Desarrollo web orientado a dispositivos móviles

Roberto Montero Miguel

1. Introducción

Hoy en día se han incrementado notablemente las capacidades de los dispositivos móviles para navegar por internet.

Las conexiones a la red a través de dispositivos inalámbricos cada vez se produce con más frecuencia, gracias a:

- Nuevas tecnologías de conexión como GPRS y UMTS , más rápidas y fiables.
- Las compañías de Telecomunicaciones han ampliado sus ofertas de este tipo de servicios, de forma que la contratación de los mismos, están al alcance económico del "ciudadano corriente".
- Auge de las redes inalámbricas WIFI. Hoy en día podemos encontrar redes WIFI en la mayor parte de los lugares públicos.
- Gran oferta de servicios públicos accesibles a través de los dispositivos móviles.

La mayor parte de las páginas de internet, no son accesibles a través de dispositivos móviles o no son practicables a través de este tipo de aparatos.

Lo que quiero decir con que "no son practicables", es que a pesar de que una página web puede ser visualizada desde un dispositivo móvil como un teléfono, no significa que esa página sea navegable cómodamente desde dicho dispositivo. Por ejemplo, un portal con un menú con 20 opciones en la página principal, puede ser útil para visualizarlo desde un PC, pero para un dispositivo móvil con una pantalla limitada, no es práctico que se le presenten al usuario tantas opciones en el menú principal. Con este ejemplo lo que pretendo hacer ver es que para que nuestra aplicación sea accesible desde diferentes dispositivos móviles, posiblemente tendremos que realizar al menos dos diseños de presentación diferentes (2 vistas para un mismo modelo): un diseño para PC como tradicionalmente se viene haciendo y otro diseño para dispositivos móviles con una estructura organizativa del portal mas simplificada que el diseño tradicional. Por ejemplo, algunas de las opciones del portal para PC, pueden ser eliminadas para el dispositivo móvil o presentadas de forma diferente.

La justificación de tener que realizar diferentes diseños para representar el mismo contenido en diferentes dispositivos, vendría dada por:

- Las grandes diferencias de interactuación entre un PC y los diferentes dispositivos móviles. Por ejemplo:
    - El uso de links (elemento más básico de una página web). Es diferente como un usuario maneja el puntero de una PDA, a como el usuario de un teléfono maneja el puntero a través del teclado, o como un usuario de un PC maneja el cursor a través del ratón. En muchos casos el dispositivo no incorpora un puntero como tal, sino que se mueve a través de la página web mediante el teclado, esto provoca la desaparición de algunos métodos javascript como "onmouseover".

- Diferencias físicas:
  - El tamaño de la pantalla. No podemos comparar la cantidad de información que puede entrar en una pantalla de 19 pulgadas, con la capacidad de una pantalla de 1 pulgada.
  - En algunos casos ausencia de Teclado o Ratón.

- Limitaciones en los navegadores (browser):

  - Como cualquier desarrollador Web ya sabe, existen grandes diferencias en la ejecución de código cliente (HTML, javascript y CSS) en distintos navegadores. Esto en algunos casos es un verdadero quebradero de cabeza para el programador. Pues bien, en el caso de los navegadores de los dispositivos móviles se acentúa aun más, ya que dentro de los dispositivos móviles, existen navegadores que soportan WML, otros CHTML (Compact HTML) y otros XHTML. Por si fuera poco, dentro de cada una de estas especificaciones existen diferentes versiones, como por ejemplo para WML podemos encontrar diferentes versiones WML 1.1, 1.2 y 1.3. El mismo problema tendremos para el soporte Javascript y CSS. Siempre tendremos que tener en cuenta que para la mayoría de los dispositivos móviles, la ejecución de Javascript es muy limitada o nula.
  - A parte de lo comentado anteriormente, una curiosidad más que cualquier programador web ya habrá experimentado, es cuando realizamos un diseño para Internet Explorer y posteriormente lo probamos por ejemplo en el navegador Safari, hay muchas posibilidades de que los elementos de nuestro pagina web se visualicen de diferente forma o incluso que ni se visualicen y eso que se supone que ambos navegadores cumplen ciertos estándares. Pues bien, esto en el caso de los navegadores de los dispositivos móviles, es llevado al extremo. Muchos navegadores de este tipo de aparatos no cumple el estándar o si se supone que lo hace, no tiene nada que ver la representación del código cliente en un dispositivo o en otro.

- Limitaciones en la ejecución de contenidos multimedia:
  - Las restricciones relativas a la ejecución de contenidos multimedia vienen dadas por las limitaciones físicas de los aparatos, principalmente por las dimensiones de la pantalla y la escasez de recursos de estos dispositivos (CPU y memoria). Algunas de estas limitaciones son:

    - Limite de tamaño y formato de las imágenes.
    - Limite de tamaño y formato de los videos. Algunos dispositivos incorporan capacidades de Streaming de video, pero hay que tener en cuenta que normalmente no permiten empotrar el reproductor dentro del navegador.

Según lo que hemos mencionado hasta ahora, podremos afirmar y justificar la existencia

de diferentes vistas para un mismo modelo, creando al menos una vista para los dispositivos con grandes capacidades y al menos otra para dispositivos con capacidades más limitadas. Probablemente usted se pregunta, " Si existen tantas diferencias entre los diferentes dispositivos y navegadores...., ¿será suficiente con dos diseños? Pues la respuesta es que posiblemente no sea suficiente, es más, en mi opinión es imposible que una aplicación web, llegue a funcionar correctamente en todos los dispositivos. Lo que intentaremos es que se visualice correctamente en la mayoría de los dispositivos, haciendo hincapié en los dispositivos y navegadores más comunes y estándar.

Un punto interesante a considerar cuando intentamos desarrollar una aplicación Web de este tipo, sería identificar el dispositivo que está realizando la petición a nuestra aplicación y dependiendo de las características del dispositivo, ofrecerle una interfaz gráfica adaptada a sus capacidades.

2. Identificación del navegador cliente

Seguramente la mayoría de los que habéis desarrollado aplicaciones web, habéis tenido que identificar el navegador del cliente y cargar unos u otros javascripts o CSS. Esto se venía realizando mediante código cliente javascript:

```
<script>
    function test(){
        var nom = navigator.appName;
        alert(nom);
    }
</script>
```

Otra forma de detectar el navegador del cliente con javascript:

```
var tipoBrowser;
if (navigator.userAgent.indexOf("MSIE")<0){
if (navigator.userAgent.indexOf("Opera")<0){
tipoBrowser = "Mozilla";
}
else{
tipoBrowser = "Opera";
}
}
else {
tipoBrowser = "Ms";
}
return tipoBrowser
```

Por ejemplo, podremos cargar una hoja de estilos u otra dependiendo del navegador:

```
<SCRIPT LANGUAGE = "JavaScript">
<!--
var browser = '';
var version = '';
var entrance = '';
var cond = '';
// BROWSER?
if (browser == ''){
if (navigator.appName.indexOf('Microsoft') != -1)
browser = 'IE'
else if (navigator.appName.indexOf('Netscape') != -1)
browser = 'Netscape'
else browser = 'IE';
}
if (version == ''){
version= navigator.appVersion;
paren = version.indexOf('(');
whole_version = navigator.appVersion.substring(0,paren-1);
version = parseInt(whole_version);
}
if (browser == 'IE' && version >= 4) document.write('<'+'link
rel="stylesheet" href="ie.css" />');
if (browser == 'Netscape' && version >= 2.02) document.write('<'+'link
rel="stylesheet" href="nn.css" />');
// -->
</SCRIPT>
```

Estos métodos son válidos para cuando el navegador cliente es el de un PC, recordemos que algunos dispositivos móviles no aceptan javascript, o no disponen del objeto "navigator". También descartaremos este método, ya que javascript es código que se ejecuta en el cliente, y nosotros queremos cargar una página JSP u otra dependiendo del tipo del cliente conectado.

Desde el lado servidor, podremos obtener una identificación del cliente a través de los "headers" de la petición, concretamente recogiendo el valor del parámetro "user-agent". Según la definición de la Wikipedia, el User-Agent o Agente de Usuario es una aplicación informática que funciona como cliente en un protocolo de red; el nombre se aplica generalmente para referirse a aquellas aplicaciones que acceden a la World Wide Web. Los agentes de usuario que se conectan a la Web pueden ser desde navegadores web hasta los web crawler de los buscadores, pasando por teléfonos móviles, lectores de pantalla y navegadores en Braille usados por personas con discapacidades.

Cuando un usuario accede a una página web, la aplicación generalmente envía una cadena de texto que identifica al agente de usuario ante el servidor. Este texto forma parte del pedido a través de HTTP, llevando como prefijo *User-agent:* o *User-Agent:* y generalmente incluye información como el nombre de la aplicación, la versión, el sistema operativo, y el idioma. Los bots, como los web crawlers, a veces incluyen

también una URL o una dirección de correo electrónico para que el administrador del sitio web pueda contactarse con el operador del mismo.

Normalmente la cadena "User-Agent", se corresponde con una serie de patrones, por lo tanto podremos trocear la cadena y averiguar ciertas características del navegador cliente, por ejemplo, el nombre del navegador, versión, Sistema Operativo o idioma.

Podremos revisar la documentación de mozilla para entender los patrones del Agente de Usuario, desde la página web:
https://developer.mozilla.org/En/User_Agent_Strings_Reference.
La misma documentación existe para Microsoft: ***http://msdn.microsoft.com/en-us/library/ms537503.aspx***

2.1. Identificación del navegador desde el lado servidor

Desde nuestro servlet podremos capturar la cadena "User-Agent" y tratarla para averiguar que navegador se está conectando a nuestra aplicación.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldDevice extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

    String user_agent = request.getHeader("user-agent");
    PrintWriter out = response.getWriter();
    out.println("Hola, tu User-Agent es " + user_agent);
  }
}
```

Si rebuscamos en internet podremos encontrar multitud de clases/ servlets capaces de reconocer el tipo de navegador para los navegadores más corrientes, generalmente los usados en estaciones de trabajo PC. Un ejemplo muy útil es:
http://dadabase.de/java/browserdetection/src/de/dadabase/webutils/BrowserDetection.java.html. Si le echáis un vistazo al código fuente de esta clase, podréis comprobar como si le pasamos al constructor el objeto HttpServletRequest, podremos averiguar de qué tipo de navegador se trata. Profundizando un poco más aún, podremos ver como al igual que nuestro anterior servlet, obtiene la cadena User-Agent y posteriormente la trata mediante expresiones regulares.

Este es un buen método para detectar el navegador del cliente, pero si revisamos la página http://www.user-agents.org/, podremos ver que existen miles de User-Agents. Este método nos puede servir para detectar los navegadores más comunes (los del PC),

pero para los dispositivos móviles puede ser bastante más complicado, además a través del User-Agent, no podemos averiguar las limitaciones del dispositivo (tamaño de la pantalla, capacidad para entender HTML, formatos de video, tamaño máximo de imágenes...). En los próximos apartados veremos algunas de las soluciones, que resuelven esta problemática.

3. UAProf

La especificación "**U**ser **A**gent **Prof**ile" (UAProf), definida por la Open Mobile Allianze, se preocupa en recoger las características y preferencias para los dispositivos inalámbricos. Esta información puede ser usada por los proveedores de contenidos para proporcionar dichos contenidos en el formato adecuado para el dispositivo en cuestión.

La idea de la especificación era buena y trataba de solucionar uno de los principales problemas del desarrollo de aplicaciones web para entornos multi-dispositivo: el de conocer las características del dispositivo que va a presentar el contenido.

UAProf se diseño como un marco común para que todos los fabricantes de teléfonos móviles pudieran describir las características técnicas y funcionalidad de sus teléfonos mediante el uso de XML y *Composite Capabilities/Preferences Profile* o CC/PP para abreviar. CC/PP es un marco más amplio que permite expresar no sólo la funcionalidad de los teléfonos sino también las preferencias del usuario. De hecho, UAProf es la primera implementación de la norma CC/PP.

UAProf ha sido acogido por la mayoría de fabricantes. Hoy día, casi todos los fabricantes publican en su web los ficheros XML que describen la funcionalidad de sus móviles. Sin embargo, **no hay homogeneidad** en el uso de las reglas CC/PP, de modo que los fabricantes utilizan términos diferentes para referirse al mismo atributo o funcionalidad. Además no están obligados a especificar todas las características del dispositivo, de manera que cada fabricante solo específica las que más le interesa.

Un sistema de gestión de contenidos (como un sitio wap) puede utilizar UAProf para adaptar el contenido a las características del dispositivo inalámbrico, o para decidir qué contenidos puede ofrecer para su descarga. Sin embargo, los inconvenientes de depender exclusivamente de UAProf son:

- No todos los dispositivos usan UAProfs.
- No todas las publicaciones UAProfs están disponibles (alrededor del 20% de los enlaces ofrecidos por teléfonos están muertos u obsoletos, según las cifras de UAProfile.com)
- UAProf puede contener un esquema o errores de datos que puede causar problemas al "parsear" el documento.

- Recibir y parsear los UAProfs en tiempo de ejecución, puede consumir muchos recursos, ralentizando la aplicación.
- No existe un organismo que controle y estandarice los campos del UAProf.
- El documento UAProf no contiene los agentes de usuario de los dispositivos que podrían aplicarse en el esquema (Nokia los pone entre los comentarios).Los encabezados de UAProf pueden ser erróneos.

4. WURFL

Wurfl (**Wireless Universal Resource File**) nace a partir de los inconvenientes detectados en UAProf, intentando unificar en una estructura única las características de los dispositivos inalámbricos. Las características de estos dispositivos son introducidas a partir de las características publicadas por los fabricantes en el UAProf y las contribuciones de desarrolladores y empresas de todo el mundo.

WURFL (**http://wurfl.sourceforge.net**) es un fichero XML que contiene las capacidades y características de la mayoría de los dispositivos. El principal objetivo de este fichero es recolectar toda la información posible de los dispositivos móviles.

Se trata de un proyecto open-source y existen multitud de APIs en diferentes lenguajes de programación para trabajar con el XML de WURFL:

- Java
- Php
- Perl
- Ruby
- Python
- .Net
- Xslt
- C++

Por supuesto, en este artículo nos centraremos en el API Java, pero antes de comenzar con el código, quizá sea interesante hacer un repaso de la información que podemos obtener del fichero XML de WURFL. Los datos aquí representados están disponibles en la Web http://wurfl.sourceforge.net/help_doc.php, en este articulo solo veremos algunas de las características más destacadas:

Group:**product_info** (marca y modelo de nombre y demás información genérica)

Capability Name	Type	Descripción
brand_name	string	Marca (ej: **Nokia**)
model_name	string	Modelo (ej: **N95**)
marketing_name	string	Además de la marca y modelo, algunos dispositivos incluyen el nombre comercial (ej: BlackBerry 8100 **Pearl**, Nokia 8800 **Scirocco**, Samsung M800 **Instinct**).
is_wireless_device	true/false	Retorna información acerca si el dispositivo es inalámbrico o no. Los teléfonos móviles y las PDA son consideradas dispositivos inalámbricos, mientras que los PC de escritorio o portátiles no.
uaprof,uaprof2,uaprof3	String (URL)	Las Urls UAProf.
device_os	String	Información acerca del Sistema Operativo.
device_os_version	String	Versión del Sistema Operativo
mobile_browser	String	Información acerca del navegador del dispositivo (Openwave, Nokia, Opera, Access, Teleca,...)
mobile_browser_version	String	Versión del navegador

Group:**markup** lenguages soportados

Capability Name	Type	Descripción
xhtml_support_level	[-1 |..| 4]	Suponiendo que el dispositivo es compatible con alguna forma de XHTML, esta capacidad determina el nivel de soporte de estas etiquetas: • Nivel "-1": No soporta XHTML, en estos casos el dispositivo normalmente soporta WML. • Nivel "0" : Soporte básico XHTML. Pantalla mínima de 100 pixel. No soporta CSS. Soporte básico o nulo de tablas. Soporte de formularios

		básico. • Nivel "1" : XHtml con cierto soporte de CSS. Anchura mínima de pantalla 120 pixel. Soporte básico de tablas. • Nivel "2" : Mismas características que en el nivel 1, pero en un futuro podrían variar. • Nivel "3" : Excelente soporte CSS. Bordes y márgenes correctamente aplicados. Anchura mínima de pantalla 164 pixel. Mayor soporte a tablas complejas. Soporta imágenes de fondo, incluso aplicadas desde estilos. • Nivel 4: Igual que el nivel 3 pero con soporte AJAX.
preeferred_markup	string	Indica cual es el mejor lenguaje de marcas para el dispositivo.
html_web_3_2	true/false	Soporta HTML versión 3.2
html_web_4_0	true/false	Soporta HTML versión 4
multipart_support	true/false	Soporta formularios multipart

Group:**display**

Capability Name	Type	Descripción
resolution_width	any integer number	Anchura de la pantalla en pixels
resolution_height	any integer number	Altura de la pantalla en pixels.
max_image_width	any integer number	Anchura máxima de las

		imágenes
max_image_height	any integer number	Altura máxima de las imágenes

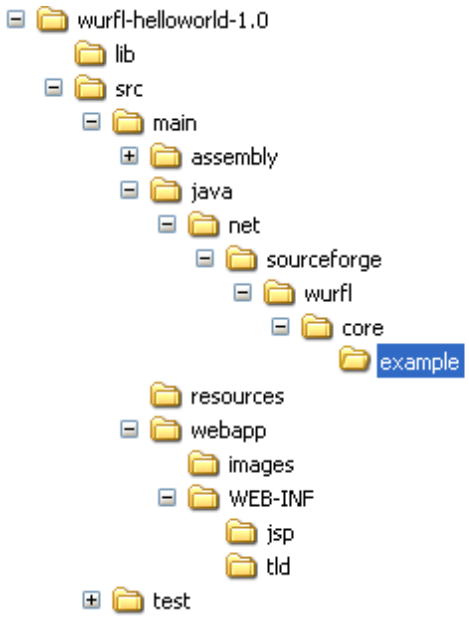
Group:**streaming**

Capability Name	Type	Descripción
streaming_video	true/false	True, si el dispositivo puede visualizar videos por streaming.
streaming_real_media	"none",8,9,10	El dispositivo acepta streaming en RealMedia .
streaming_3gpp	true/false	El dispositivo acepta streaming en 3GPP.
streaming_mp4	true/false	El dispositivo acepta streaming en MP4
streaming_wmv	"none",7,8,9	El dispositivo acepta streaming en WMV
streaming_mov	true/false	El dispositivo acepta streaming en MOV
streaming_flv	true/false	El dispositivo acepta streaming en FLV
streaming_3g2	true/false	El dispositivo acepta streaming en 3GPP 2

4.1. WURFL Java API

El proyecto WURFL, incluye un API java basada en Spring, para facilitar la interacción con el fichero XML de WURFL.
Para este articulo utilizaremos el código de ejemplo disponible en la web de WURFL: http://sourceforge.net/project/showfiles.php?group_id=55408&package_id=312989.

La estructura de directorios del ejemplo es la siguiente:



Bajo el directorio \wurfl-helloworld-1.0\src\main\java\net\sourceforge\wurfl\core\example, encontramos el servlet controlador de la aplicación, que será el punto de entrada de las peticiones web. Este servlet capturará cadena user-agent del navegador cliente, consultará las especificaciones de WURFL para dicha cadena y redirigirá a una JSP(WEB-INF/jsp) u otra dependiendo de las capacidades del dispositivo.

El fichero de WURFL se encuentra ubicado bajo el directorio WEB-INF, y generalmente viene comprimido en un fichero zip:

- wurfl.zip: XML con los user-agent para los dispositivos inalámbricos.
- web_browsers_patch.xml.gz: XML con los user-agent de los Web browsers de estaciones de trabajo PC.

En el directorio WEB-INF encontraremos los descriptores de configuración de Spring. Mediante estos descriptores, se configuran los beans del API de WURFL, que se encargan de tratar la estructura del XML, optimizando los accesos mediante el uso de la cache proporcionada por la librería de EHCache. Por suerte añadiendo estos descriptores del API de WURFL al contexto de nuestra aplicación Spring, el tratamiento del XML es transparente para nosotros. (No debemos olvidar incluir también la librería de WURFL wurfl-1.0.jar y sus dependencias).

El código más interesante de este ejemplo es el servlet controlador:

```
private static final String XHTML_ADV = "xhtmladv.jsp";
private static final String XHTML_SIMPLE = "xhtmlmp.jsp";
private static final String CHTML = "chtml.jsp";
private static final String WML = "wml.jsp";
```

```
   protected void processRequest(HttpServletRequest request,
         HttpServletResponse response) throws ServletException, IOException {

      WURFLHolder wurflHolder = (WURFLHolder)
getServletContext().getAttribute("net.sourceforge.wurfl.core.WURFLHolder");
      WURFLManager wurfl = wurflHolder.getWURFLManager();
      Device device = wurfl.getDeviceForRequest(request);
      MarkUp markUp = device.getMarkUp();
      String jspView = null;
      if (MarkUp.XHTML_ADVANCED.equals(markUp)) {
         jspView = XHTML_ADV;
      } else if (MarkUp.XHTML_SIMPLE.equals(markUp)) {
         jspView = XHTML_SIMPLE;
      } else if (MarkUp.CHTML.equals(markUp)) {
         jspView = CHTML;
      } else if (MarkUp.WML.equals(markUp)) {
         jspView = WML;
      }
      // MIME type is decided by JSP. Only in case of XHTML we will need to
      // multi-serve
      // i.e text/html vs application/xml+xhtml vs
      // application/vnd.wap.xml+xhtml
      if (markUp == MarkUp.XHTML_ADVANCED || markUp ==
MarkUp.XHTML_SIMPLE) {
         String contentType = "text/html";
         try {
            contentType = device
                  .getCapability("xhtmlmp_preferred_mime_type");
         } catch (CapabilityNotDefinedException e) {
            throw new RuntimeException(
                  "Somethingh is seriously wrong with your WURFL:"
                        + e.getLocalizedMessage(), e);
         }
         request.setAttribute("contentType", contentType);
         log.debug("ContentType: " + contentType);
      }
      request.setAttribute("markUp", markUp);
      request.setAttribute("device", device);
      request.getRequestDispatcher("WEB-INF/jsp/" + jspView).forward(request,
response);
   }
```

Este servlet redirige a una vista u otra dependiendo de las características del dispositivo. Con las siguientes tres líneas de código podemos acceder a todas las características del dispositivo cliente:

```
WURFLHolder wurflHolder = (WURFLHolder)
getServletContext().getAttribute("net.sourceforge.wurfl.core.WURFLHolder");
WURFLManager wurfl = wurflHolder.getWURFLManager();
Device device = wurfl.getDeviceForRequest(request);
```

En este ejemplo se utiliza la propiedad MARKUP para detectar la especificación de etiquetas que "entiende" el dispositivo (XHTML,WML...).

```
MarkUp markUp = device.getMarkUp();
```

Por ejemplo podríamos detectar la capacidad de soporte de video Streaming del dispositivo y ofrecerle los videos en el formato adecuado.

```
String strStreamingVideo = device.getCapability("streaming_Video");
```

5. MyMobileWeb: Framework basado en WURFL

A continuación veremos un framework que hace uso de los conceptos introducidos en los puntos anteriores. MyMobileWeb utiliza, entre otras tecnologías, el API de Wurfl para identificar el dispositivo conectado. Este framework, tiene un sistema de etiquetas jsp (taglib) para el diseño de la vista. Estas etiquetas adaptan automáticamente los contenidos a las características de los dispositivos conectados.

La filosofía de MyMobileWeb, es la de crear dos diseños: Uno para las estaciones de trabajo PC y otro para los dispositivos inalámbricos.

MymobileWeb se trata de un proyecto financiado en la convocatoria de proyectos PROFIT Tractor, del Ministerio de Ciencia y Tecnología, en el que participan una gran cantidad de partners (Telefónica I+D, TPI, UPM, URJC, y Fundación CTIC e InterMark, por parte asturiana). Plantea la realización de actividades de I+D para desarrollar tecnologías que propicien el desarrollo de la Web Móvil y su masiva utilización en todos los ámbitos de la vida diaria.

MyMobileWeb es una plataforma open source, basada en estándares abiertos, modular y de bajo coste que simplifica el desarrollo de aplicaciones y portales .mobi de calidad, proporcionando un entorno avanzado de adaptación de contenidos.

Incluye diferentes módulos que cubren todos los requisitos básicos que debe cumplir un sitio web móvil completo e integrado, ocultando a las las aplicaciones toda la complejidad relacionada con la gestión de múltiples contextos de entrega. MyMobileWeb usa **Device Information Simple API** para reconocer y obtener las capacidades de los dispositivos. Como característica de valor añadido, MyMobileWeb

incorpora algunos módulos experimentales capaces de explotar la semántica en un entorno móvil, implementando el concepto de "Web Móvil Semántica".

MyMobileWeb a lo largo de su vida, ha introducido un importante número de etiquetas para representar diferentes componentes web en el navegador, es por ello que en ocasiones puede resultar complicado conocer todos los componentes o las estructuras del proyecto. Por esta razón, el equipo de MyMobileWeb ha implementado una serie de plugins de Eclipse para facilitar las tareas al desarrollador.

5.1 Entorno de Desarrollo MyMobileWeb

Para instalar el entorno de desarrollo, primeramente necesitaremos:

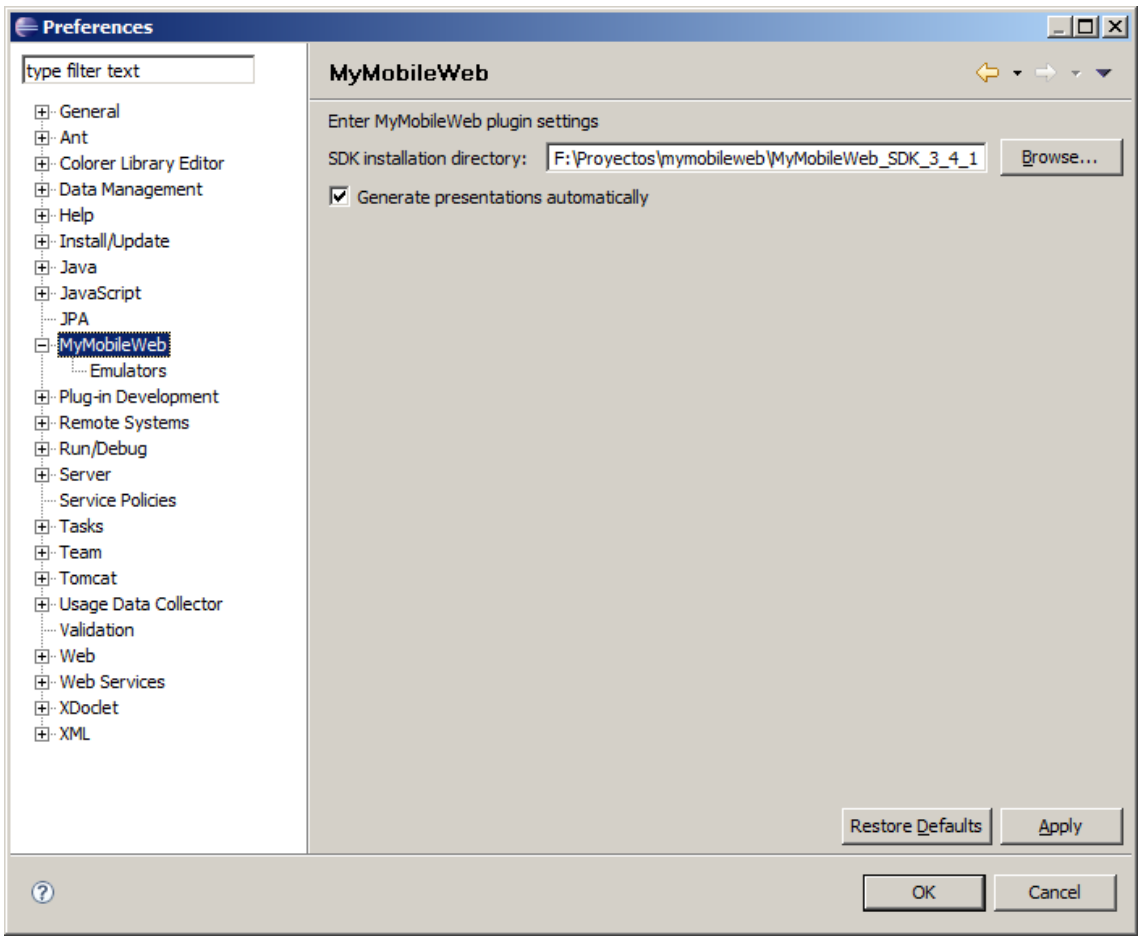
- **JDK 1.5**
- **Apache Tomcat**: Servidor de aplicaciones que usaremos por defecto. Este Framework también se integrará con nuestro servidor J2EE favorito.
- **Plugin Tomcat**: Si usamos el servidor de aplicaciones Tomcat, con este plugin podemos acelerar nuestros desarrollos, desplegando las aplicaciones fácilmente desde Eclipse.
- **MyMobileWeb_EclipsePlugin:** Este plugin añadirá "wizard" de ayuda al eclipse. Por ejemplo: Nuevo Proyecto MyMobileWeb, Nueva Operación de Presentación, Desplegar aplicación ….
- **MyMobileWeb_SDK:** Contiene una serie de utilidades y librerías que el propio plugin para eclipse usará tomando los contenidos de este paquete para construir los proyectos MyMobileWeb. Por ejemplo, una opción del plugin es Crear un nuevo Proyecto de ejemplo, para crearlo utiliza las plantillas definidas en la SDK. (En algunas versiones es necesario descargarse el paquete "examples" y descomprimirlo dentro del directorio de la SDK)

5.1.1 Configurando el entorno

Una vez instalado el software, arrancaremos el eclipse y seguiremos los siguientes pasos:

1. **Configurar la JDK**
2. **Configurar las referencias a la SDK**

Desde la ventana de preferencias (Window → Preferences) actualizaremos la ruta al directorio donde hemos descomprimido la SDK.



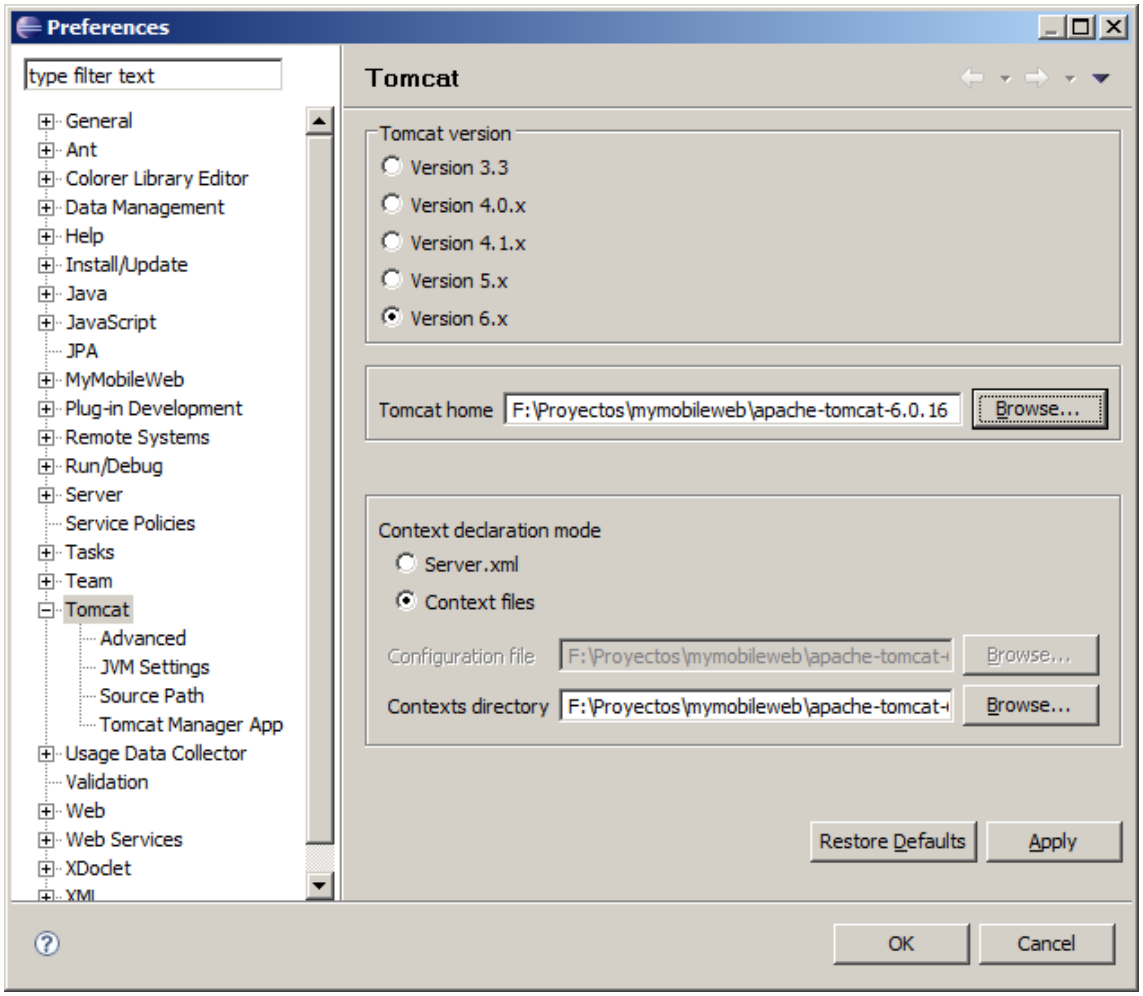
Si queremos probar nuestras aplicaciones rápidamente desde eclipse podremos configurar el emulador. Según la pagina web de este Framework, se asegura que los siguientes emuladores se integran correctamente:

- Explorer
- Firefox
- Open Ware
- Opera
- Safari
- WinWap SpartPhone

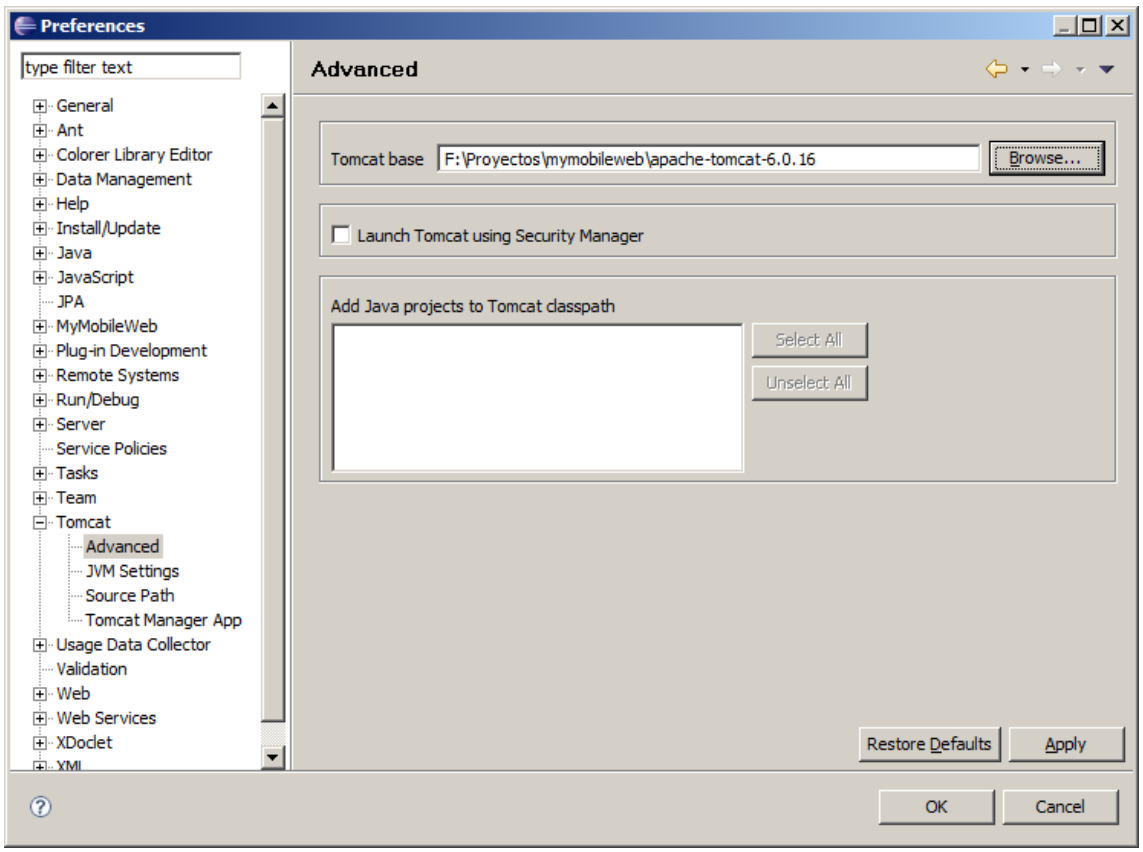
3. Configurar el servidor de aplicaciones

En este ejemplo vamos a configurar el servidor de aplicaciones Tomcat, pero también podríamos trabajar otro servidor de aplicaciones.

Configuraremos el servidor desde la opción de menú Window→Preferences→Tomcat:



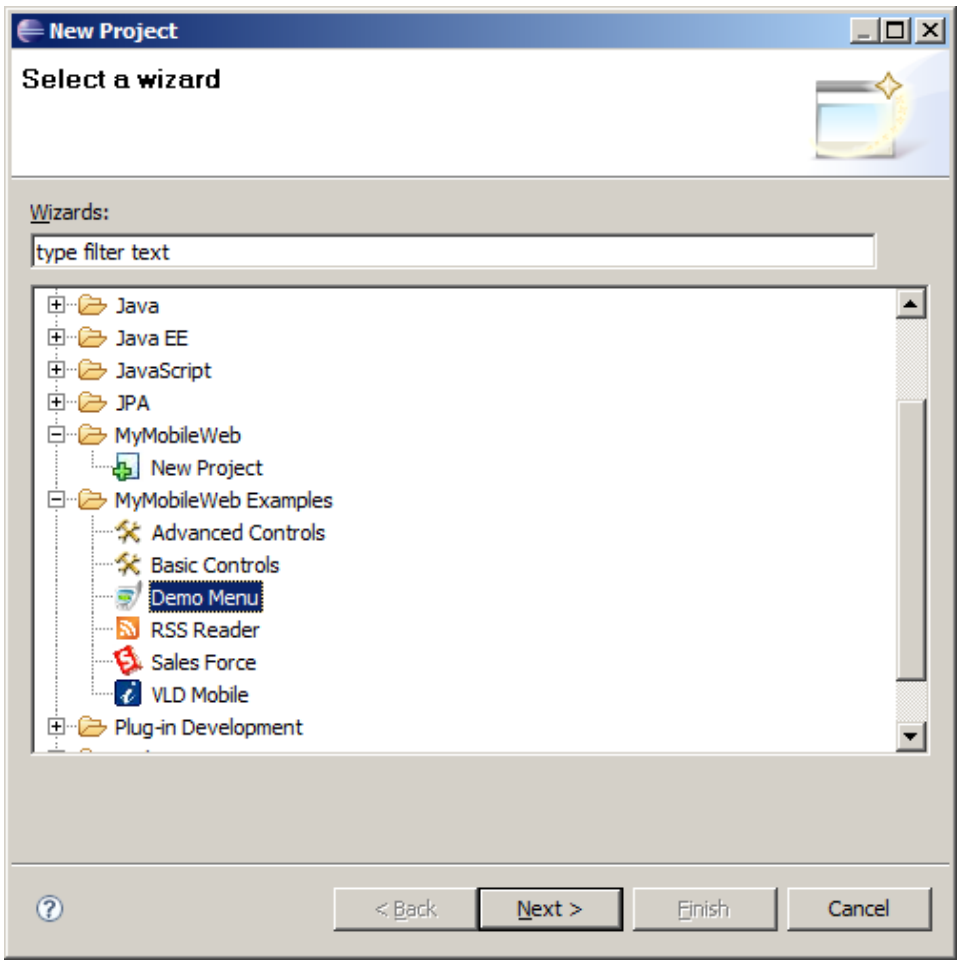
También modificaremos el contenido de la pestaña de Advanced:



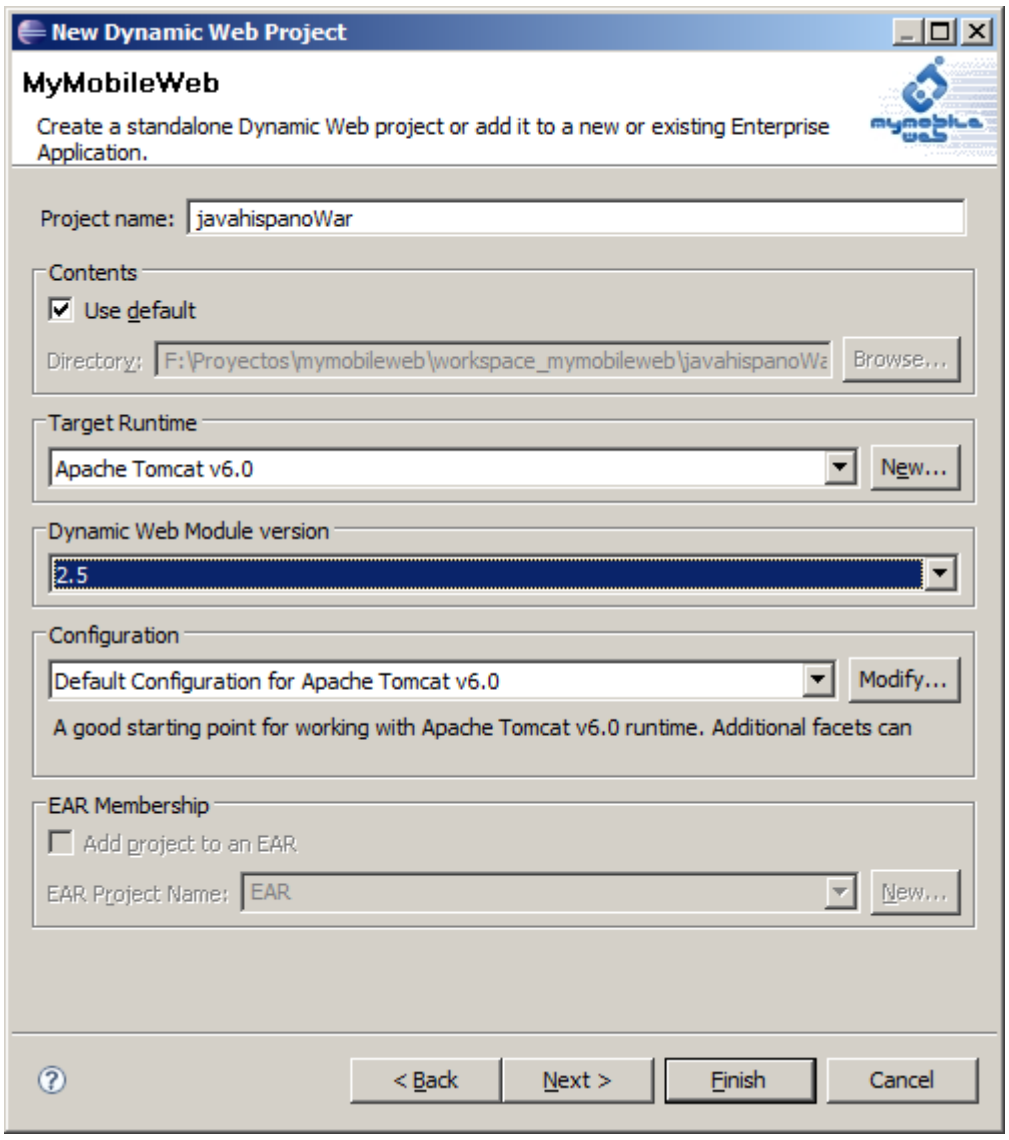
5.1.2 Primer Proyecto con MyMobileWeb

Una vez configurado el entorno, podremos crearnos nuestro primer proyecto. Para este artículo usaremos los proyectos preconfigurados que incorpora el plugin.

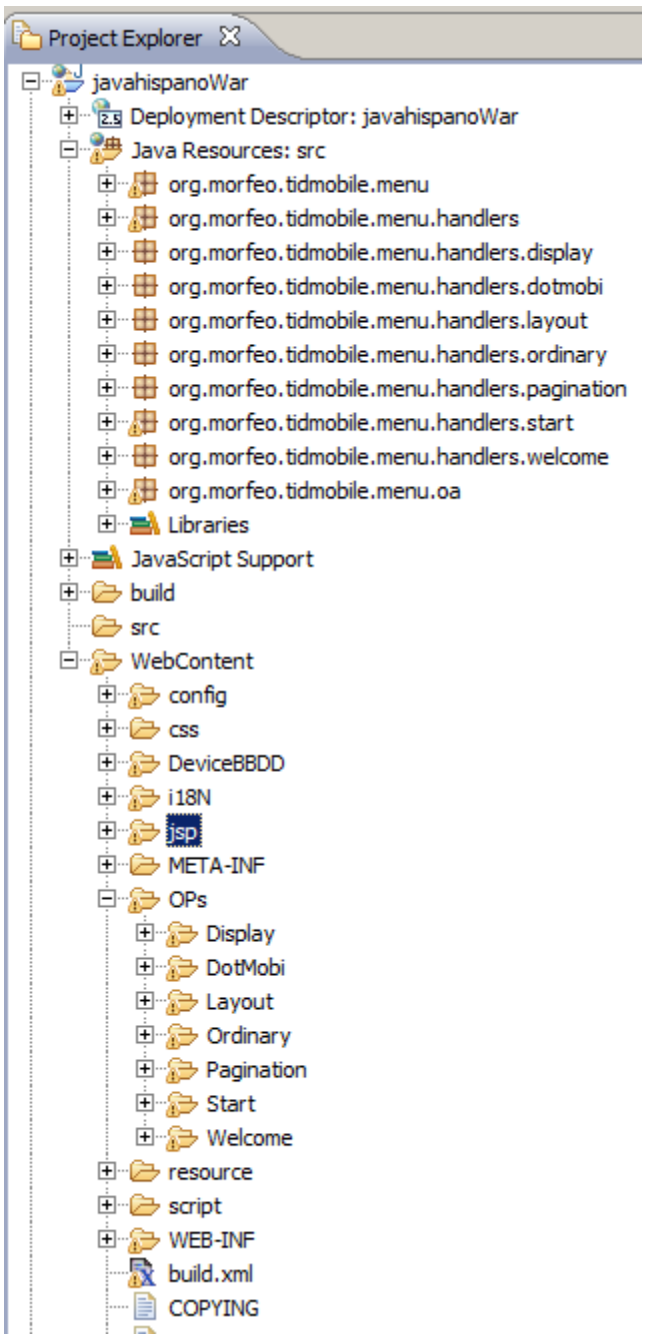
Desde la opción de menú File → Project…



En este ejemplo seleccionaremos el proyecto de ejemplo “Demo Menú”:



Cuando pulsemos el botón “Finish”, automáticamente se nos creará el nuevo proyecto en el workspace, con la siguiente estructura:



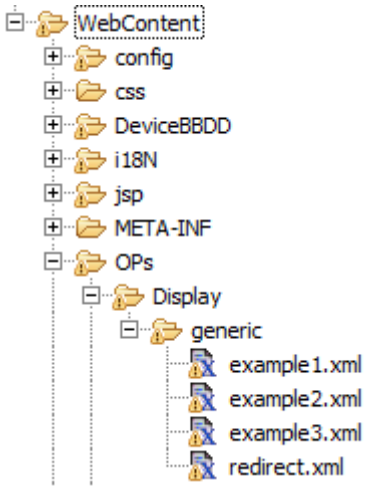
No entraré muy en detalle en la arquitectura de directorios, pero si comentaré algunas cosas por encima.

Primero decir que la aplicación se construirá a través de Operaciones de Presentación. Podemos definir una operación de presentación como cada una de las peticiones elementales que los usuarios pueden solicitar al Sistema para realizar sus operaciones de negocio. Son unidades lógicas funcionales que representan la funcionalidad mínima que tiene sentido para el usuario. Digamos que viene a ser un Caso de Uso.

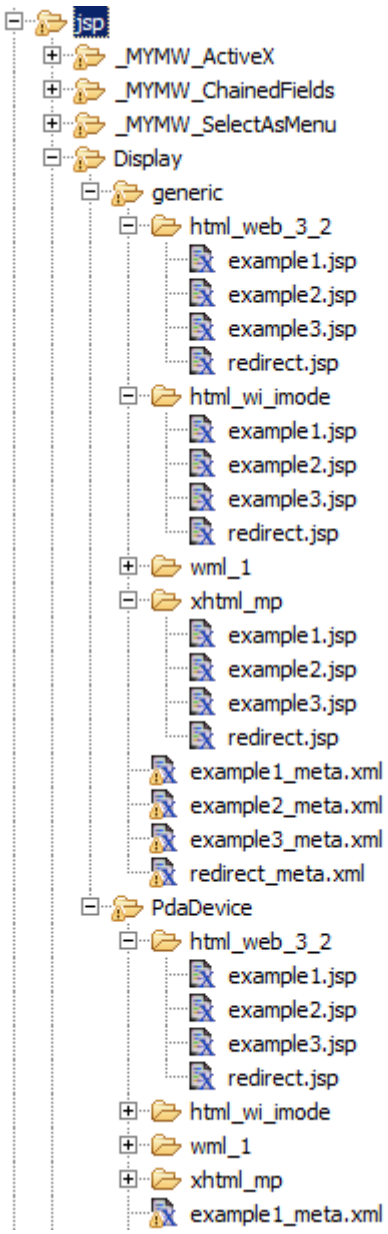
Otro detalle importante en la arquitectura de este tipo de aplicaciones es el WorkFlow. MyMobileWeb permite crear a través de los Wizard, flujos de navegación gestionados por clases o por XML.

A la hora de desarrollar una vista, trabajaremos bajo el directorio OPs. Cuando compilemos la aplicación se generará el directorio jsp, donde se habrá transformado nuestra OP, en diferentes JSPs o XMLs para los distintos tipos de dispositivos.

Nosotros editaremos las páginas del directorio OPs. En este caso tenemos la operación de presentación "Display", con cuatro vistas:



Cuando generemos o compilemos la aplicación, a través de nuestros XML de presentación se generarán:



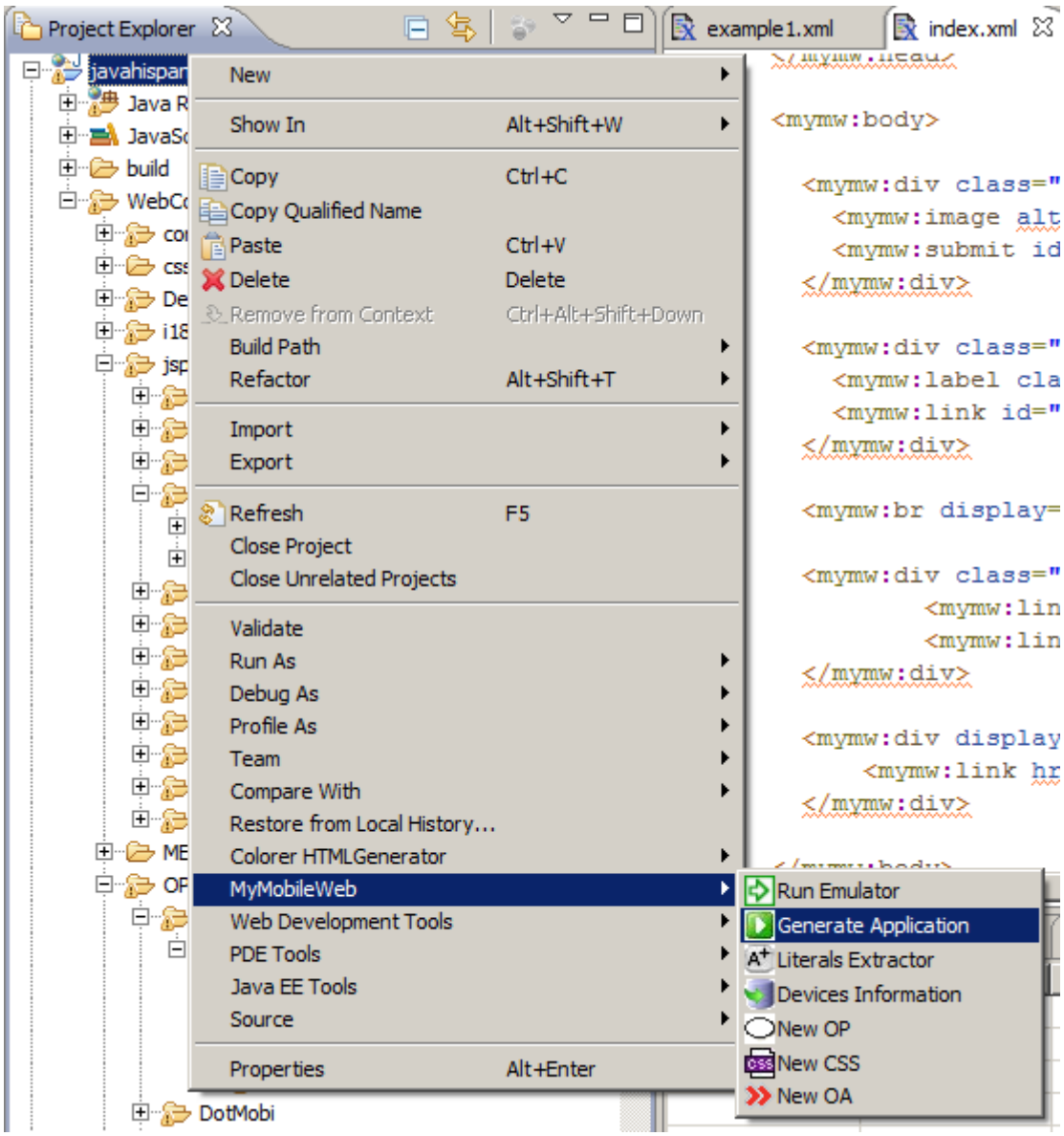
A continuación vamos a mostrar una de las vistas para posteriormente desplegarla junto con la aplicación y finalmente visualizarla en un dispositivo móvil. Abrimos el fichero \javahispanoWar\WebContent\OPs\Display\generic\example1.xml:

```
<mymw:document xmlns:mymw="http://morfeo-project.org/mymobileweb"
id="combo">

  <mymw:head>

    <mymw:title>Combo</mymw:title>
    <mymw:style href="example1.css"/>
```

```
   </mymw:head>

   <mymw:body>
         <mymw:div class="vertical" id="p1">
             <mymw:menu bind="${op}" buttonlabel="Ok" class="combo"
id="m17">
                   <mymw:link id="a">Opción a</mymw:link>
                         <mymw:link id="b">Opción b</mymw:link>
                         <mymw:link id="c">Opción c</mymw:link>
                         <mymw:link id="d">Opción d</mymw:link>

                         <mymw:link id="e">Opción e</mymw:link>
                         <mymw:link id="f">Opción f</mymw:link>
                  </mymw:menu>
         </mymw:div>

         <mymw:div display="${dcn:belongsTo('PcDevice')}" id="p2">
                   <mymw:link class="both"
href="OPs/Display/generic/example1.xml" id="code"
resourceid="code">Ver código fuente</mymw:link>
             </mymw:div>

         <mymw:include content="Layout/generic/example1/foot"/>

   </mymw:body>

</mymw:document>
```

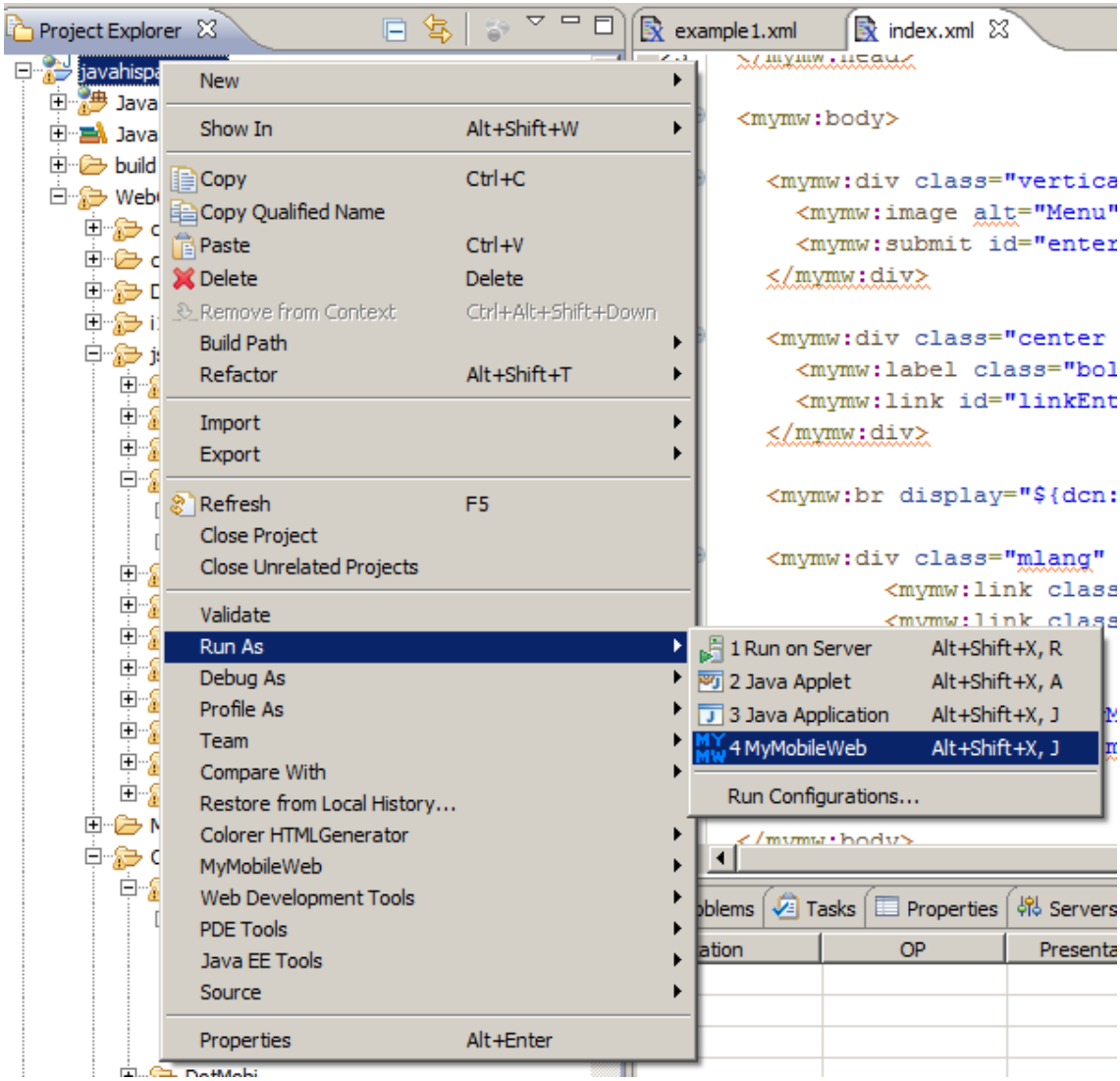
Vemos que la creación de las vistas esta sujeta a etiquetas del propio Framework. Estos tags son bastante intuitivos pero podremos encontrar una descripción de cada una de ellas en: http://mymobileweb.morfeo-project.org/wp-content/uploads/2008/02/mymobileweb_languagereference.pdf

Una vez editada la vista, podemos generar la aplicación haciendo click sobre el proyecto con el botón derecho:



Durante la generación de la aplicación se crearán las paginas Jsp comentadas anteriormente (bajo el directorio JSP).

Una vez generada la aplicación podremos desplegarla:



Desde nuestro emulador favorito, accedemos a la url:
http://localhost:8080/javahispanoWar:

En este artículo no hemos entrado apenas en detalle sobre la mayoría de las características de este Framework, pero se puede encontrar abundante documentación:

- http://forge.morfeo-project.org/wiki/index.php
- http://mymobileweb.morfeo-project.org/mymobileweb/lng/es
- http://forge.morfeo-project.org/wiki_en/index.php/MyMobileWeb_Getting_Started_3.4.1

6. Emuladores web para dispositivos inalámbricos

Uno de los problemas que nos encontramos cuando desarrollamos aplicaciones Web dirigidas a dispositivos inalámbricos, es la de como probar nuestra aplicación en diferentes dispositivos. Dado las grandes diferencias existentes entre los diferentes dispositivos, se hace de vital importancia realizar pruebas en el mayor número de ellos.

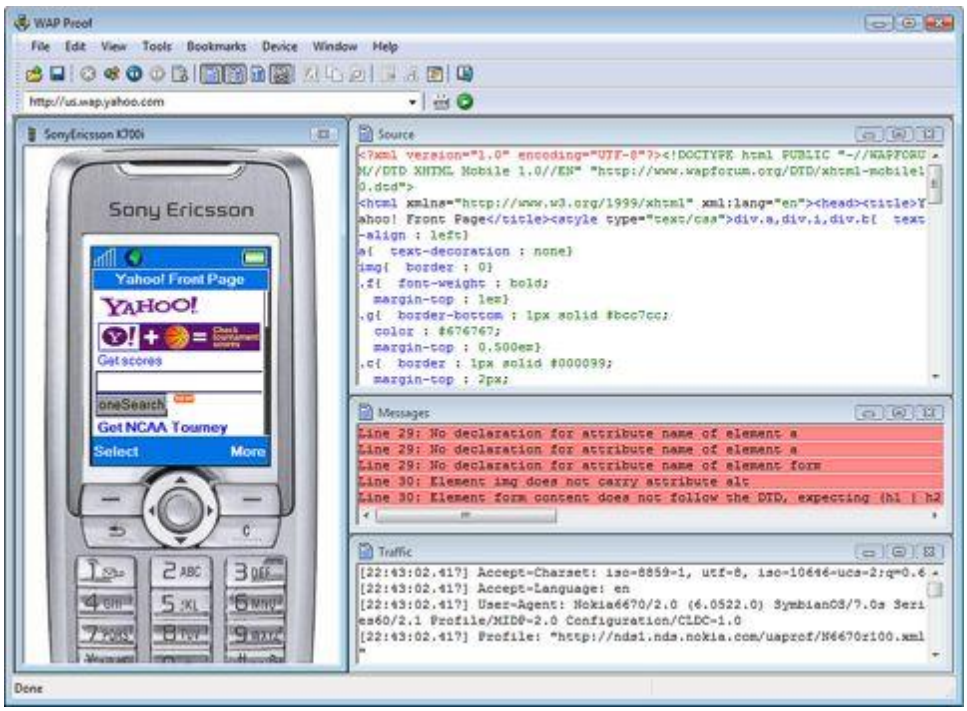
En el mercado existen algunos emuladores para realizar este tipo de pruebas. Lamentablemente la mayoría de este tipo de emuladores son de pago y no ofrecen todas las funcionalidades que podría darnos un dispositivo real. (Por ejemplo, no podremos probar el Streaming de video en estos emuladores).

WAP Proof

Es un emulador de un navegador WAP para Microsoft Windows. Está diseñado para previsualizar y depurar los sitios web móviles. Al utilizar el emulador de WAP, puedes ver páginas web móviles (WML, XHTML, HTML y cHTML) exactamente como si se abrirían por un teléfono móvil y cambiar las visualizaciones entre los diferentes dispositivos móviles. La petición HTTP enviada por este emulador WAP es similar a la petición real de un navegador móvil. WAP Proof puede utilizarse para el desarrollo de los recursos sensibles a los "user-agent.

WAP Proof puede ser utilizado para validar la sintaxis de los documentos WML o XHTML, depuración interactiva páginas WAP, y comprobar el tráfico HTTP entre el navegador y el servidor.

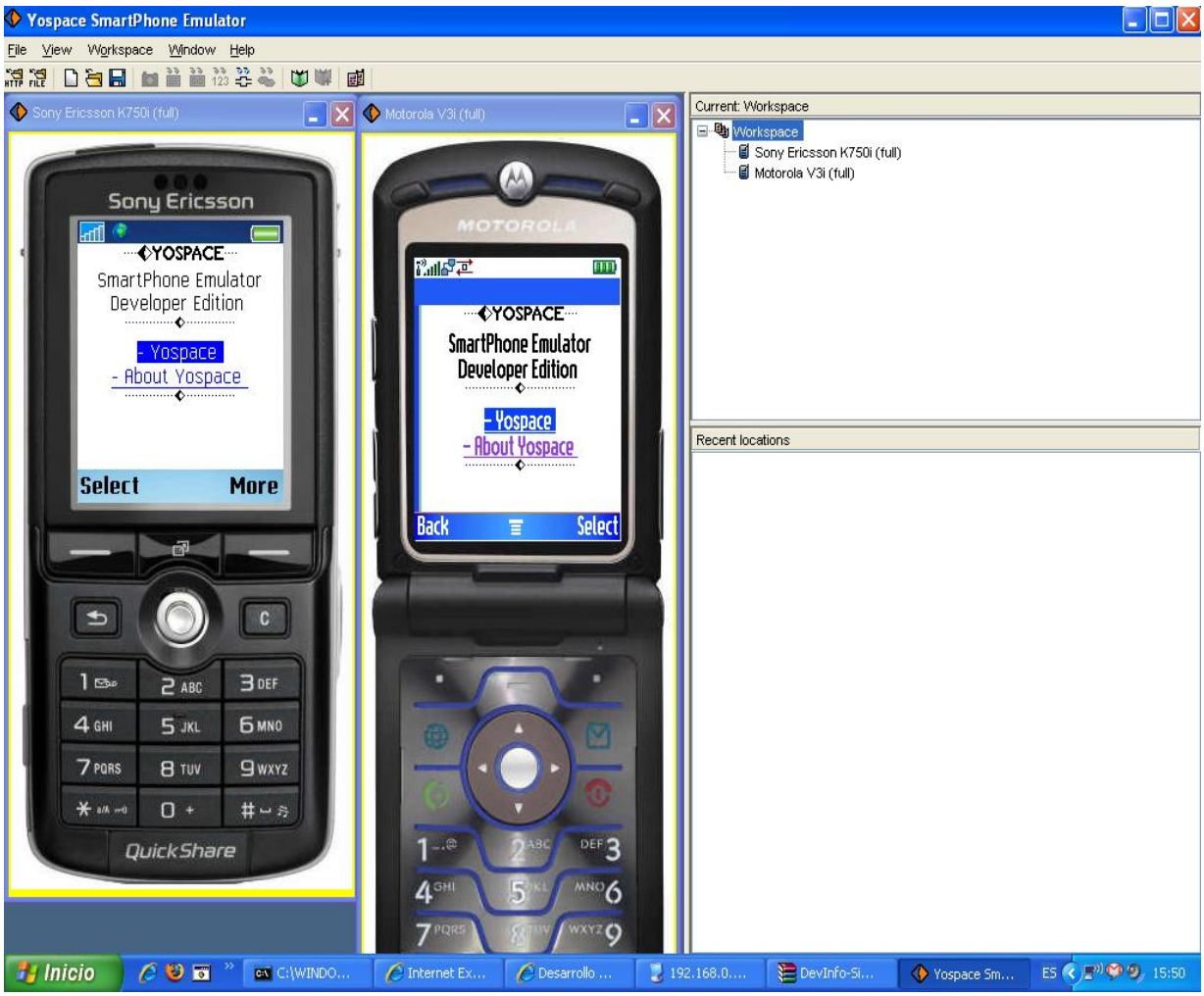
WAP Proof se trata de una herramienta comercial, su versión de prueba está limitada a 15 de evaluación y la visualización de 50 páginas por sesión.



Yospace SmartPhone Emulator

Una aplicación que te permite navegar y probar el contenido WAP desde su escritorio. Su sencilla y amigable interfaz de usuario, junto con su fiabilidad, la convierten en una valiosa herramienta para los desarrolladores de Windows, UNIX y MacOS.

Se trata de un herramienta comercial, su versión de prueba no está limitada en el tiempo, pero si en tiempo de ejecución por sesión (a 5 minutos).



WinWAP Smartphone Browser Emulator

WinWAP Smartphone es un emulador de navegador que permite utilizar los servicios WAP desde un PC con Windows. El navegador es un navegador de Internet móvil que emula las páginas web de la forma que lo haría un móvil.

Microsoft Device Emulator 2.0

Es un emulador de Windows Mobile. El emulador es algo parecido al VirtualPc, pero con Windows Mobile. Si configuramos correctamente las conexiones a través del PC,

podremos navegar y probar nuestras aplicaciones web , a través del navegador de Windows Mobile.

Con este emulador probamos la ejecución del código cliente (HTML) en el internet explorer de Windows Mobile. Hay que tener en cuenta que existen multitud de dispositivos de diferentes características con este sistema operativo. Este emulador solo tiene en cuenta la ejecución de Windows Mobile, no emula características concretas que puedan tener los dispositivos. (dimensiones de la pantalla, interacción con el teclado...)

Lo podemos descargar en:

http://www.microsoft.com/downloads/details.aspx?FamilyID=DD567053-F231-4A64-A648-FEA5E7061303&displaylang=es

Otros emuladores

Existen otros fabricantes que distribuyen su propio emulador. Normalmente este tipo de emuladores suele ser para realizar pruebas software empotrado en el teléfono o PDA, pero en algunos casos nos permite realizar pruebas con su navegador web. Un ejemplo de este tipo de emuladores, es el emulador de BlackBerry:
http://www.blackberry.com/developers/downloads/simulators/index.shtml

7. Enlaces

Agentes de Usuario Existentes: *http://www.user-agents.org/*
Definición de Agente de Usuario: *http://es.wikipedia.org/wiki/Agentes_de_usuario*
Agentes de Usuario Mozilla:
https://developer.mozilla.org/En/User_Agent_Strings_Reference
Agentes de Usuario Microsoft: *http://msdn.microsoft.com/en-us/library/ms537503.aspx*
Ejemplo de detección de navegadores desde java:
http://dadabase.de/java/browserdetection/src/de/dadabase/webutils/BrowserDetection.java.html
WURFL: *http://wurfl.sourceforge.net*
WURFL capabilities: *http://wurfl.sourceforge.net/help_doc.php*
WURFL Hello World:
http://sourceforge.net/project/showfiles.php?group_id=55408&package_id=312989
Manual MyMobileWeb: *http://forge.morfeo-project.org/wiki_en/index.php/MyMobileWeb_Getting_Started_3.4.1*
Emulador Wap Proof: *http://www.wap-proof.com/*
Yospace SmartPhone Emulator: *http://www.yospace.com/spe.html*