13. GUIA DE INSTALACIÓN

13.1 Introducción

En el siguiente capítulo se explicará cómo instalar todos los componentes necesarios para que la aplicación funcione correctamente.

La guía de instalación se estructura en dos partes:

- Servidor de Aplicaciones, que incluye:

    1. Servidor Web Apache Tomcat 5.25
    2. Apache Axis2 1.3

- Cliente móvil, que incluye:

    1. NetBeans IDE 6.0.1 con Mobility Pack
    2. Dispositivo móvil real para realizar las pruebas correspondientes.

13.2 Servidor Web Apache Tomcat 5.25

Tomcat es un servidor web con soporte de servlets y JSPs. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache, como en nuestro caso.

Tomcat puede funcionar como servidor web por sí mismo. En sus inicios existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual .

Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la *Apache Software Licence*. Las primeras distribuciones de Tomcat fueron las versiones 3.0.x. Las versiones más recientes son las 6.x, que implementan las especificaciones de Servlet 2.4 y de JSP 2.0. A partir de la versión 4.0, Jakarta Tomcat utiliza el contenedor de servlets Catalina.

13.2.1 Instalación

En este caso, hemos elegido el servidor de aplicaciones Apache-Tomcat, ya que, además de ser de

código abierto, su uso está muy extendido y es muy estable. En concreto, se ha elegido la versión 5.5, que puede descargarse en el enlace: http://www.apache.org/dist/tomcat/tomcat-5/v5.5.25/bin/.

Una vez, podemos descargarnos el .zip o simplemente, el ejecutable .exe. Una vez descargado, basta con ejecutarlo. Su instalación es muy sencilla, ya que lo único que hay que especificar es el directorio donde queremos que sea extraído. En nuestro caso, será instalado en C:\apache-tomcat-5.5.25\.

Para que todo funcione correctamente, es necesario tener instaladas las variables de entorno correspondientes:

- JAVA_HOME: C:\Java\jdk1.5.0_14
- JRE_HOME C:\Archivos de programa\Java\jre1.5.0_14

13.2.2 Funcionamiento

Para arrancar el servidor web, basta con ejecutar el archivo startup.bat contenido en el directorio C:\apache-tomcat-5.5.25\bin.

Para comprobar que efectivamente se encuentra arrancado y funcionando, debemos ejecutar:

```
http://localhost:8080/
```

Si todo es correcto, deberá aparecer la pantalla siguiente:

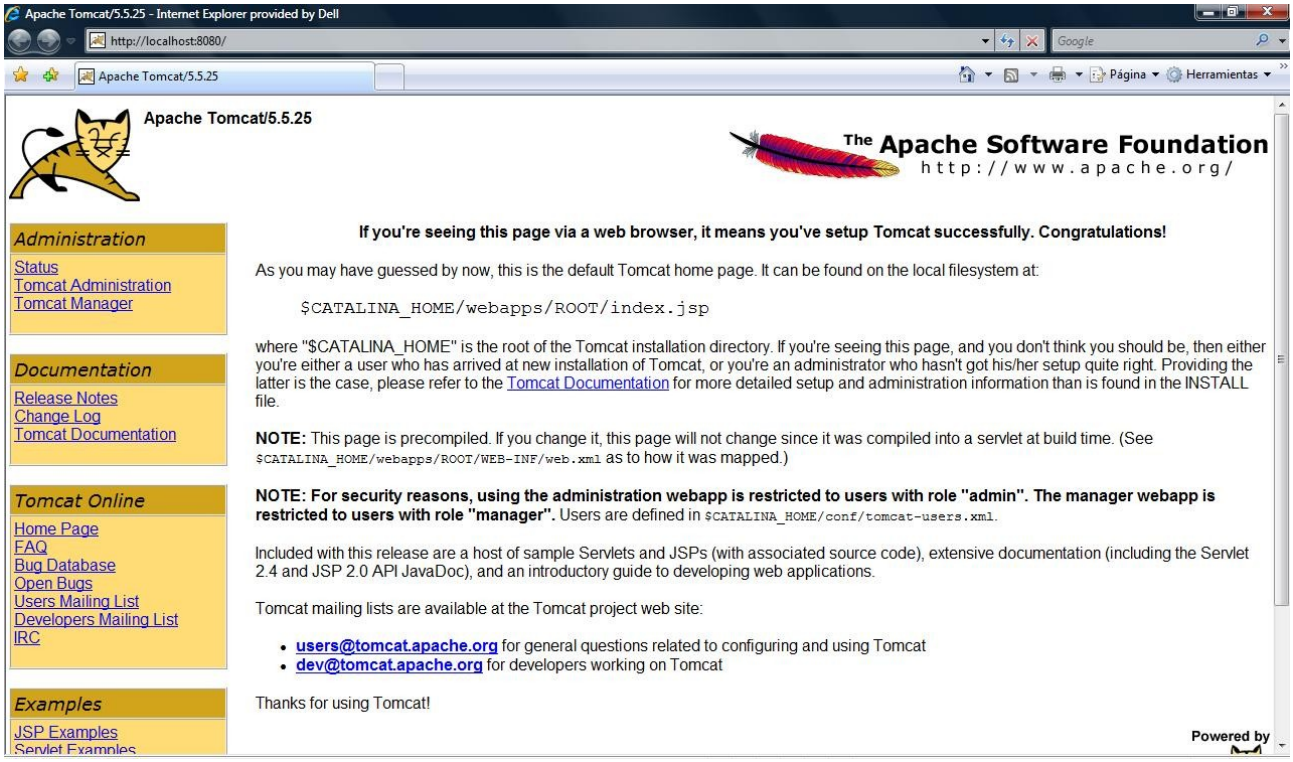


Figura 13.1. Funcionamiento de Apache-Tomcat

Para detener el servidor, ejecutaremos C:\apache-tomcat-5.5.25\bin\shutdown.

La jerarquía de directorios de instalación de Tomcat incluye:

- bin - arranque, cierre, y otros scripts y ejecutables
- common - clases comunes que pueden utilizar Catalina y las aplicaciones web
- conf – ficheros XML y los correspondientes DTD para la configuración de Tomcat
- logs - logs de Catalina y de las aplicaciones
- server - clases utilizadas solamente por Catalina
- shared - clases compartidas por todas las aplicaciones web
- webapps - directorio que contiene las aplicaciones web
- work - almacenamiento temporal de ficheros y directorios

13.3 Módulo Apache Axis2 versión 1.3

Para poder desplegar los servicios web, es necesario tener instalado el módulo apache-axis en el servidor de aplicaciones Apache-Tomcat.

13.3.1 Instalación

Antes de instalar el módulo de axis2 para apche, es necesario tener instalado la variable de entorno JAVA_HOME, que ya se definió con anterioridad. Una vez hecho esto, lo siguiente es descargar el módulo axis2.

Para descargar la distribución de axis2, es necesario ir a la página de distribución oficial: http://ws.apache.org/axis2/download/1_3/download.html#std-bin. Una vez ahí, nos descargamos tanto la distribución fuente (Source Distribution), como la distribución WAR.

Antes que nada, extraemos el directorio axis2-1.3.zip descargado en el directorio donde está contenido el motor de servlets, es decir, en C:\apache-tomcat-5.5.25\webapps\axis2-1.3.

Lo siguiente es copiar el archivo axis2.war descargado justo con el .zip al directorio de los webapps del motor de servlets de apache-tomcat, es decir, en C:\apache-tomcat-5.5.25\webapps.

Antes de seguir, es necesario establecer la variable de entorno necesaria: AXIS2_HOME que apunte a C:\axis2-1.3. Dicha variable apunta al directorio bin donde se encuentra instalado el módulo de apache, es decir, el directorio bin del .zip extraído con anterioridad.

La estructura del directorio de axis2.war es la siguiente:

```
axis2-web
META-INF
WEB-INF
    classes
    conf
        axis2.xml
    lib
        activation.jar
        ...
        xmlSchema.jar
    modules
        modules.list
        addressing.mar
        ...
        soapmonitor.mar
    services
        services.list
        aservice.aar
        ...
        version.aar
    web.xml
```

Comenzando en la cima, axis2-web es una colección de JSPs que permiten la administración de aplicaciones Axis2, con la cual se puede realizar cualquier acción tal como la adición de servicios y el enganchar y desenganchar módulos. El directorio WEB-INF contiene las clases Java y otros archivos de soporte para ejecutar cualquier servicio desplegado en el directorio *services*. El archivo principal en todo esto es axis2.xml, el cual controla cómo la aplicación se ocupa de los mensajes recibidos, determinando si Axis2 necesita aplicar cualesquiera de los módulos definidos en el directorio *modules*. Los servicios se pueden desplegar como archivos *.aar, como se puede ver aquí, pero su contenido se debe arreglar de una manera específica.

13.3.2 Funcionamiento

Para comprobar el correcto funcionamiento del módulo de apache, es necesario parar el servidor web apache-tomcat y volverlo a arrancar, tal y como se indicó anteriormente. Si todo funciona correctamente, la salida tras ejecutar la url http://localhost:8080/axis2 debe ser la siguiente:

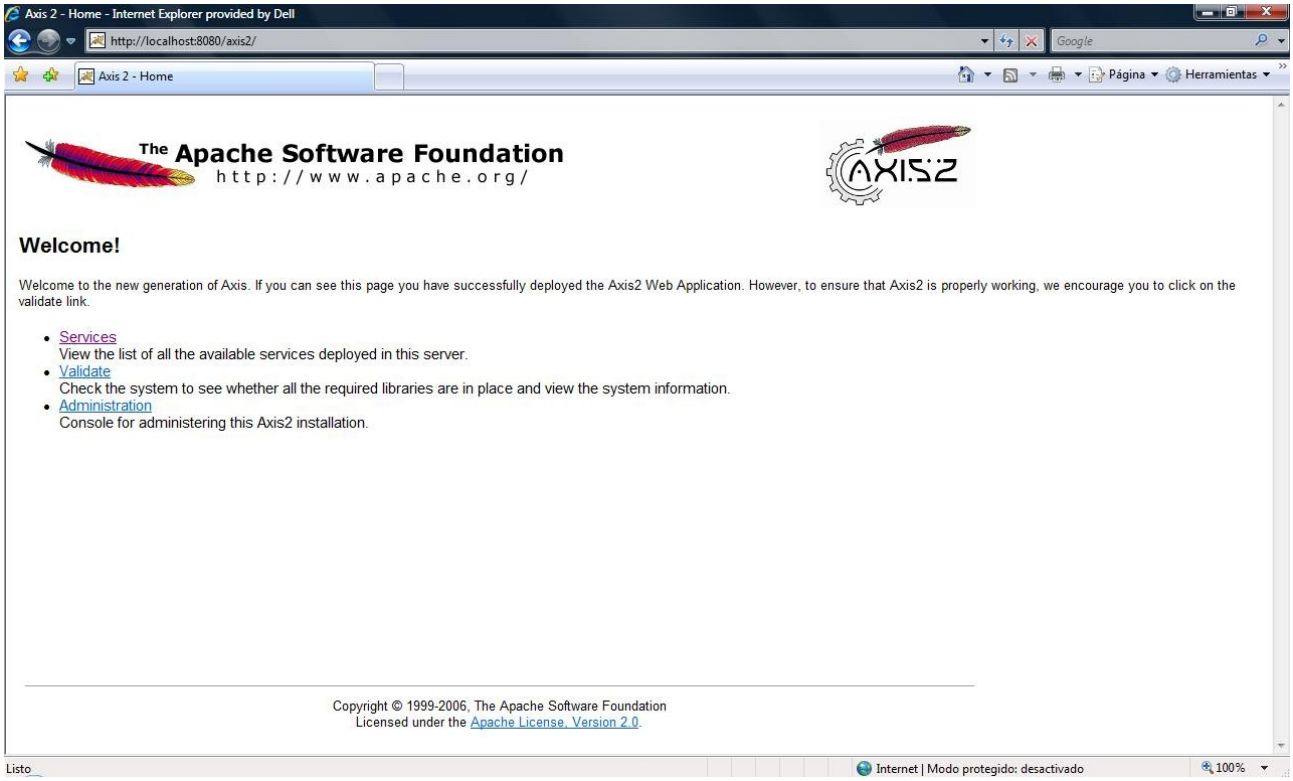
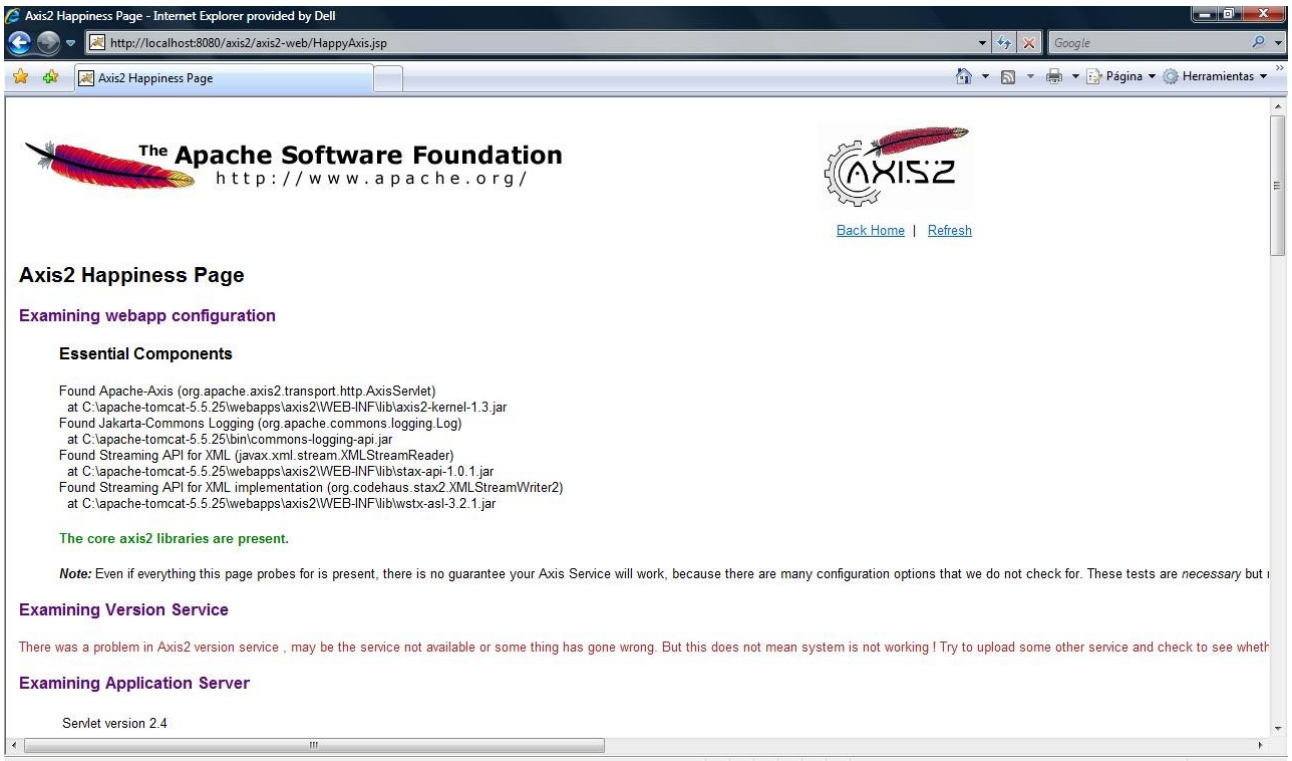


Figura 14.2. Funcionamiento de Axis2

Para asegurarnos que todo se ha desarrollado con normalidad, o por si hubiera ido algo mal, es posible ver la página llamada *Happiness Page,* que indicará cualquier error encontrado. Si todo ha sido instalado correctamente, su salida deberá ser la siguiente:



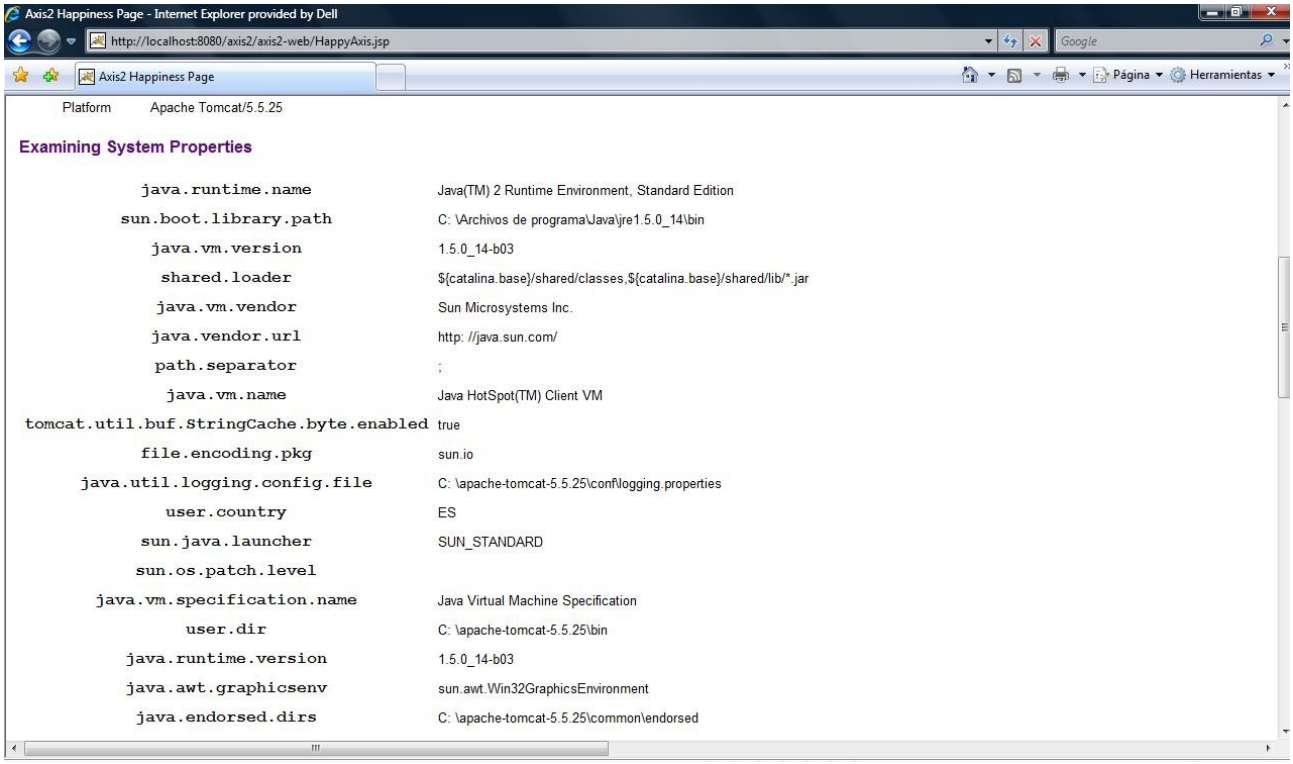


Figura 14.3. Correcto funcionamiento de axis2

13.3.3 Despliegue de servicios

Para explicar el desarrollo de los servicios web, es necesario partir de una clase escrita en lenguaje Java y que posteriormente pasará a convertirse en el servicio web empleado. Como ejemplo, citaremos la siguiente clase java, correspondiente a la clase Login.java:

```
public class Login{


      public String login (String username, String password) throws Exception {


            ResourceBundle prop =
ResourceBundle.getBundle("resources.conexion");

            String host = prop.getString("conexion.host");
            String puerto=prop.getString("conexion.port");
            int port = Integer.parseInt(puerto);
            UtilesUrl utiles= new UtilesUrl();
            String uri=utiles.obtieneUrlServidor(username);
            String idsesion=username+"+"+password;
            byte[] data =idsesion.getBytes();

            Base64Coder code = new Base64Coder();
            byte[] codificado =code.encodeBase64(data);

            String texto_codificado = new String(codificado,"UTF8");
```

```
            DAVConnection dc=DAVConnectionPool.getDAVConnection(host, port,
false);

            dc.setUsername(username);
            AuthorizationInfo ai= new AuthorizationInfo(host,
                        port,
                        "Basic",
                        "KTWebDav Server",
                        Codecs.base64Encode(username + ":" + password));
            dc.setAuthorization(ai.toString());
            PropFindMethod method;

            method=new PropFindMethod(uri);

            dc.execute(method);

            int status=method.getStatus();
            if(status >= 200 && status < 300)
                  texto_codificado=texto_codificado;
            else
                  texto_codificado=String.valueOf(-1);


            return texto_codificado;
      }
}
```

Lo siguiente será generar el .wsdl. La herramienta Java2WSDL de Axis2 se puede utilizar para generar un WSDL. Para generar un archivo WSDL a partir de una clase de Java, es necesario realizar los pasos siguientes:

1. Compilar la clase Java.
2. Generar el WSDL utilizando el comando:

```
%AXIS2_HOME%/bin/java2wsdl -cp . -cn org.samples.Login -of Login.wsdl
```

Una vez que haya generado el archivo WSDL, se puede realizar los cambios que se necesite. La estructura de este servicio será como sigue:

```
- Login
   - META-INF
     - services.xml
   - lib
   - org
     - samples
       - Login.class
```

Aquí, el nombre del servicio es Login, que se especifica en el archivo services.xml y se corresponde con la carpeta a nivel superior de este servicio. Las clases compiladas Java se ponen por debajo de ella en su lugar apropiado basado en el nombre del paquete. El directorio lib contiene cualquier archivo JAR específico al servicio necesario para que el servicio se ejecute (ninguno en este caso)

además de los ya almacenados en el archivo WAR de Axis2 y los directorios JAR comunes del contenedor de servlets. Finalmente, el directorio META-INF contiene cualquier información adicional sobre el servicio que Axis2 necesite para ejecutarlo correctamente. El archivo services.xml define el servicio en sí mismo y vincula los Java class a él:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This file was auto-generated from WSDL -->
<!-- by the Apache Axis2 version: 1.3  Built on : Aug 10, 2007 (04:45:47 LKT)
-->
<serviceGroup>
    <service name="Login">
        <messageReceivers>
            <messageReceiver mep="http://www.w3.org/ns/wsdl/in-out"
class="samples.LoginMessageReceiverInOut"/>
        </messageReceivers>
        <parameter name="ServiceClass">samples.LoginSkeleton</parameter>
        <parameter name="useOriginalwsdl">true</parameter>
        <parameter name="modifyUserWSDLPortAddress">true</parameter>
        <operation name="login" mep="http://www.w3.org/ns/wsdl/in-out">
            <actionMapping>urn:login</actionMapping>
            <outputActionMapping>urn:loginResponse</outputActionMapping>
            <faultActionMapping
faultName="Exception">urn:loginException</faultActionMapping>
        </operation>
    </service>
</serviceGroup>
```

En el archivo services.xml anterior, el servicio es definido junto con los tipos messageReceiver relevantes para los diversos patrones del intercambio de mensajes. El directorio META-INF es también la ubicación para cualquier archivo WSDL personalizado que se preponga incluir para esta aplicación.

El servicio puede ser desplegado simplemente tomando esta jerarquía de archivos y copiándola al directorio webapps/axis2/Web-INF/services del motor de servlets. (el archivo WAR de Axis2 debe estar previamente instalado en el motor de servlets.) Esto se conoce como formato *exploded*. También se pueden comprimir los archivos en un archivo *.aar, similar a a un archivo *.jar, y colocar el archivo *.aar directamente en el directorio webapps/axis2/Web-INF/services del motor de servlets. Ahora que entiende lo que estamos intentando lograr, estamos casi listos para comenzar a construir. A continuación, veremos la forma de crear el servicio.

Para crear el servicio, vamos a utilizar el framework ADB (Axis2 Databinding Framework). Para ello, es necesario generar el skeleton de la siguiente manera:

```
%AXIS2_HOME%/bin/WSDL2Java -uri Login.java -p samples.adb -d adb -s -ss -sd
-ssi -o build/service
```

El significado de los parámetros anteriores, ya fue descrito en el capítulo 8 de esta memoria.

El skeleton generado habrá que completarlo del siguiente modo:

```
public class LoginSkeleton implements LoginSkeletonInterface {
    /**
     * Auto generated method signature
     * @param login0
     */
    public org.skunk.dav.client.LoginResponse login(
        org.skunk.dav.client.Login login0) throws ExceptionException0 {
        //TODO : fill this with the necessary business logic


            ResourceBundle prop =
ResourceBundle.getBundle("resources.conexion");

            String host = prop.getString("conexion.host");
            String puerto=prop.getString("conexion.port");
            int port = Integer.parseInt(puerto);
            UtilesUrl utiles= new UtilesUrl();
String uri=null;
      try{
            uri=utiles.obtieneUrlServidor(login0.getUsername());
            }catch(Exception e){

            }

            String idsesion=login0.getUsername()+"+"+login0.getPassword();
            byte[] data =idsesion.getBytes();

            Base64Coder code = new Base64Coder();
byte[] codificado=null;
try{
            codificado =code.encodeBase64(data);
}catch(Exception e){
}
            String texto_codificado=null;
try{
            texto_codificado = new String(codificado,"UTF8");
}catch(Exception e){
}

            DAVConnection dc=DAVConnectionPool.getDAVConnection(host, port,
false);

            dc.setUsername(login0.getUsername());
            AuthorizationInfo ai= new AuthorizationInfo(host,
                        port,
                        "Basic",
                        "KTWebDav Server",
                        Codecs.base64Encode(login0.getUsername() + ":" +
login0.getPassword()));
            dc.setAuthorization(ai.toString());
            PropFindMethod method;

            method=new PropFindMethod(uri);
            try{
            dc.execute(method);
}catch(Exception e){
}

            int status=method.getStatus();
```

```
            if(status >= 200 && status < 300)
                  texto_codificado=texto_codificado;
            else
                  texto_codificado=String.valueOf(-1);
org.skunk.dav.client.LoginResponse res = new
org.skunk.dav.client.LoginResponse();
res.set_return(texto_codificado);
return res;

            //return texto_codificado;

    }
}
```

El último paso es completar la creación del servicio. Para ello, es necesario tener instalado el módulo pache ant. Dicha distribución, puede descargarse en el enlace http://ant.apache.org/bindownload.cgi. Una vez ahí, nos descargaremos la distribución .zip. Para instalarlo, simplemente será necesario descomprimirlo en alguna ubicación, por ejemplo en C:\ant, y crear la siguiente variable de entorno ANT_HOME, que apunte al directorio C:\ant\bin. Una vez instalado, basta con ejecutar el siguiente comando en el directorio build/service generado con anterioridad:

```
%ANT_HOME%/ant jar.server
```

Si todo va bien, deberá verse el mensaje BUILD SUCCESSFUL en la ventana de comandos, y el archivo de Login.aar en el directorio build/service/build/lib. Lo siguiente será copiar ese archivo al directorio de webapps/axis2/Web-INF/services del motor de servlets.

Lo siguiente será arrancar el servidor web y comprobar que el servicio ha sido desplegado correctamente ejecutando:

```
http://localhost:8080/axis2/services/Login?wsdl
```

Hasta aquí, todo lo relativo al despliegue de los servicio web.

13.4 NetBeans IDE 6.0.1 con Mobility Pack

13.4.1 Introducción

NetBeans es una plataforma que permite el desarrollo de aplicaciones usando el lenguaje Java y que contiene un entorno de desarrollo integrado llamado IDE desarrollado usando la Plataforma NetBeans.

El **NetBeans IDE** es un IDE de código abierto escrito completamente en Java usando la plataforma NetBeans. El NetBeans IDE soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles). Entre sus características se encuentra un sistema de proyectos basado en Ant y control de versiones, entre otros. La versión actual es la 6.1, aunque para el desarrollo del proyecto hemos usado la versión 6.0.1.

Entre los paquetes adicionales de NetBeans, se encuentra el pack de movilidad llamado Mobility Pack, que permite desarrollar aplicaciones que se ejecutan en teléfonos móviles. Puede ser usado para escribir, probar, y depurar aplicaciones para la plataforma Java ME tecnología existente en dispositivos móviles.

El Pack de Movilidad, permite desarrollar las aplicaciones con dos configuraciones distintas de la plataforma J2ME: CDC y CLDC, ya estudiadas en el capítulo 4 de esta memoria.

Entre sus numerosas ventajas, destacan:

- Es de código abierto
- Consta de un editor de texto
- Posee como emulador por defecto el Sun Java Wireless Toolkit para CLDC, que permite desarrollar aplicaciones inalámbricas J2ME basadas en J2ME y CLDC, diseñadas para ejecutarse en dispositivos móviles y PDAS .
- Permite desarrollar clientes de servicios web en teléfonos inalámbricos compatibles con la interfaz JSR-172.

13.4.2 Instalación

El software puede descargarse de la siguiente dirección Web: http://download.netbeans.org/netbeans/6.0/final/. Una vez ahí, elegiremos la opción *ALL,* que posee todos los módulos extras.

El ejecutable que se descarga netbeans-6.0.1-ml-windows.exe, es fácil de instalar. Aparecerá el asistente de instalación al que simplemente hay que indicarle la ruta donde queremos que se instale. Tras la instalación, se abrirá el programa automáticamente:

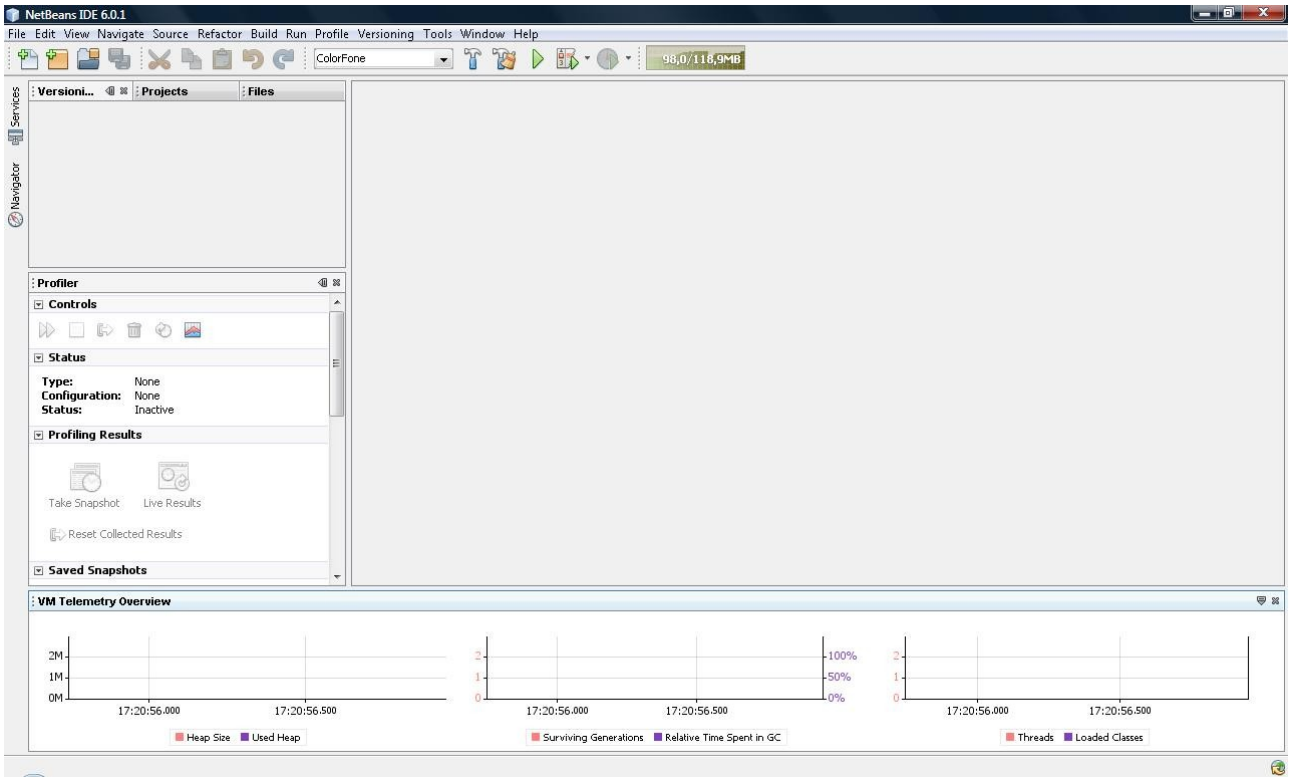
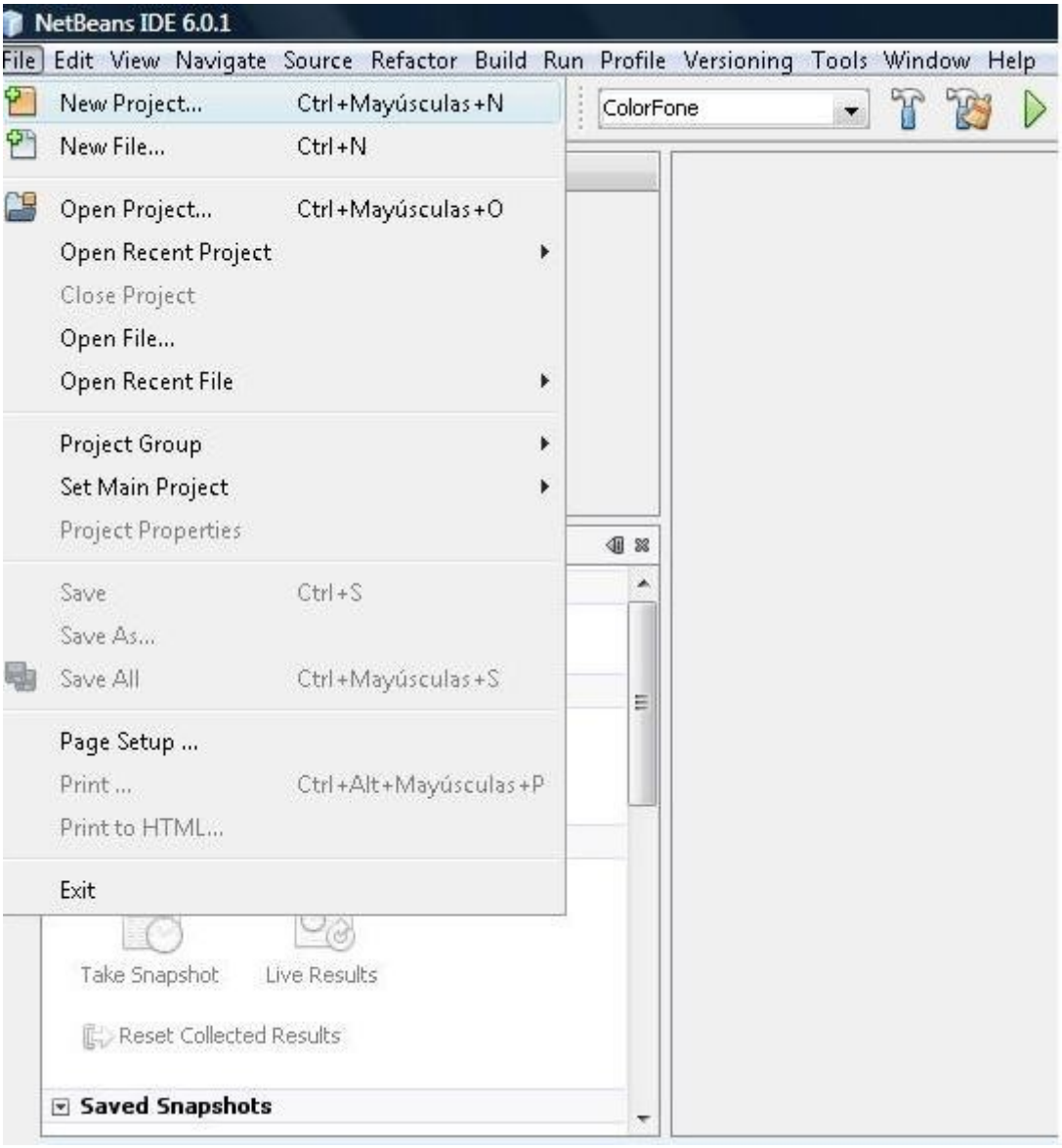


Figura 14.4. NetBeans 6.0.1

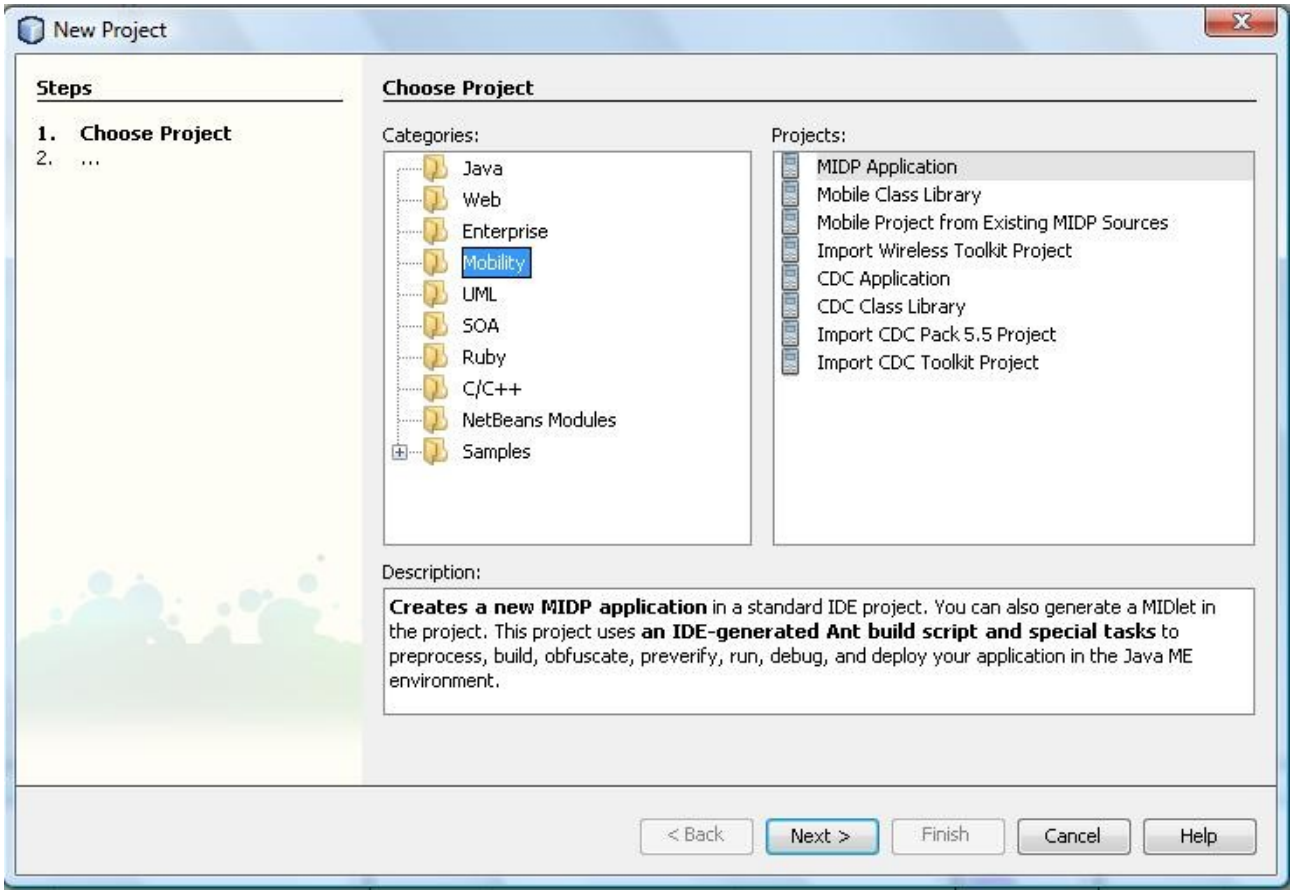
13.4.3 Funcionamiento

Para comenzar desde cero y crear un nuevo proyecto, seguiremos los siguientes pasos:

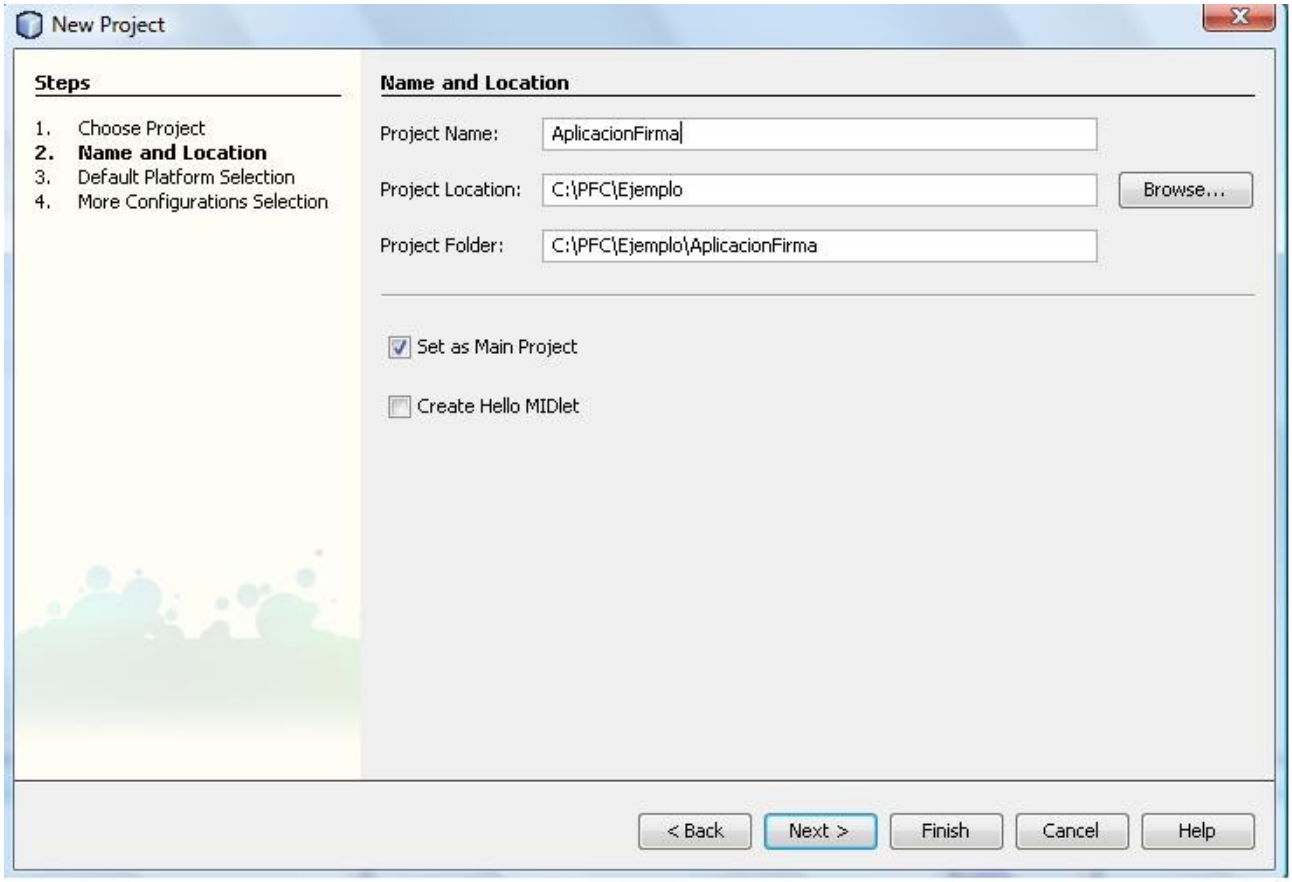
1. Seleccionamos File -> New Project



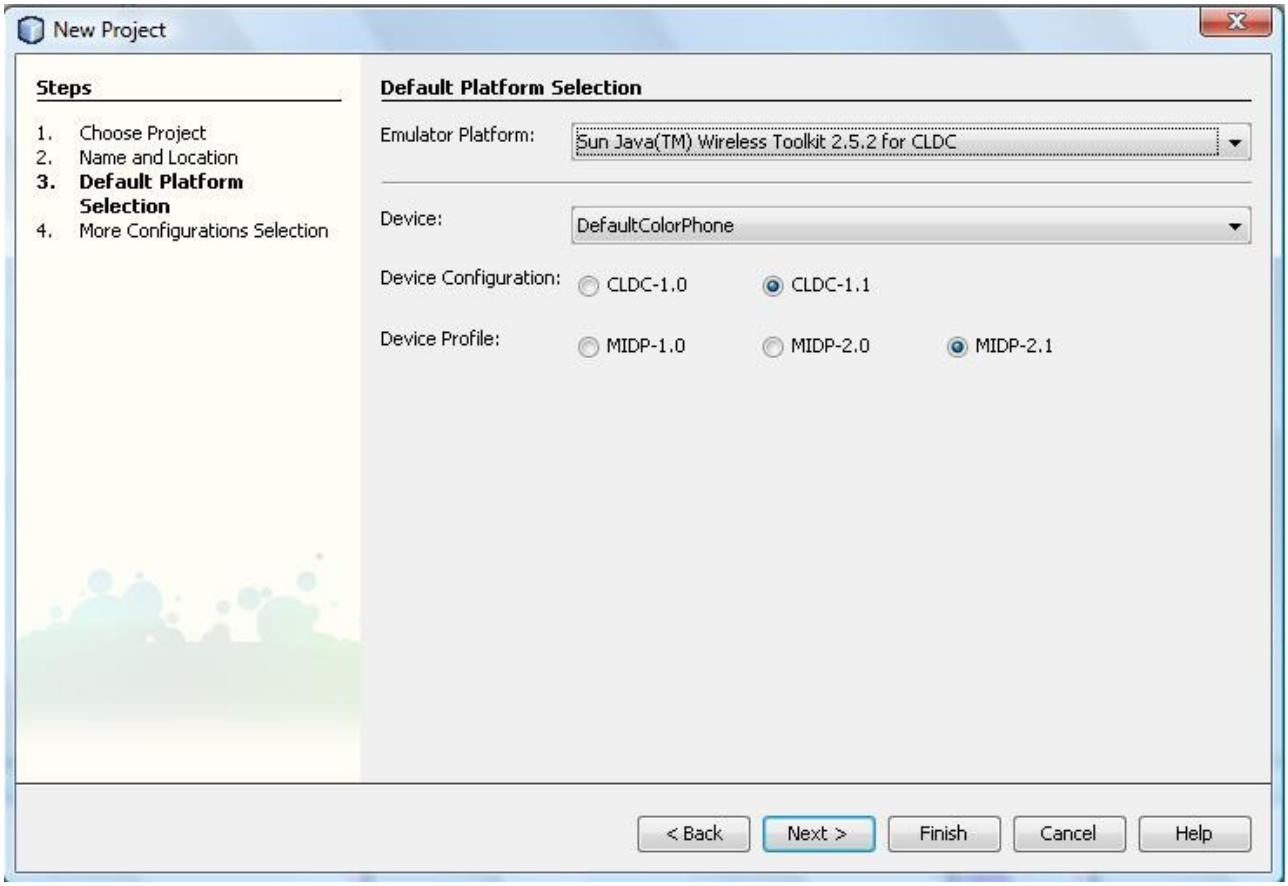
2. A continuación especificamos que se trata de una aplicación móvil



3. Escribimos los parámetros necesarios, como nombre del proyecto y demás.



4. Por último, seleccionamos la configuración y los perfiles que queramos. En nuestro caso, dejaremos que aparezca la de por defecto.



Hasta aquí, todo lo relativo a la plataforma NetBeans IDE.