccionar el radio button correspondiente a **"UDDI or URL"**, ya que lo que se hará es ingresar una dirección donde está alojado el web service a utilizar.

Nota: En general se utiliza la dirección donde está el descriptor WSDL para importar un Web Service, pero como se verá en la sección **"Para tener en cuenta …"** (especificado más abajo en este tip) , se puede contar con el archivo WSDL localmente y con él generar el modelo de datos en la aplicación.

8. Presionar **Next.**



9. Posteriormente, en la pantalla que se obtiene, ingresar la **dirección del WSDL** correspondiente al Web Service a utilizar, en este caso :

http://coeservice.en.kku.ac.th:8080/TemperatureConvertor/TemperatureConvertorService?WSDL

10. Luego de presionar **Next**, se presiona **Finish**, y quedará creado el modelo de datos.

Lo que resta es incorporarlo en el componente WebDynpro creado en pasos anteriores.

11. Para ello, desplegar el nodo **WebDynpro Components → ConsumoWSComponent → Used Models**. Presionar botón derecho del Mouse sobre el mismo y seleccionar **Add…** . Luego, seleccionar **TemperaturaModel** y, finalmente **OK**.

Pasos para la creación de una o más vistas

1. En el panel Web Dynpro Explorer, expandir el nodo ***Web Dynpro → Web Dynpro Components → ConsumoWSComponent → Windows***.
2. Realizar doble click en el nodo del objeto ***Windows*** creado anteriormente con nombre **ConsumoWSComponent**, de esta forma en el panel derecho se visualiza el ***Diagram View.***
3. Seleccionar el ícono ***Embed View*** de la paleta que se encuentra a la izquierda de la ventana del ***Diagram View.*** Se realiza un click en el ícono, luego el mouse se ubica en el panel del diseño y se "estira" para formar un área rectangular de un tamaño considerable.

4. Aparecerá el asistente, se deberá elegir la opción ***Embed new View*** y presionar el botón ***Next***.
5. Ingresar el nombre para la nueva vista **ResultadoView** y presionar el botón ***Finish.***

Luego, una vez definidas las dos vistas, se debe indicar la **navegación entre las mismas**. Por lo tanto, se deberán crear **puntos de entrada y salida para cada vista**, habilitando conectores de entradas y salidas ("inbound plugs" y "outbound plugs"). Los "inbound plugs" definen los posibles puntos de entrada de una vista, mientras que, los "outbound plugs" brindan la posibilidad de navegar hacia otra vista. Los "plugs" forman parte del **controlador de la vista** y se asignan a una sola de ellas cada vez.

Consideraciones importantes sobre las Vistas y los "plugs":

- Cada vista tiene que tener al menos un inbound plug mientras que el outbound plug es opcional.
- En general, varias vistas se encuentran embebidas en un objeto *Windows*, por lo tanto hay que determinar a través de la propiedad StartView cuál va a ser la primera vista que se despliegue en el Browser. La estructura de navegación partirá de esta vista.
- Cada vez que se coloca un *inbound plug* se genera automáticamente un método que maneja dicho evento en el fuente java que representa a la vista

La secuencia de pasos para definir la navegación está dividida en dos partes. La **primera parte** se refiere a la creación de los " plugs" y la **segunda parte** para conectar a los links de navegación.

1º Parte: Pasos para la creación de "inbound " y "outbound plugs"

1. Dentro del ***Diagram View***, se selecciona el rectángulo que representa la primera vista, **FormularioView**, con un click derecho del mouse, se despliega el menú contextual.
2. Se selecciona la opción ***Create Outbound Plug***.
3. Aparece el asistente, se ingresa el nombre del "outbound plug " **AResultadoView** y se presiona ***Finish.***
4. Se selecciona el rectángulo que representa la segunda vista, **ResultadoView**, con un click derecho del mouse, se despliega el menú contextual.
5. Se selecciona la opción ***Create Inbound Plug***.
6. Se ingresa el nombre del "inbound plug" **DesdeFormularioView**, se dejan las especificaciones por defecto para el manejador de eventos y se presiona ***Finish.***
7. Se repiten los pasos apropiadamente para la creación del "outbound plug" **AformularioView** para la vista ResultadoView y el "inbound plug" **DesdeResultadoView** para la vista FormularioView.

Tabla indicadora de las correspondencias entre "plugs" y "Vistas"

	outbound plug	**inbound plug**
FormularioView	AResultadoView	DesdeResultadoView
ResultadoView	AformularioView	DesdeFormularioView

2 º Parte: Pasos para la conexión de los "plugs" por el uso de links de navegación

1. Para la creación del link de navegación de la primera vista, se selecciona el ícono ***Create Link*** desde la paleta que se encuentra a la izquierda de la ventana ***Diagram View*** y se traza la línea desde el "outbound plug "de **FormularioView** al "inbound plug" de **ResultadoView**.
2. De manera similar, se crean los links de navegación desde la segunda vista **ResultadoView** hacia la primera vista **FormularioView**.

Si se expande el árbol perteneciente al Web Dynpro Explorer (panel izquierdo de la perspectiva), desde Windows → Primera Component, al abrir este último nodo, se verán las dos vistas **FormularioView** y **ResultadoView.** Al realizar doble click sobre, por ejemplo: **FormularioView**, se despliega una ventana llamada ***Editor View*** inmediatamente debajo de la ventana de ***Diagram View.*** Esta ventana presenta 5 solapas. Al seleccionar la solapa ***Implementation***, se observa el código de java generado para dicha vista. De manera automática, en dicho código veremos implementados los métodos correspondientes a cada "inbound plug", que se llaman **onPlug<nombre del plug>.**

Solapa ***Implementation*** de la vista **FormularioView**

```
public void onPlugDesdeResultadoView
(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent)
 {
    //@@begin onPlugDesdeResultadoView(ServerEvent)
    //@@end
  }
```

Solapa ***Implementation*** de la vista **ResultadoView**

```
public void onPlugDesdeFormularioView
(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvt wdEvent )
  {
    //@@begin onPlugDesdeFormularioView(ServerEvent)


    //@@end
  }
```

Para navegar de una vista a otra, se necesita una acción apropiada que permita relacionar un elemento de interfaz de usuario (por ejemplo un botón) con el traslado a otra vista. En consecuencia, se necesita implementar un método que actúe como un "manejador de eventos" y "reaccione" ante tal acción (por ej: presionar un botón), ejecutándose luego el cambio correspondiente.

Pasos para la creación de actions (acciones) e implementación de la navegación

1. Dentro del ***EditorView*** de **FormularioView**, seleccionar la solapa ***Actions*.**
2. Presionar el botón ***New*.**

3. Aparece el asistente para crear una nueva acción. Luego, ingresar el nombre, denominado en este caso **Traducir,** para la nueva acción, y dejar las opciones del ***Event handler*** por default.
4. Asignar al campo *Fire plug*, el "Outbound plug " **AResultadoView**, y presionar ***Finish.*** La nueva acción, **Traducir**, se encuentra asociada con el método que actúa como manejador de eventos **onActionEnviar**, que se puede observar en la lista de acciones.
5. Se repiten los mismos pasos para la acción **Volver** perteneciente a la vista **ResultadoView** y se asigna al campo *Fire plug*, el "Outbound plug " **AFormularioView**.
6. Luego, guardar la nueva metadata completa eligiendo el ícono ***Save All Metadata*** desde la barra de herramientas.

Finalmente, después de haber creado las 2 acciones: **Enviar** y **Volver,** se tendrán implementados los métodos relacionados al manejo de eventos de cada acción. Para poder visualizarlos, se debe seleccionar la solapa ***Implementation*** de ambas vistas.

Solapa *Implementation* de la vista **FormularioView**

```
public void onActionTraducir
(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent)
  {
    //@@begin onActionEnviar(ServerEvent)
    wdThis.wdFirePlugAResultadoView();
    //@@end
  }
```

Solapa *Implementation* de la vista **ResultadoView**

```
public void onActionVolver
(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
  {
    //@@begin onActionVolver(ServerEvent)
    wdThis.wdFirePlugAFormularioView();
    //@@end
  }
```

Arriba se puede observar cómo se llama al método ***wdFirePlugToResultView()*** perteneciente al "outbound plug". Para la ejecución de dicho método, se utiliza la variable privada ***wdThis*** que pertenece a la Interface *IPrivateResultadoView*. La misma se utiliza siempre que se necesite un método que se comunique internamente al ViewController (el concepto del "View Controller" será desarrollado posteriormente).

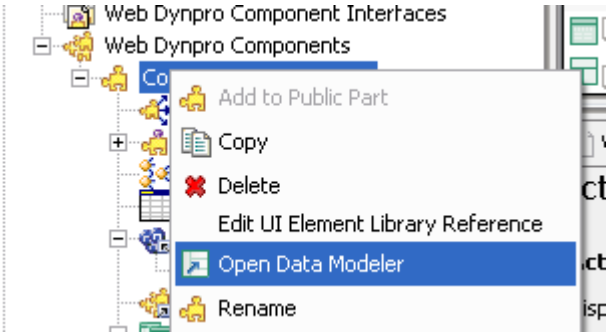
Una vez que se tienen definidas las acciones se puede empezar a trabajar con el **"layout " de la vista.**

Para comenzar, se deberá seleccionar la solapa ***Layout*** en el *EditorView*, donde se muestra por ej: la vista **FormularioView** representada por una caja de texto predefinida de color gris (*predefined default text*). Simultáneamente, si se observa el *Outline* view (panel izquierdo zona inferior) desplegará una lista de todos los elementos de la interfaz de usuario que se vayan incluyendo. Todos los elementos de la interfaz de usuario están acomodados bajo el nodo raíz y representados en formato de árbol. Si se realiza un click derecho sobre

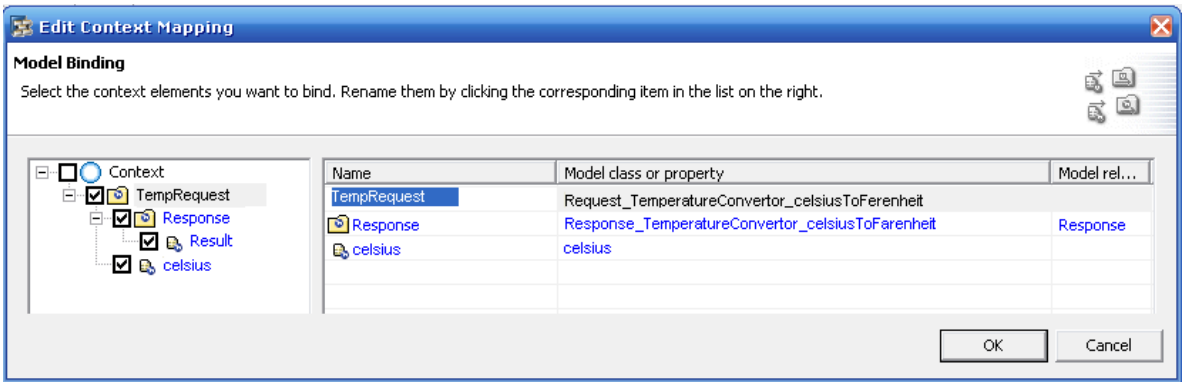
un elemento de UI que se encuentra en el árbol y se selecciona *Properties* del menu contextual, se puede visualizar las propiedades de dicho elemento UI, a través de la ventana *Properties* view .

Pasos para la estructura de los contextos

1. Abrir el *Data Modeler* desde el menú contextual de **ConsumoWSCompoment**.

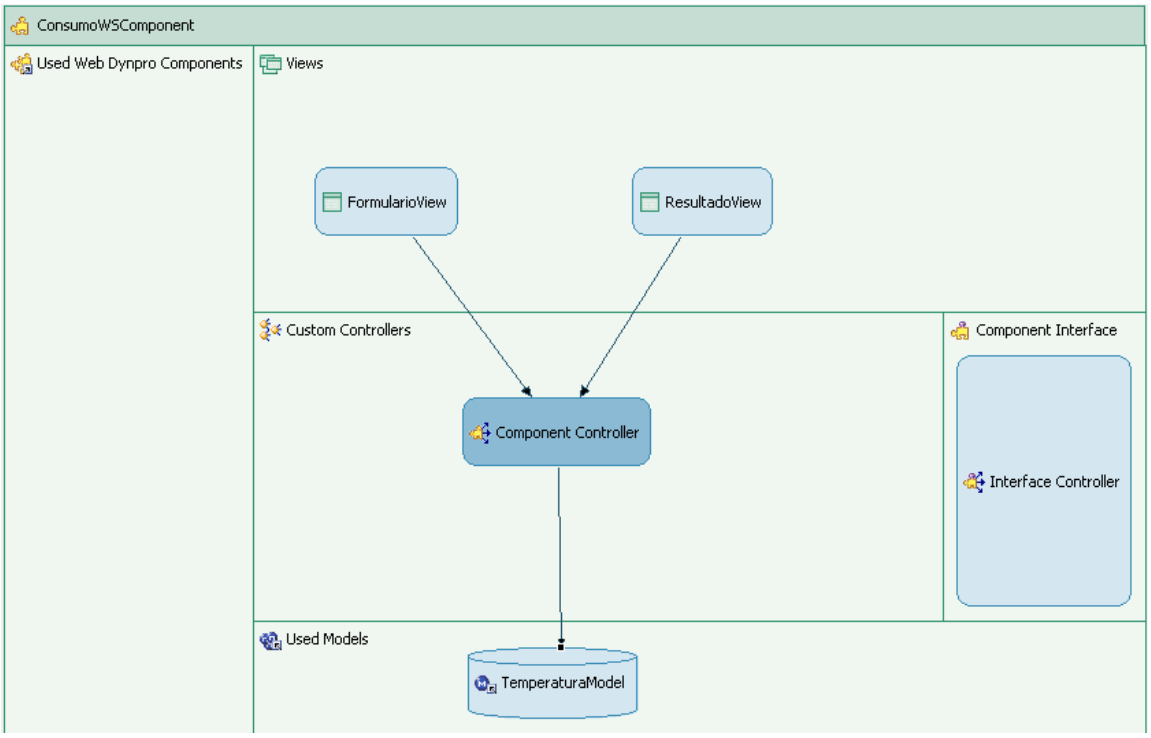


2. Se compartirán los contextos del modelo y del Component Controller, para lo cual se deberá crear un data link entre ellos.
3. Arrastrar **Request_TemperatureConvertor_celsiusToFerenheit** desde el modelo (derecha) hacia el Component Controller (izquierda).
4. Seleccionar la estructura completa del nodo que recientemente se "arrastró", para ser incluida en el contexto del Component Controller.
5. Renombrar el nodo **Request_TemperatureConvertor_celsiusToFerenheit** a **TempRequest** para facilitar el posterior manejo en el código de dicho nodo. Pulsar **OK**, y luego **Finish.**



6. Luego, se realiza lo propio con los contextos de ambas vistas. Se debe crear un data link entre cada una de las vistas y el Component Controller, arrastrando en cada caso todos los elementos del contexto de dicho controlador.

7. El resultado será el siguiente:



8. Realizar "click" en **Save All Metadata.**

Pasos para la realización del diseño de la vista FormularioView

1. Desde la ventana *Outline* view, se elige el nodo raíz **RootUIElementContainer** y se especificacn las propiedades tal como indica la siguiente tabla:

Property	Value
Layout	GridLayout
CellPadding	5
ColCount	1

Nota: No todos los valores de la propiedades son editables, en algunas se debe seleccionar un valor predefinido desde una lista desplegable.

Se elige el nodo hijo **DefaultTextView** y se elimina el mismo:

2. Desde la ventana *Outline* view, se selecciona el elemento **RootUIElementContainer** y desde el menú contextual se elige la opción *Insert Child.* Seleccionar Group. Asignarle el nombre FormGroup, y establecer los siguientes valores a sus propiedades:

FormGroup

Property	Value
layout	GridLayout
colCount	1

3. Desde la ventana *Outline* view, se selecciona el elemento **FormGroup** y desde el menú contextual elegir la opción *Insert Child.* Seleccionar **TransparentContainer** y asignarle el nombre FormContainer
4. Desde la ventana *Outline* view, se selecciona el elemento **FormContainer** y desde el menú contextual elegir la opción *Apply Template.* Seleccionar **Form.** Luego seleccionar el atributo **celsius** del contexto.

5. Con el paso previo se crean 2 nuevos elementos de la interfaz de usuario: **celsius** (con el tipo *InputField*), **celsius_label** (con el tipo *Label*). Se debe aplicar las siguientes propiedades:

celsius_label

Property	Value
Text	Celsius
LabelFor	celsius
PaddingTop	large

celsius

Property	Value
tooltip	Temperatura en grados Celsius
PaddingTop	large

6. Desde la ventana *Outline* view, se selecciona el elemento **FormGroup** y desde el menú contextual elegir la opción *InsertChild.* Posteriormente, seleccionar **HorizontalGutter** y presionar **Ok.**

7. Desde la ventana *Outline* view, se selecciona el elemento **FormGroup** y desde el menú contextual elegir la opción *InsertChild.* A continuación, seleccionar **Button** y asignar el nombre ConvertirButton. Asignar las siguientes propiedades:

ConvertirButton

Property	Value
tooltip	Ir a la vista siguiente a ver el resultado
Event > onAction	Traducir
hAlign	center

Una vez concluida la vista ***FormularioView***, se continúa con la vista ***ResultadoView***.

Pasos para la realización del diseño de la vista *ResultadoView*

1. Desde la ventana *Outline* view, se elige el nodo raíz **RootUIElementContainer** y se agrega un nuevo elemento del tipo Group llamado ResultadoGroup. Con las siguientes propiedades:

Property	Value
layout	GridLayout
cellPadding	5
colCount	1

2. Se elige el nodo hijo **Group_header** y se especifican las siguientes propiedades:

Property	Value
text	Conversión Temperatura

3. Desde la ventana *Outline* view, se selecciona el elemento **ResultadoGroup** y desde el menú contextual elegir la opción *Insert Child.* Se selecciona un elemento del tipo Transparent Container. Asignar el nombre ResultadoContainer.
4. Desde la ventana *Outline* view, se selecciona el elemento **ResultadoContainer** y desde el menú contextual se elige la opción *Apply Template.* Nuevamente se selecciona el tipo Form y elegir el atributo Result para ser desplegado en el formulario.
5. Desde la ventana *Outline* view, se selecciona el elemento **ResultadoGroup** y desde el menú contextual elegir la opción *Insert Child.* Seleccionar un elemento del tipo HorizontalGutter y asignarle el nombre Gutter.
6. Desde la ventana *Outline* view, se selecciona el elemento **ResultadoGroup** y desde el menú contextual elegir la opción *Insert Child.* Seleccionar un elemento del tipo Button, con el nombre VolverButton, y asignar las siguientes propiedades.

VolverButton

Property	Value
Text	Volver
Tooltip	Ir a la vista anterior
Event -> onAction	Volver

Finalmente, se guardan los cambios en la metadata y se selecciona el ícono "***Save All Metadata***" que se encuentra en la barra de Herramientas.

Pasos para la implementación de la lógica de negocio en el Component Controller

1. Abrir la ventana *EditorView* de **Component Controller,** haciendo doble clic sobre el nodo correspondiente en la jerarquía del proyecto.
2. Agregar un método en la solapa Methods del mismo. ***New → Method → Next.***
3. Al método asignarle el nombre **EjecutarConversion** y seleccionar **OK**.

4. Editar los siguientes métodos en la solapa "***Implementation***".

```
public void wdDoInit()
  {
//@@begin wdDoInit()
        // crear el objeto del modelo
        Request_TemperatureConvertor_celsiusToFahrenheit requestMO = new
        Request_TemperatureConvertor_celsiusToFahrenheit();
        // bindear el objeto ejecutable al nodo correspondiente del contexto
        wdContext.nodeTempRequest().bind(requestMO);
        msgMgr = wdThis.wdGetAPI().getMessageManager();
        //@@end
  }
```

Luego,

```
public void EjecutarConversion( )
  {
    //@@begin EjecutarConversion()
        try
        {
          // Ejecutar el Web Service y actualizar el modelo 'Response'
Request_TemperatureConvertor_celsiusToFarenheit requestMO =
wdContext.currentTempRequestElement().modelObject();
requestMO.execute();
          wdContext.nodeResponse().invalidate();
        }
        catch (Exception ex)
        {
          msgMgr.reportException(ex.getLocalizedMessage(), true);
        }
    //@@end
  }
```

Y finalmente en la sección **Others** se agrega:

```
  //@@begin others
        IWDMessageManager msgMgr;
  //@@end
```

Pasos para agregar la llamada al Servicio en FormularioView

1. Se editan los siguientes métodos en la solapa "***Implementation"*** de la vista **FormularioView.**

```
public void onActionTraducir(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
 {
   //@@begin onActionTraducir(ServerEvent)
   wdThis.wdGetConsumoWSComponentController().EjecutarConversion();
   wdThis.wdFirePlugAResultadoView();
   //@@end
 }
```

2. Se guarda la nueva metadata completa con el ícono ***Save All Metadata*** que se encuentra en la barra de herramientas.

Antes de poder seguir con las tareas de "deployment" (se puede realizar por contar con el motor J2EE); se necesita la obtención de un objeto que tenga comprimido en su interior **todos los elementos y la componente que estuvimos creando**. Dicho objeto es una **Web Dynpro Application**. Una vez que tenemos ese objeto "Web Dynpro Application", la aplicación se encuentra en condiciones de ser "deployada".

Pasos para la creación de una Web Dynpro Application

1. Para abrir el Asistente , elegir ***Create Application*** desde el menú contextual del nodo ***Applications.***
2. Ingresar el nombre para la Web Dynpro Application: **ConversionApp**, y especificar el nombre del paquete **com.teknoda.convert.app** para las clases de Java y luego presionar ***Next.***
3. En la ventana siguiente, se elige la opción ***Use existing component*** y se presiona ***Next.***
4. Y luego, en la próxima ventana que aparezca, se dejan los valores asignados por default y se presiona ***Finish***.

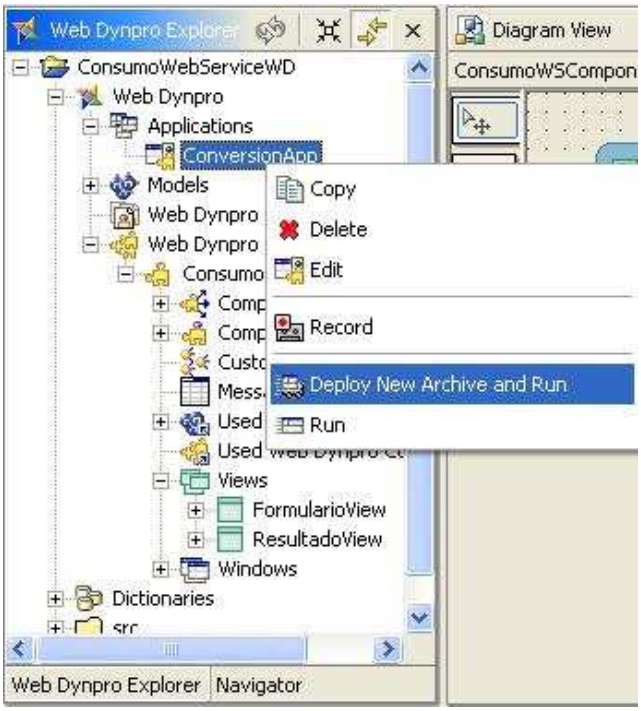
Finalmente, resta realizar un "deploy " de la **Web Dynpro Application**, para luego poder "lanzarla" (ejecutarla) y ver la misma desplegada en el Browser:

Para poder hacer un "deployment" de la aplicación y luego ejecutarla, se requiere que se encuentren ejecutando el ***SAP J2EE Engine*** *y el* ***Software Deployment Manager (SDM)***

(Nota: Para el chequeo de los seteos del Servidor, se selecciona el menú ***Window → Preferences → SAP J2EE Engine.)***

Ejecución de una Web Dynpro Application

Para ejecutar la aplicación, se debe hacer click derecho sobre el nodo **ConversionApp** en el panel **Web Dynpro Explorer** y seleccionar la opción: **"Deploy New Archive and Run",** como se muestra en la figura siguiente:



5. Para tener en cuenta …

- **Opción de Importar modelo desde archivo local :** Las opciones de importación de la estructura del modelo abarcan no sólo la vía que hemos utilizado a través de la URL correspondiente al archivo WSDL del Web Service., sino que también existe la posibilidad de contar con el archivo WSDL en una carpeta local. En este caso se puede consultar dicho archivo, que muchas veces contiene datos respecto de la estructura del web service y documentación acerca de la funcionalidad de dicho servicio.

  Como ejemplo, se podrá observar que si en un Browser se colocara la dirección utilizada en el ejemplo para importar el Web Service, obtendremos el propio archivo WSDL correspondiente y podremos guardarlo localmente para examinarlo, y hasta eventualmente modificarlo.

  Generalmente ésto no es necesario, y sólo en casos excepcionales se requerirá modificar dichos archivos, pero un caso particular y útil es aquel en el que se desea pasar algún **parámetro fijo** al WebService.

  Para este caso mostramos un ejemplo utilizando el fragmento específico del archivo WSDL a modificar.

  Tomemos como ejemplo un web service que realiza la traducción de un texto a distintos idiomas según se especifique en un parámetro llamado LanguageMode. El siguiente fragmento especifica la dirección a la cual se realizará la petición del servicio:

```
<wsdl:port name="TranslateServiceHttpGet" binding="tns:TranslateServiceHttpGet">
<http:address location="http://www.webservicex.net/TranslateService.asmx"/>
</wsdl:port>
```

Lo que se puede realizar es agregar en esa dirección el pasaje de algún parámetro conocido, que necesitemos que quede establecido como fijo. A veces en algunos Web Services de SAP, como aquellos generados a partir de RFC, necesitan del parámetro **sap-client**, y éste es un punto interesante donde agregarlo.

En este ejemplo podemos hacer que la traducción sea siempre de Inglés a Español, como se ve a continuación:

```
<wsdl:port name="TranslateServiceHttpGet" binding="tns:TranslateServiceHttpGet">
<http:address
location="http://www.webservicex.net/TranslateService.asmx?LanguageMode=EnglishTOSpanish"/>
</wsdl:port>
```

LanguageMode podría haber tomado otros valores como especifica la documentación de dicho Web Service, por ejemplo, SpanishTOEnglish, EnglishTOChinese, y habérselo asignado dinámicamente, pero es interesante la posibilidad antes mencionada.

- **Disponibilidad del Web Service:** En el ejemplo práctico que desarrollamos en el presente tip, se utilizó un Web Service alojado en un **server privado**, que ofrece la posibilidad de consumirlo en forma gratuita, siempre y cuando se encuentre disponible. Pero, hay que tener que cuenta que, en algunos casos, ya sea por congestionamiento del servicio o por simple baja del mismo, es probable que no se pueda acceder a él.

El ejemplo desarrollado en este tip es una aplicación práctica, genérica y simple que se puede adaptar fácilmente al consumo de **cualquier Web Service al cual se tenga acceso.** Por lo tanto es válido aclarar que los pasos seguidos en este tip **son los mismos que se llevarían a cabo** para cualquier otro Web Service que se quiera consumir.

6. Dónde obtener información adicional

SAP Community Network http://sdn.sap.com

SAP Help Portal - http://help.sap.com

IMPORTANTE

*Dudas o consultas: **sapping@teknoda.com***