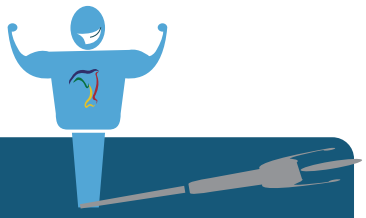
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

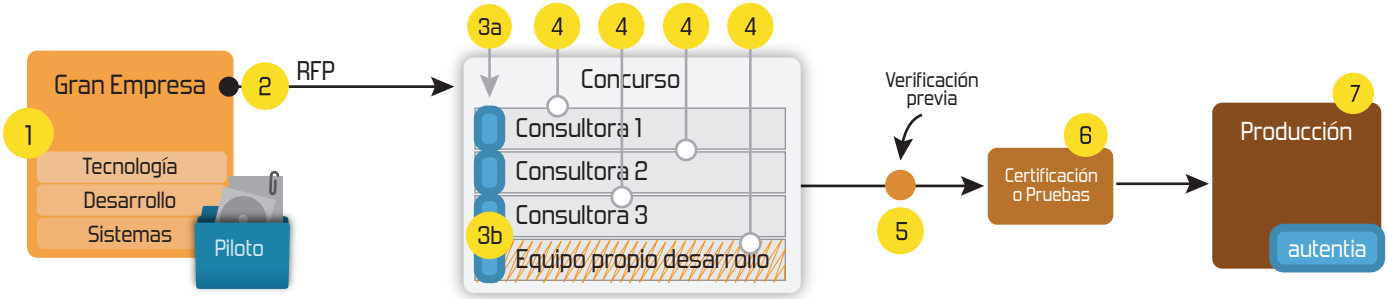
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Despliegue de EJBs en J2EE 1.3.1

A partir de ahora, vamos a subir un poco el nivel .... comenzando a comentar buenas técnicas y prácticas en el desarrollo de EJBs.

Para ello, debemos crear un entorno adecuado de pruebas donde, de un modo sencillo, podamos aplicar nuestros cambios.

El desarrollo de EJBs no es algo complicado.... aunque lo es más dificil es probar su funcionamiento, optimización y depuración.

Vamos a crear un EJB desde cero y lo vamos a desplegar en el servidor de aplicaciones de referencia de Sun.

Para instalar el servidor, lo único que tenemos que hacer es descargárnoslo y establecer correctamente las variables de entorno. Para ello, podéis ver otro de nuestros tutoriales.

Definición de interfaces Home y Remoto

En principio vamos a crear un EJB de sesión sin estado por ser más sencillo.

Lo primero que debemos hacer es crear los interfaces necesarios para el desarrollo de un EJB.

En este caso, hace falta un interfaz, llamado Home, que establece como se deben crear e inicializar los objetos. Este interfaz debe heredar de EJBHome

```
/*
 * iFacturasHome.java
 *
 * Created on 20 de septiembre de 2003, 9:41
 */

package ejbfacturas;

/**
 *
 * @author  Roberto Canales
 */

import java.rmi.*;
import javax.ejb.*;
import java.io.*;

/**
 * Interfaz Home para definir los metodos de creación de nuestro EJB
 *
 */
public interface iFacturasHome extends EJBHome
{
```

```
    public iFacturas create() throws CreateException, RemoteException;
}
```

Posteriormente se define un interfaz con la definición de los métodos que tendrá el servicio que deseamos construir. Este es el llamado interfaz Remoto y debe heredar de EJBObject

Nuestro EJB, deberá implementar este interfaz... es decir, tener el código de estos métodos.

En el desarrollo de EJBs, el EJB no se define con este interfaz, sino que es en tiempo de ensamblaje y despliegue donde se realiza la validación. Esto no nos proporciona ningura garantía de funcionamiento .... por lo que vamos a usar otra técnica para asegurarnos que si nuestro código compila..... entonces no dará problemas al desplegar.

Creamos un interfaz simple que heredará el interfaz remoto y que implementará en Bean

```
/*
 * iFacturasContrato.java
 *
 * Created on 20 de septiembre de 2003, 10:11
 */

package ejbfacturas;

import java.rmi.*;
import javax.ejb.*;
import java.util.*;
import facturascomun.*;

/**
 *
 * @author  Administrator
 */
public interface iFacturasContrato {

   /**
    * Función que retorna el numero de facturas en la base de datos
    */
    public int recuperaNumeroFacturas() throws RemoteException;
    public Vector recuperaIdFacturas() throws RemoteException;
    public Factura recuperaFacturaPorId(int pId) throws RemoteException;
}
```

Ahora escribimos el Remove

```
/*
 * ifacturas.java
 *
 * Created on 20 de septiembre de 2003, 9:40
 */

package ejbfacturas;

/**
 *
 * Interfaz remoto a nuestro EJB de Facturas
 *
 * @author  Roberto Canales
 */
import java.rmi.*;
import javax.ejb.*;
import facturascomun.*;
import java.util.*;

public interface iFacturas extends EJBObject, iFacturasContrato
{
    // todos los metodos estan definidos en iFacturasContrato
}
```

Y por último el código de nuestro Bean

```
/*
 * iFacturasBean.java
 *
 * Created on 20 de septiembre de 2003, 9:41
 */

package ejbfacturas;

/**
 *
 * @author Roberto Canales
 */
```

```
import java.rmi.*;
import javax.ejb.*;
import java.util.*;

import facturascomun.*;


public class iFacturasBean extends Object implements SessionBean, iFacturasContrato
{

   public void ejbCreate() throws CreateException {
   }

   public void ejbActivate() {
   }

   public void ejbPassivate() {
   }

   public void ejbRemove() {
   }

   public void setSessionContext(SessionContext context) {
   }

   /**
    * Retorna el numero de facturas en base de datos
    */
   public int recuperaNumeroFacturas()
   {
      // Retornamos un valor por defecto
      return 1;
   }

   /**
   * Recupera las lista de facturas de base de datos
   */
   public Vector recuperaIdFacturas()
   {
      Vector vIdFacturas = new Vector();
      vIdFacturas.addElement(new Integer(0));
      return vIdFacturas;
   }

   /**
    * Recupera una factura concreta
    */
   public Factura recuperaFacturaPorId(int pId)
   {
      return new Factura(0,"AdictosAlTrabajo","Desarrollo creativo","54332211s", "Calle Desconocida", "20/10/2203", 1234.12);
   }
}
```

Otra buena práctica en el desarrollo de aplicaciones consiste en utilizar clases para agrupar los datos que deben viajar entre distintas máquinas o instancias de servidores de aplicaciones.

En nuestro caso, hemos creado una clase llamada **Factura**...

```
/*
 * Factura.java
 *
 * Created on 20 de septiembre de 2003, 9:50
 */

package facturascomun;

import java.io.*;

/**
 * La clase Factura es la clase de transito que nos permite manejar facturas
 * de un modo sencillo en nuestra aplicación
 * @author  Administrator
 */
public class Factura implements Serializable
{
   int iId;
   String sTitular;
   String sConcepto;
   String sCif;
   String sDireccion;
   String sFecha;
   double dCandidad;


   /** Constructor por defecto */
   public Factura() {

   }

   /**
    * Constructor para inicializar valores básicos
    */
```

```
   public Factura(int pId, String pTitular, String pConcepto, String pCif, String pDireccion, String pFecha, double pCantidad)
   {
      iId        = pId;
      sTitular    = pTitular;
      sConcepto   = pConcepto;
      sCif        = pCif;
      sDireccion  = pDireccion;
      sFecha      = pFecha;
      dCandidad   = pCantidad;
   }
}
```

Servlet de Prueba

Ahora, vamos a crear un servlet básico (aunque poco óptimo) para probar de un modo sencillo si nuestro EJB funciona correctamente.

```
/*
* servletVerificadorFacturas.java
*
* Created on 20 de septiembre de 2003, 10:48
*/

import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;

import facturascomun.*;
import ejbfacturas.*;

import javax.naming.*;
import ejbfacturas.*;
import javax.rmi.*;

import java.util.*;

/**
*
* @author Roberto Canales
* @version
*/
public class servletVerificadorFacturas extends HttpServlet {

/** Initializes the servlet.
*/
public void init(ServletConfig config) throws ServletException {
super.init(config);

}

/** Destroys the servlet.
*/
public void destroy() {

}

private static InitialContext getInitialContext() throws NamingException
{
Hashtable env = new Hashtable();
env.put
(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContextFactory");
env.put("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");
env.put(Context.PROVIDER_URL, "iiop://127.0.0.1:1050");

return new InitialContext(env);
}


/** Processes requests for both HTTP <code>GET</code> and <code>POST</code> methods.
* @param request servlet request
* @param response servlet response
*/
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();


out.println("<html>");
out.println("<head>");
out.println("<title>Prueba de EJB Factura</title>");
out.println("</head>");
out.println("<body>");

Context contexto = null;
iFacturasHome miHome = null;


try
{
contexto = getInitialContext();
Object objetoGenerico = contexto.lookup("java:comp/env/ejb/iFacturasHome");
miHome = (iFacturasHome)PortableRemoteObject.narrow(objetoGenerico,iFacturasHome.class);
```

```
iFacturas ejbGestorFacturas = miHome.create();
depura(out,"Número de facturas en BBDD = " + ejbGestorFacturas.recuperaNumeroFacturas());
}
catch(Exception e)
{
depura(out,"Error al gestionar Facturas\n" + e.getMessage());
e.printStackTrace();
}

out.println("</body>");
out.println("</html>");

out.close();
}

/**
*  Funcion basica de depuracion
*/
void depura(PrintWriter pOut, String sCadena)
{
pOut.println("Depuracion: " + sCadena);
System.out.println("Depuracion: " + sCadena);
}

protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
processRequest(request, response);
}

protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
processRequest(request, response);
}

}
```

Generador de descriptores

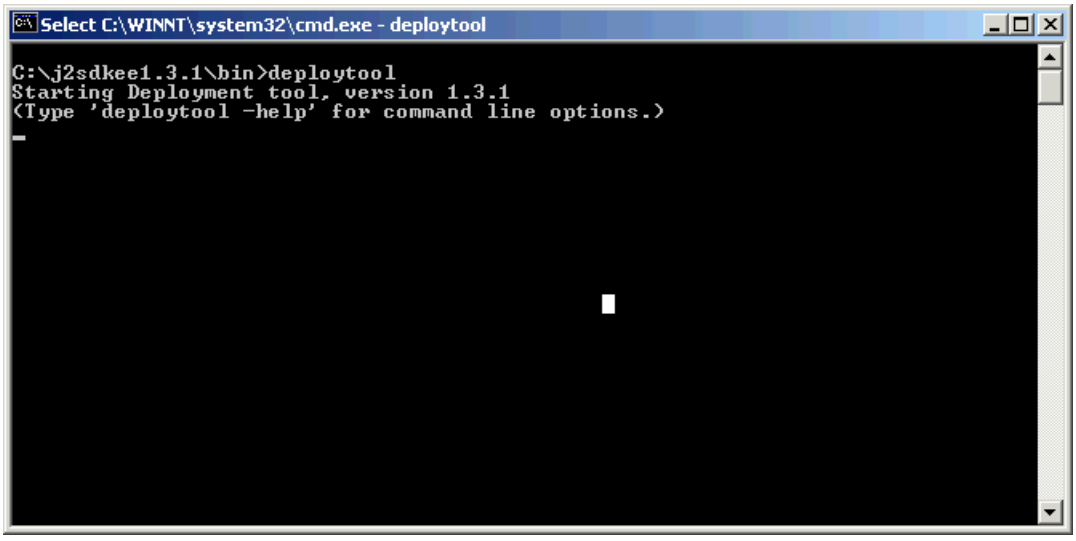
Para poder instalar un EJB en un servidor de aplicaciones, hay que crear unos ficheros XML que describen que es lo que pretendemos desplegar y algunos parámetros de configuración.

Estos ficheros deberían ser iguales para cualquier servidor de aplicaciones anque ... como siempre pasa ..... esto luego no es cierto y hay que introducir variaciones particulares para cada servidor.

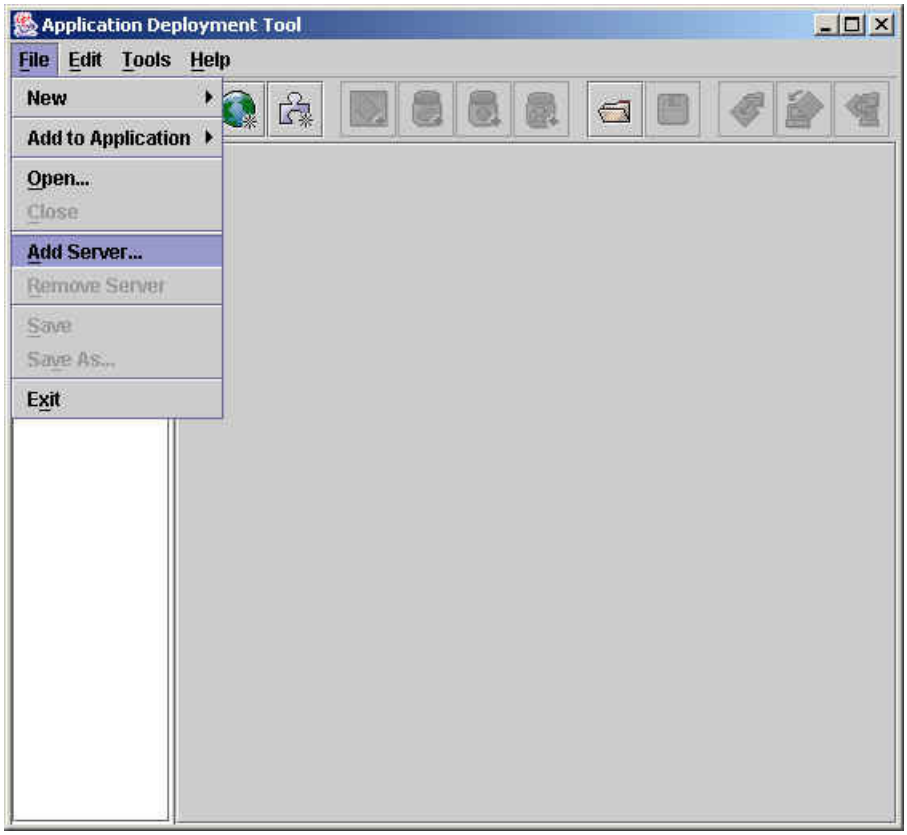
Nuestro servidor de aplicaciones trae una herramienta gráfica para ayudarnos a hacerlo (generar los XML) y empaquetar (en ficheros comprimidos) e instalar la aplicación dentro del servidor.

Normalmente, luego no lo haremos así sino que trabajaremos con un repositorio de código (porque mucha gente trabaja en paralelo sobre el mismo código) y se compilará y empaquetará con ANT (aunque reuitilzaremos los XML que genere esta herramienta)

Vamos a nuestro directorio y ejecutamos la utilidad **deploytool**

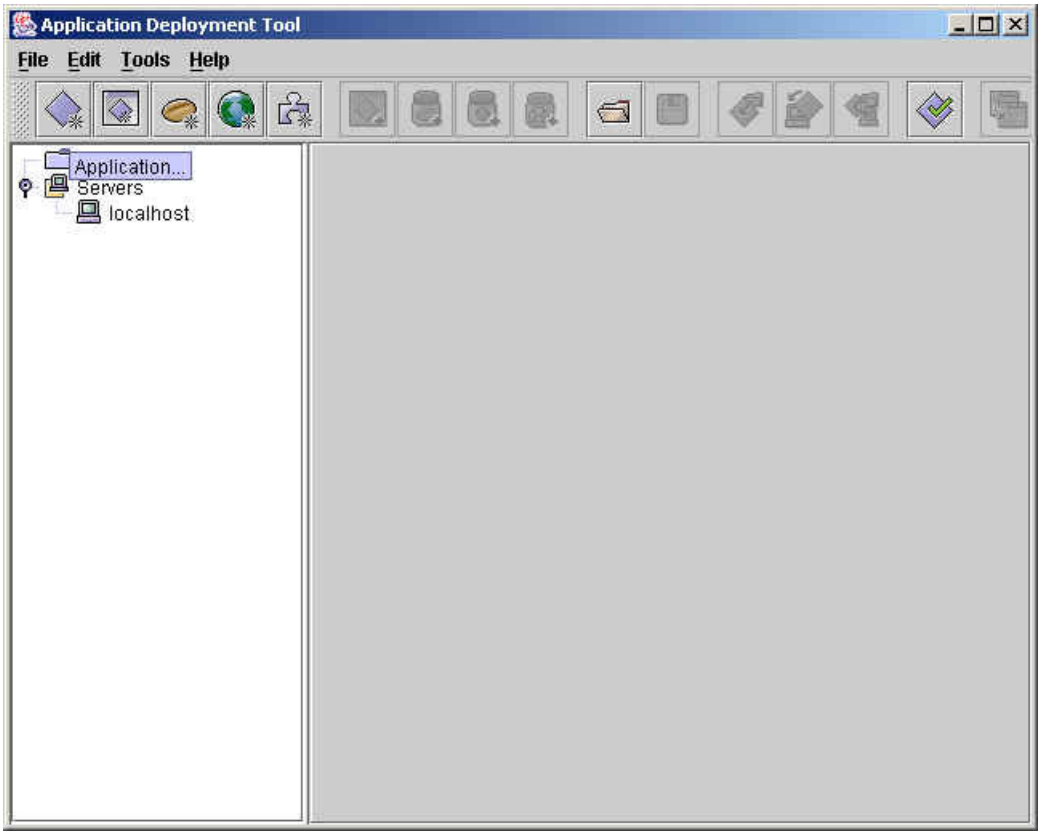


Añadimos un nuevo servidor a la lista de servidores que queremos administrar, en este caso, el servidor local



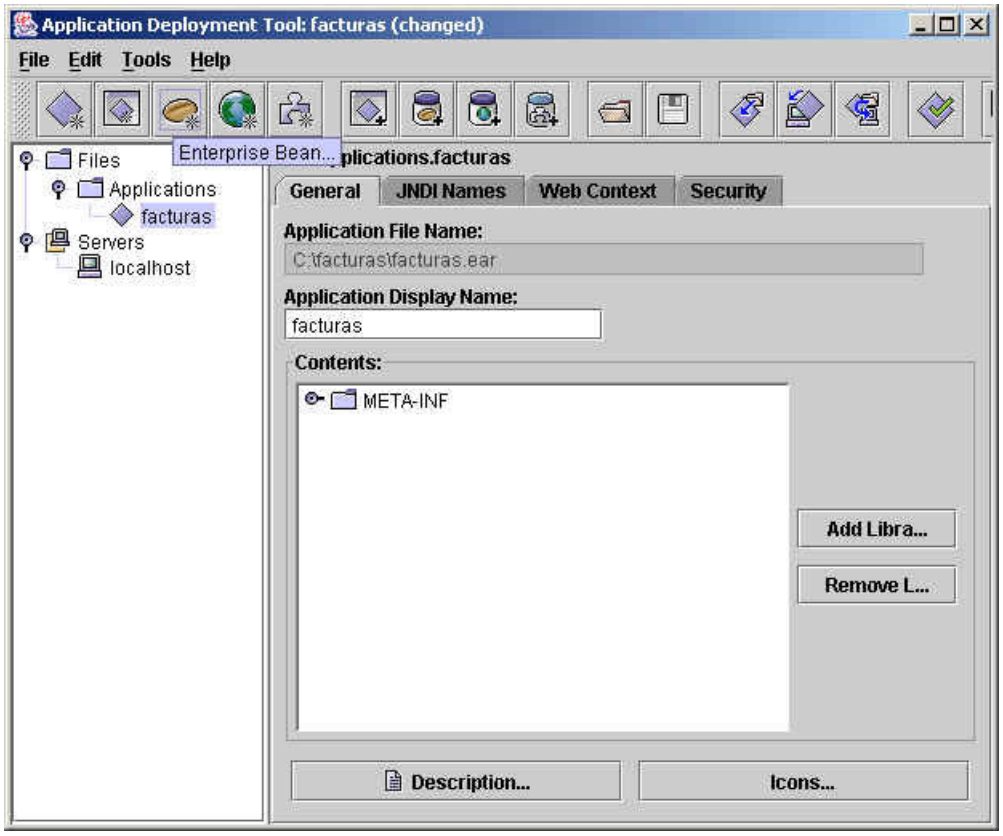
Añadimos el servidor **localhost**

Pinchamos el primer botón, es decir, creamos una nueva aplicación



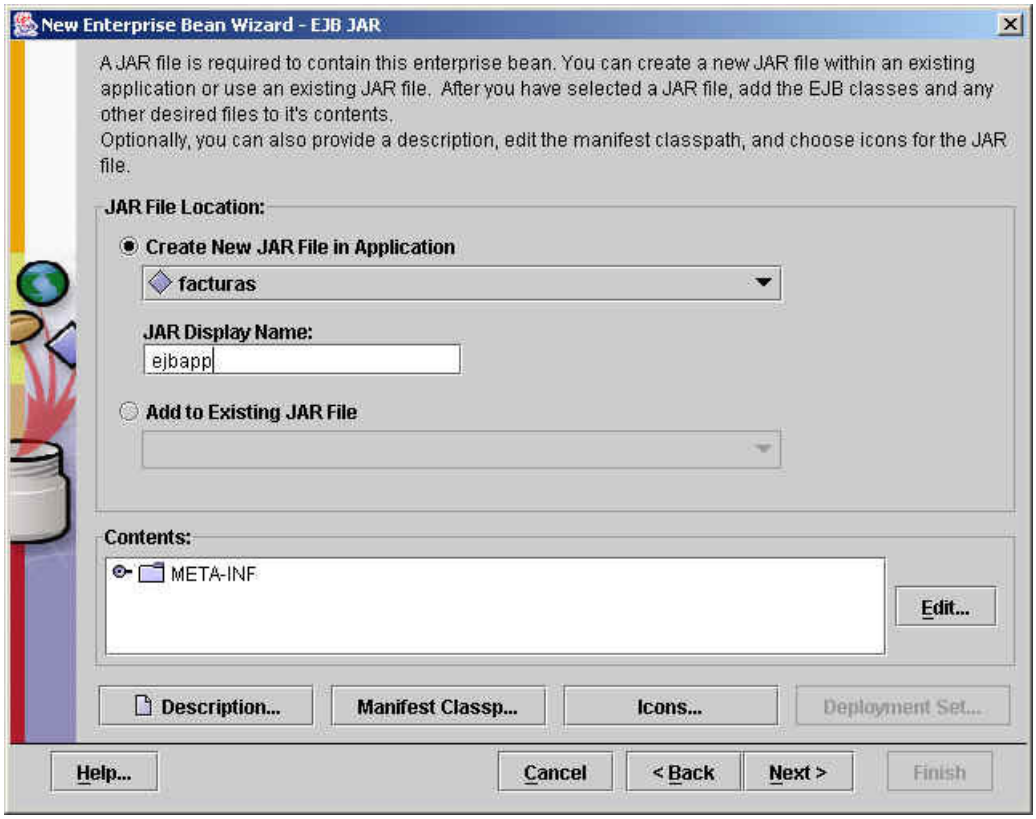
Le asignamos un trayecto al fichero

Seleccionamos el botón para añadir un nuevo EJB

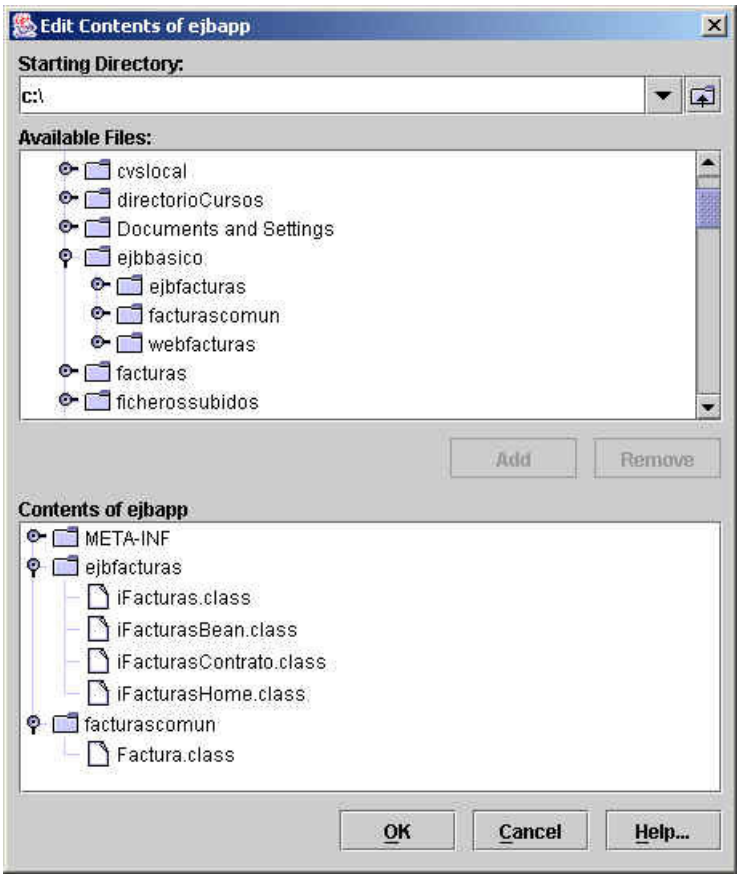


Vemos la primera ventana de instrucciones y seguimos

Le asignamos un nombre al contenedor de nuestro EJB y pinchamos el **botón de editar,** para añadir las clases necesitadas

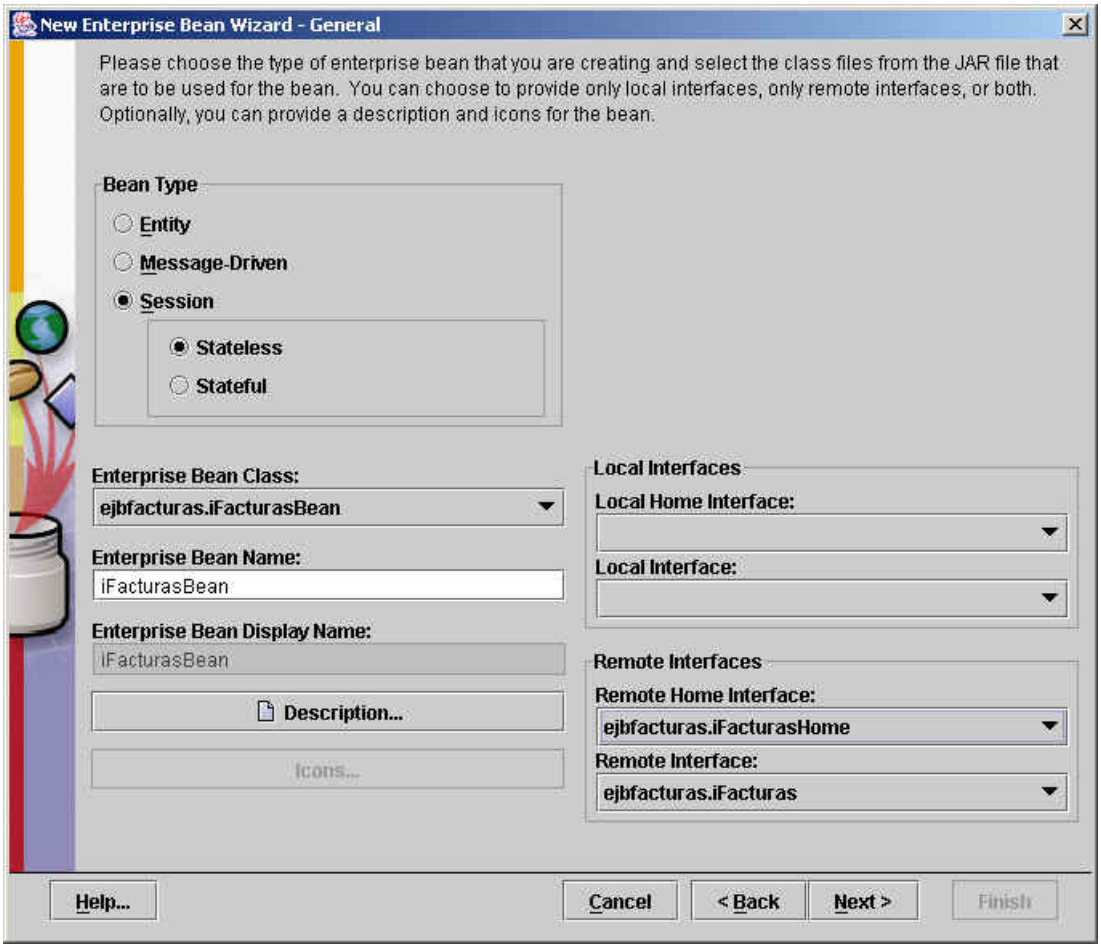


Seleccionamos, en el sistema de ficheros, nuestos archivos (todos los .class de nuestra aplicación)

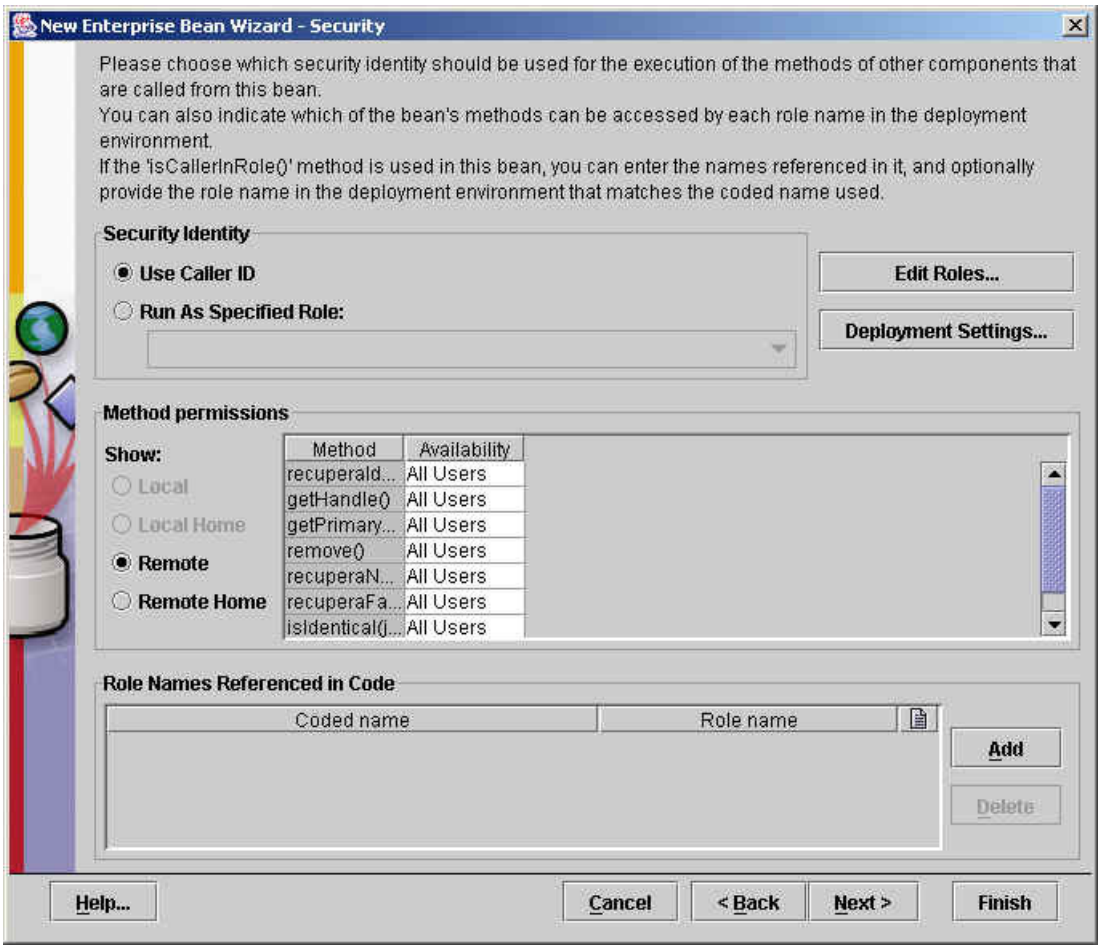


Seleccionamos el EJB (iFacturasBean) de tipo Session sin estado.

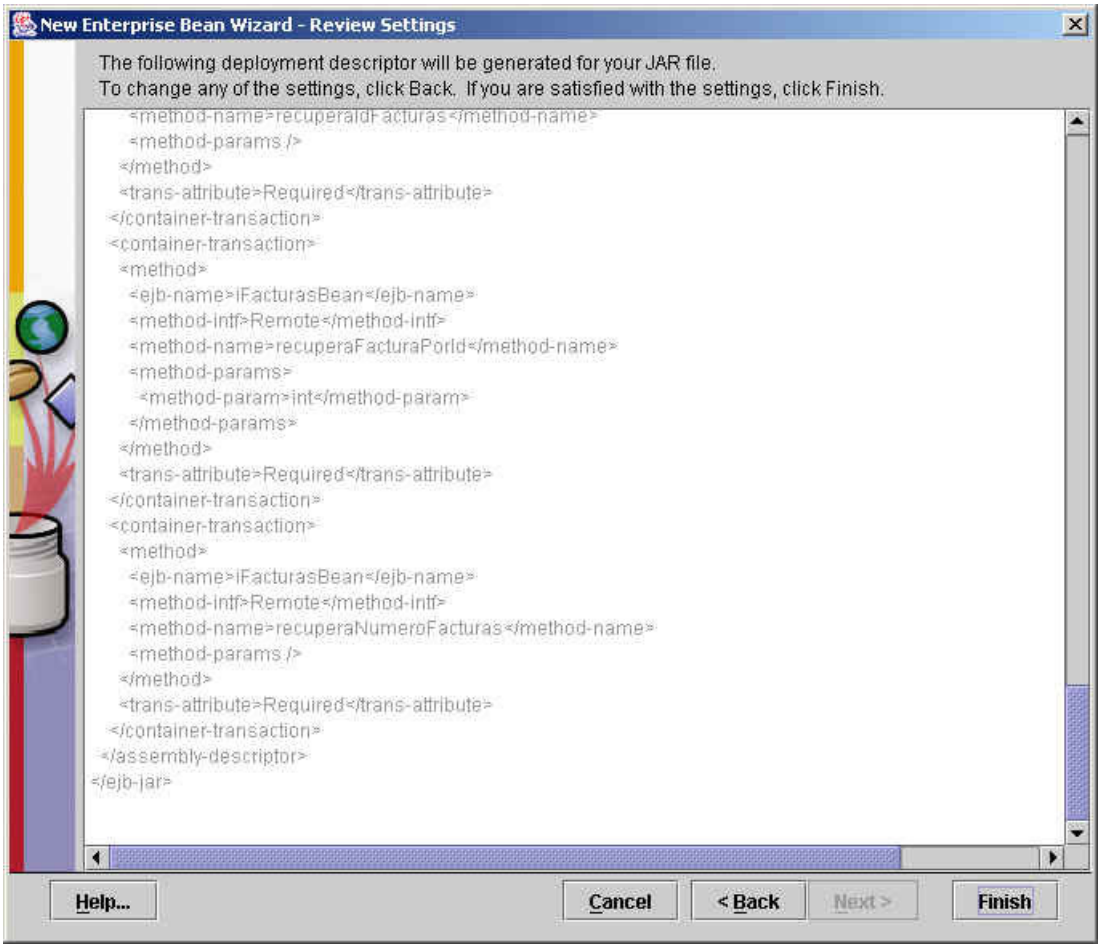
Tambien seleccionamos las clases del interfaz Home y Remoto



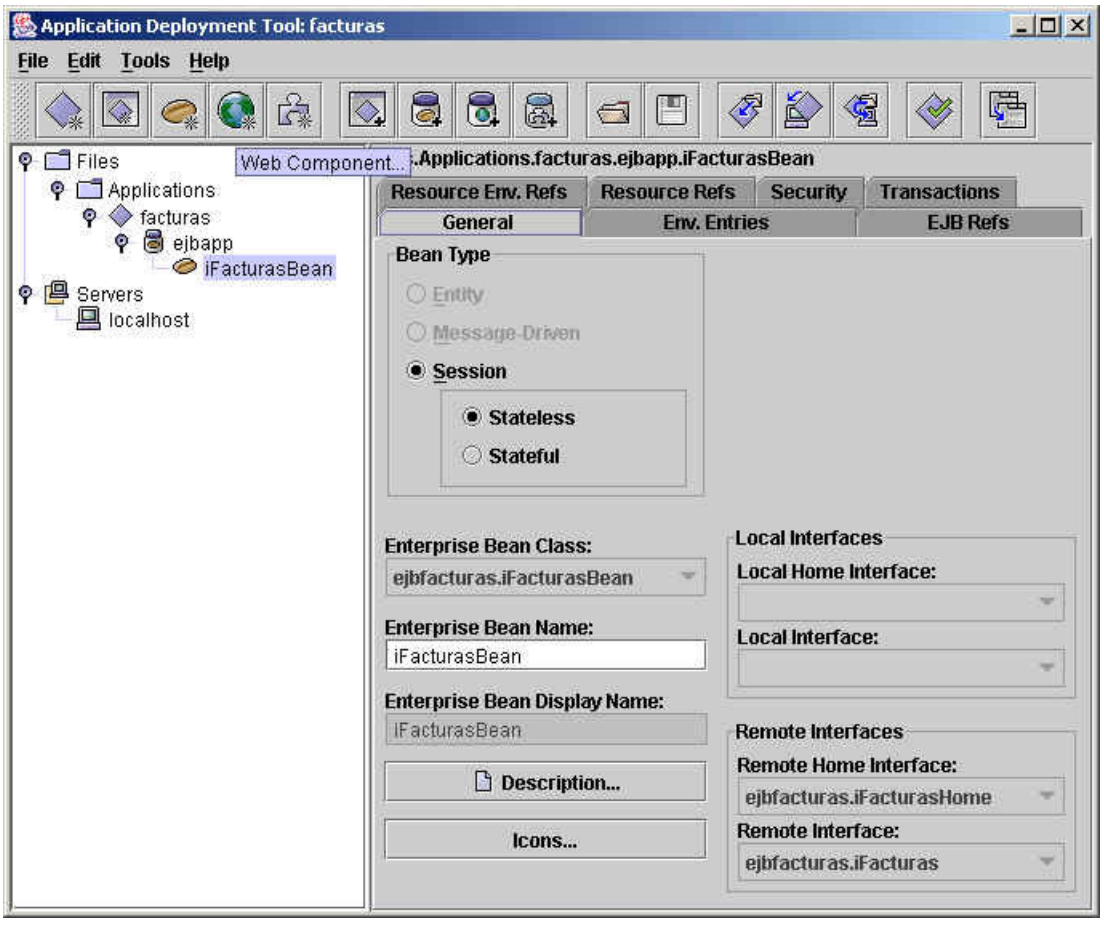
Avanzamos y seleccionamos el resto de los valores por defecto



Y ya hemos terminado

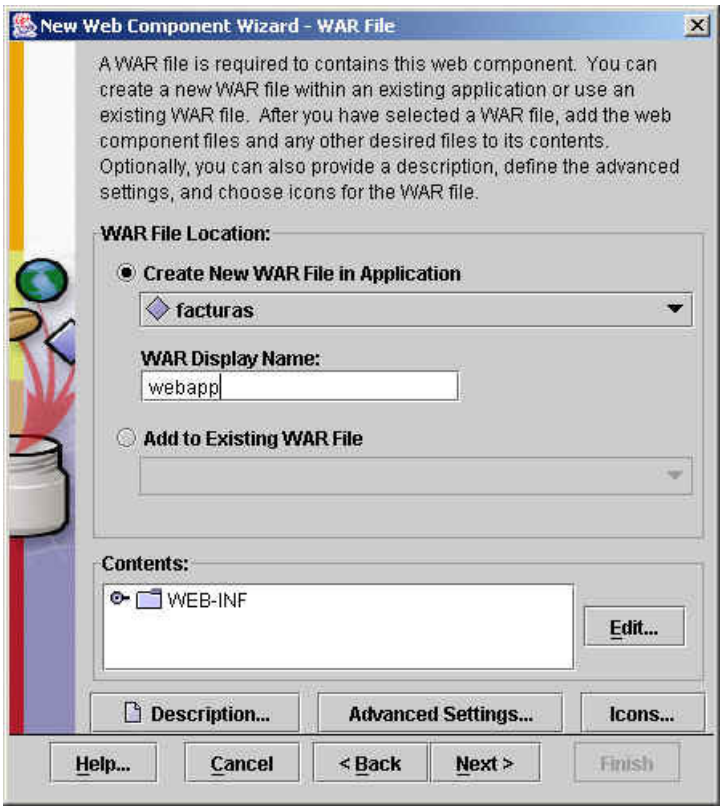


Ahora añadimos un nuevo contenedor Web, una llamada Web-app, para desplegar nuestro servlet de prueba

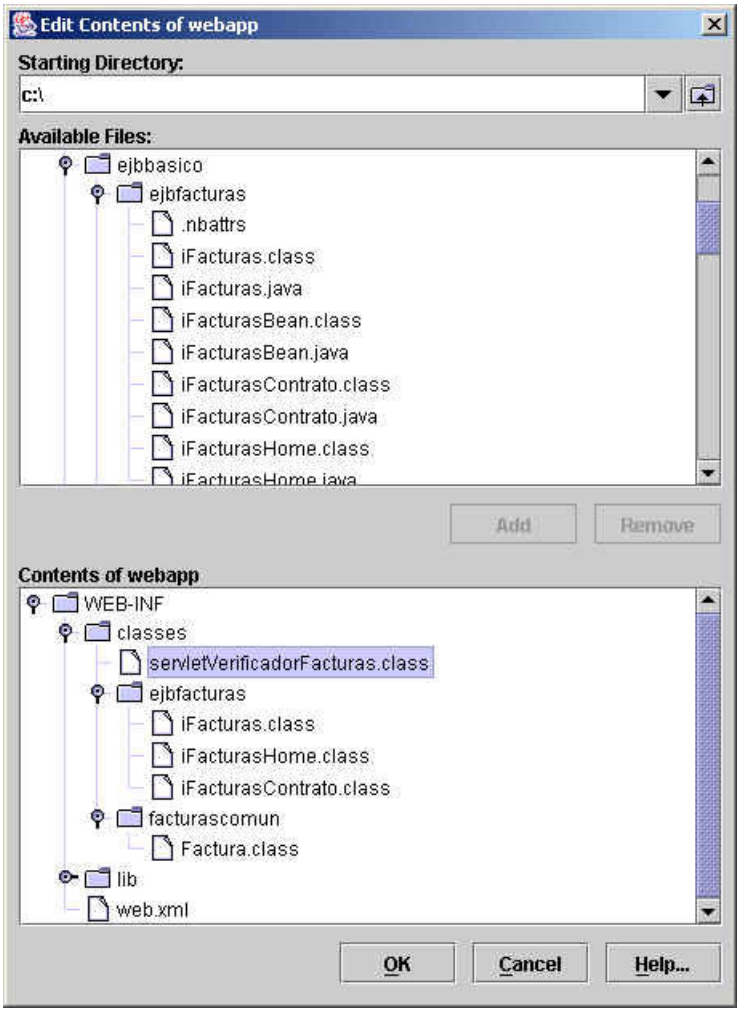


Vemos la descripción.

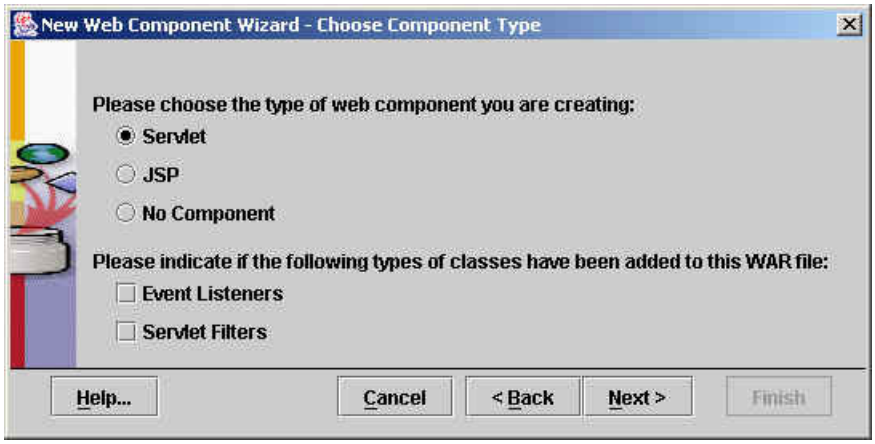
Asignamos el nombre y pinchamos editar



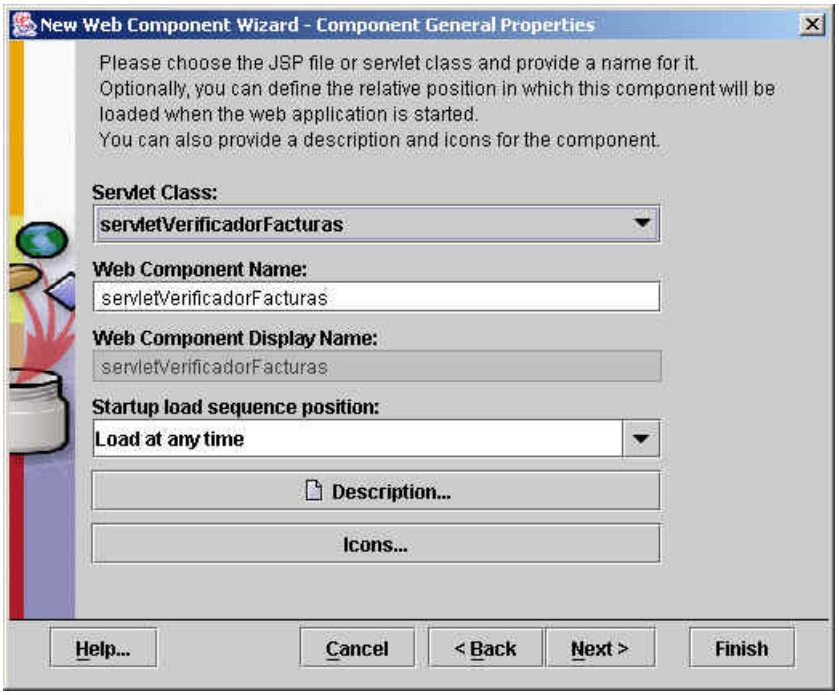
Seleccionamos el Servlet y las clases necesarias del EJB.



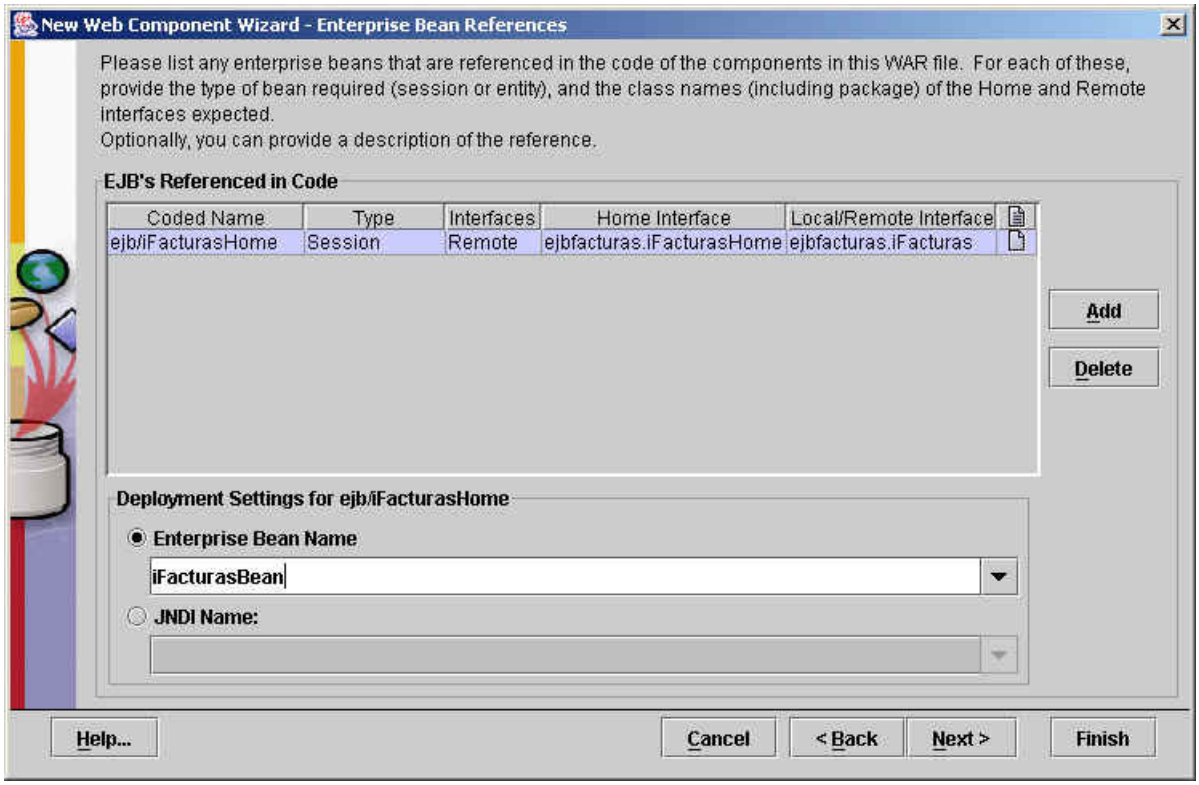
Seleccionamos un servlet



Elegimos la clase adecuada



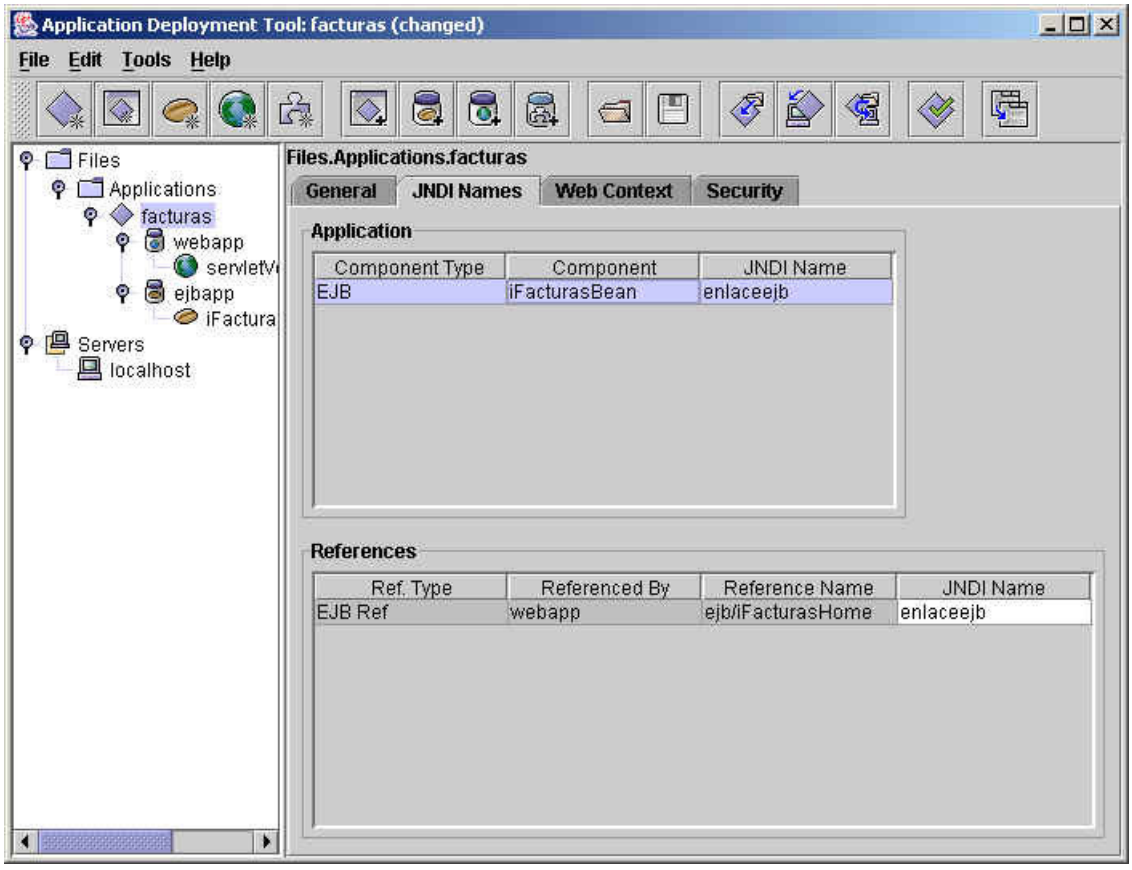
Y como nuestro servlet utiliza un EJB, añadimos una referencia. Desde nuestro servlet lo invocamos como ejb/iFacturasHome



En la Web-app (y en nuestro servlet) buscaremos un EJB llamado ejb/iFacturasHome. Podría darse la circunstancia de que nosotros no hubieramos desarrollado el EJB y ya estuviera desplegado en el servidor con un nombre determinado. Para evitar problemas y tener que tocar el código (adaptando los nombres) se pueden crear alias.... de este modo... si alguien cambia algun nombre ... solo hay que cambiar las referencia.

Esto es introducir un nivel más de indirección.

Creamos un nombre JNDI .... en este caso **enlaceejb.** Cualquier aplicación podrá localizar el EJB por este alias y como vemos en la siguiente pantalla, asociamos ejb/FacturasHome con el alias creado.



Descriptores

Vamos a ver cada uno de los descriptores..... No lo comentamos porque para eso esta la especificación EJB que podéis encontrar en el Web de Sun

application.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE application PUBLIC '-//Sun Microsystems, Inc.//DTD J2EE Application 1.3//EN'
'http://java.sun.com/dtd/application_1_3.dtd'>

<application>
<display-name>facturas</display-name>
<description>Application description</description>
<module>
<ejb>ejb-jar-ic.jar</ejb>
</module>
<module>
<web>
<web-uri>war-ic.war</web-uri>
<context-root>adictos</context-root>
</web>
</module>
</application>
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app PUBLIC '-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN'
'http://java.sun.com/dtd/web-app_2_3.dtd'>

<web-app>
<display-name>webapp</display-name>
<servlet>
<servlet-name>servletVerificadorFacturas</servlet-name>
<display-name>servletVerificadorFacturas</display-name>
<servlet-class>servletVerificadorFacturas</servlet-class>
</servlet>
<session-config>
<session-timeout>30</session-timeout>
</session-config>
<ejb-ref>
<ejb-ref-name>ejb/iFacturasHome</ejb-ref-name>
<ejb-ref-type>Session</ejb-ref-type>
<home>ejbfacturas.iFacturasHome</home>
<remote>ejbfacturas.iFacturas</remote>
<ejb-link>iFacturasBean</ejb-link>
</ejb-ref>
</web-app>
```

ejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN'
'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>

<ejb-jar>
<display-name>ejbapp</display-name>
<enterprise-beans>
<session>
<display-name>iFacturasBean</display-name>
<ejb-name>iFacturasBean</ejb-name>
<home>ejbfacturas.iFacturasHome</home>
<remote>ejbfacturas.iFacturas</remote>
<ejb-class>ejbfacturas.iFacturasBean</ejb-class>
<session-type>Stateless</session-type>
<transaction-type>Bean</transaction-type>
<security-identity>
<description></description>
<use-caller-identity></use-caller-identity>
</security-identity>
</session>
</enterprise-beans>
<assembly-descriptor>
<method-permission>
<unchecked />
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Remote</method-intf>
<method-name>recuperaIdFacturas</method-name>
<method-params />
</method>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Remote</method-intf>
<method-name>getHandle</method-name>
<method-params />
</method>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Home</method-intf>
<method-name>getEJBMetaData</method-name>
```

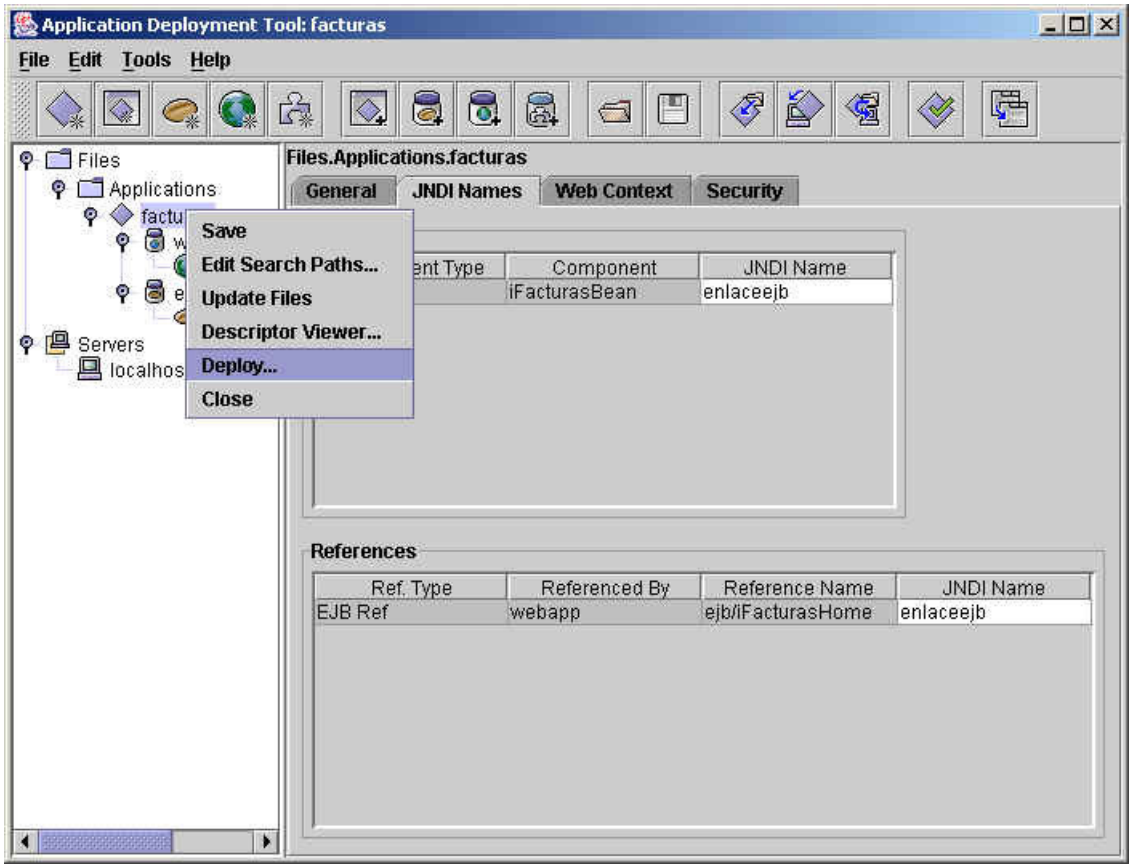
```
<method-params />
</method>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Remote</method-intf>
<method-name>getPrimaryKey</method-name>
<method-params />
</method>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Remote</method-intf>
<method-name>remove</method-name>
<method-params />
</method>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Remote</method-intf>
<method-name>recuperaNumeroFacturas</method-name>
<method-params />
</method>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Remote</method-intf>
<method-name>recuperaFacturaPorId</method-name>
<method-params>
<method-param>int</method-param>
</method-params>
</method>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Home</method-intf>
<method-name>remove</method-name>
<method-params>
<method-param>javax.ejb.Handle</method-param>
</method-params>
</method>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Home</method-intf>
<method-name>getHomeHandle</method-name>
<method-params />
</method>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Remote</method-intf>
<method-name>isIdentical</method-name>
<method-params>
<method-param>javax.ejb.EJBObject</method-param>
</method-params>
</method>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Home</method-intf>
<method-name>remove</method-name>
<method-params>
<method-param>java.lang.Object</method-param>
</method-params>
</method>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Home</method-intf>
<method-name>create</method-name>
<method-params />
</method>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Remote</method-intf>
<method-name>getEJBHome</method-name>
<method-params />
</method>
</method-permission>
<container-transaction>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Remote</method-intf>
<method-name>recuperaIdFacturas</method-name>
<method-params />
</method>
<trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Remote</method-intf>
<method-name>recuperaFacturaPorId</method-name>
<method-params>
<method-param>int</method-param>
</method-params>
</method>
<trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
<method>
<ejb-name>iFacturasBean</ejb-name>
<method-intf>Remote</method-intf>
<method-name>recuperaNumeroFacturas</method-name>
```

```
<method-params />
</method>
<trans-attribute>Required</trans-attribute>
</container-transaction>
</assembly-descriptor>
</ejb-jar>
```

Despliegue de la aplicación

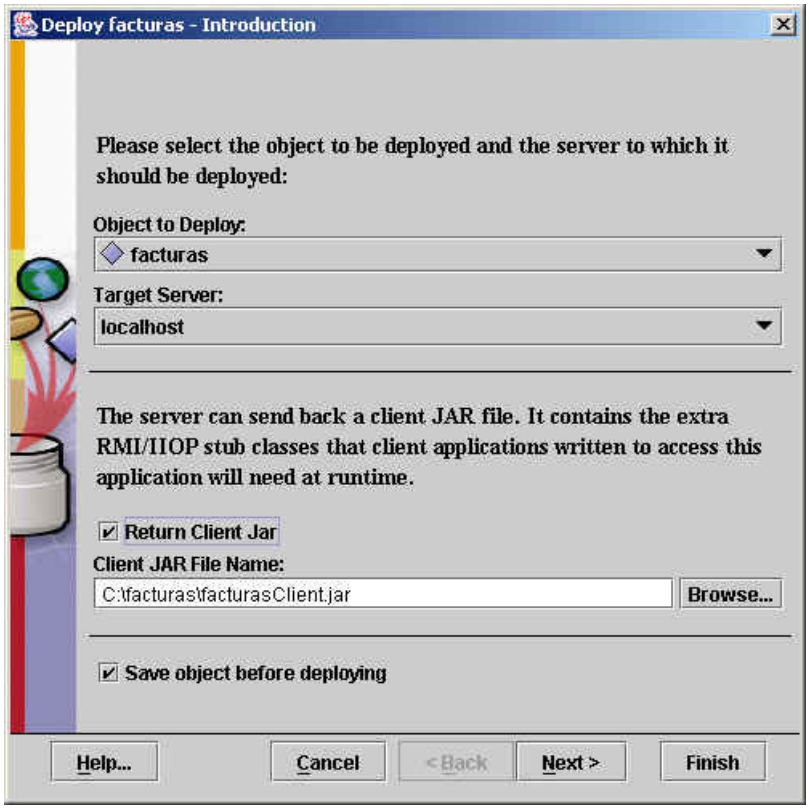
Ahora, pulsamos el botón de grabar y desplegamos la aplicación. Al grabar se recopilará toda la información y se generarán los ficheros comprimidos con toda la información de despliegue:

- Clases de la aplicación
- Ficheros descriptores
- Clases generadas para acceder de modo remoto a nuestra aplicación (proxy y stubs)

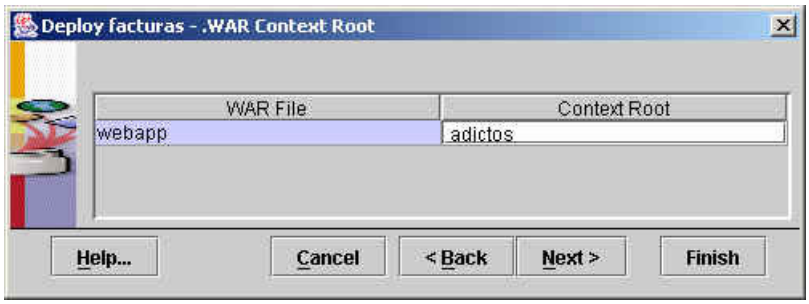


Ordenamos crear los ficheros necesarios para poder acceder a nuestro EJB desde otros servidores de aplicaciones (Return Client Jar).

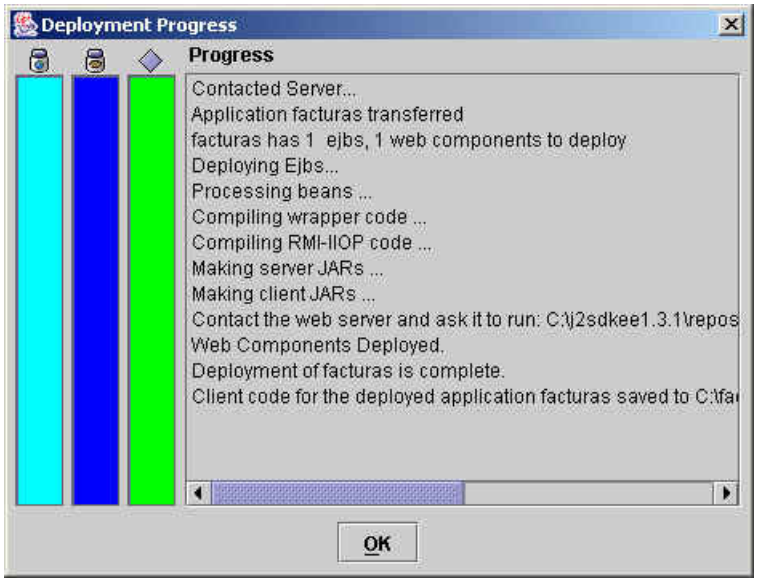
Si queremos que estos EJB puedan ser utilizados desde otras máquinas y aplicaciones, deberemos distribuir este jar al equipo que intente hacerlo.



Asignamos un contexto a nuestra Web-app, en este caso, adictos



Al pulsar el botón finalizar, vemos que se despliega bien la aplicación



Ahora invocamos nuestra web-app y vemos como funciona correctamente

Tambien vamos a ver el descriptor generado en **facturasCliente.jar**

Las lineas importantes son

<remote-home-impl>**ejbfacturas.iFacturasBean_RemoteHomeImpl**</remote-home-impl>
<remote-impl>**ejbfacturas.iFacturasBean_EJBObjectImpl**</remote-impl>

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE j2ee-ri-specific-information PUBLIC '-//Sun Microsystems Inc.//DTD J2EE Reference
Implementation 1.3//EN' 'http://localhost:8000/sun-j2ee-ri_1_3.dtd'>

<j2ee-ri-specific-information>
<rolemapping />
<web>
<module-name>war-ic.war</module-name>
<context-root>adictos</context-root>
<ejb-ref>
<ejb-ref-name>ejb/iFacturasHome</ejb-ref-name>
<jndi-name>enlaceejb</jndi-name>
</ejb-ref>
</web>
<enterprise-beans>
<module-name>ejb-jar-ic.jar</module-name>
<unique-id>-1096004843</unique-id>
<ejb>
<ejb-name>iFacturasBean</ejb-name>
<jndi-name>enlaceejb</jndi-name>
<ior-security-config>
<transport-config>
<integrity>supported</integrity>
<confidentiality>supported</confidentiality>
<establish-trust-in-target>supported</establish-trust-in-target>
<establish-trust-in-client>supported</establish-trust-in-client>
</transport-config>
<as-context>
<auth-method>username_password</auth-method>
<realm>default</realm>
<required>false</required>
</as-context>
<sas-context>
<caller-propagation>supported</caller-propagation>
</sas-context>
</ior-security-config>
<gen-classes>
<remote-home-impl>ejbfacturas.iFacturasBean_RemoteHomeImpl</remote-home-impl>
<remote-impl>ejbfacturas.iFacturasBean_EJBObjectImpl</remote-impl>
</gen-classes>
</ejb>
</enterprise-beans>
</j2ee-ri-specific-information>
```

Prueba externa del EJB

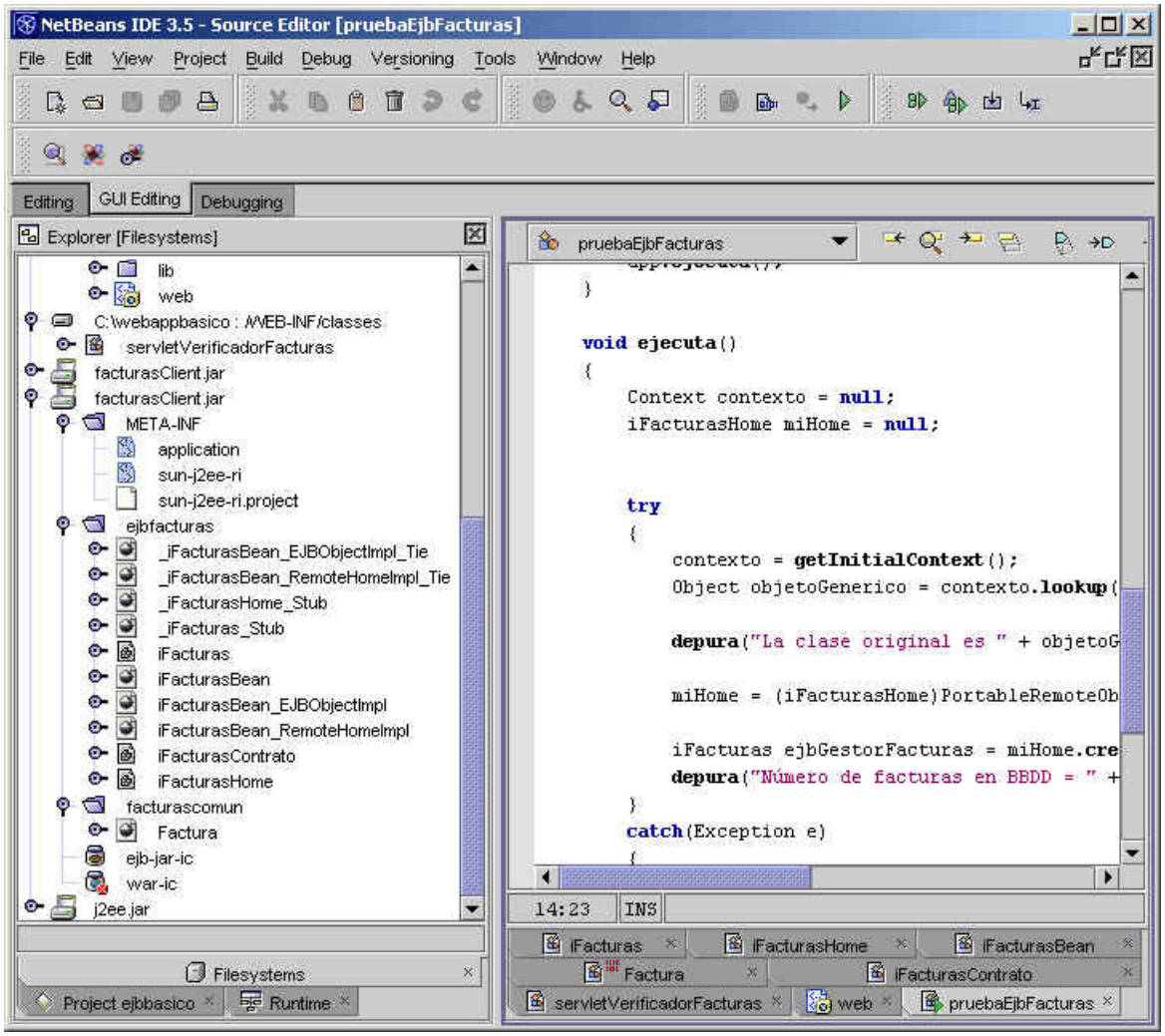
Tambien es interesante tener un sistema sencillo de probar desde una aplicación de un modo externo.

Vamos a crear una función main para llamar desde fuera del servidor de aplicaciones a nuestro EJB.

```
/*
 * pruebaEjbFacturas.java
 *
 * Created on 20 de septiembre de 2003, 17:25
 */
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;

import facturascomun.*;
import ejbfacturas.*;

import javax.naming.*;
import ejbfacturas.*;
```

```
import javax.rmi.*;

import java.util.*;

/**
 *
 * @author  Administrator
 */
public class pruebaEjbFacturas {

    /** Creates a new instance of pruebaEjbFacturas */
    public pruebaEjbFacturas() {
    }


    private static InitialContext getInitialContext() throws NamingException
    {
        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContextFactory");
        env.put("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");
        env.put(Context.PROVIDER_URL, "iiop://127.0.0.1:1050");

        return new InitialContext(env);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        pruebaEjbFacturas app = new pruebaEjbFacturas();
        app.ejecuta();
    }

    void ejecuta()
    {
        Context contexto = null;
        iFacturasHome miHome = null;


        try
        {
            contexto = getInitialContext();
            Object objetoGenerico = contexto.lookup("enlaceejb");

            depura("La clase original es " + objetoGenerico.getClass().getName());

            miHome = (iFacturasHome)PortableRemoteObject.narrow(objetoGenerico,iFacturasHome.class);

            iFacturas ejbGestorFacturas = miHome.create();
            depura("Número de facturas en BBDD = " + ejbGestorFacturas.recuperaNumeroFacturas());
        }
        catch(Exception e)
        {
            depura("Error al gestionar Facturas\n" + e.getMessage());
            e.printStackTrace();

        }
    }

    void depura(String sCadena)
    {
        System.out.println("Depuracion: " + sCadena);
    }
}
```

Para que este código funcione, es necesario incluir en el classpath los ficheros **j2ee.jar** y **facturasCliente.jar**



Un vez realizados todos estos pasos, estamos en marcha.....

A partir de aquí, empezaremos a complicar un poquito la aplicación e introducir nuevas técnicas ..... hasta pronto.

Sobre el Autor ..

Puedes opinar sobre este tutorial aquí

Recuerda

que el personal de Autentia te regala la mayoría del conocimiento aquí compartido (Ver todos los tutoriales)

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?

¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?

info@autentia.com

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos .....
Autentia = Soporte a Desarrollo & Formación

Autentia S.L. Somos expertos en:
J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ..

y muchas otras cosas

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
Message-Driven Beans al instante	Os mostramos como crear un EJB que consuma los mensajes JMS de una cola
Desarrollo de Entity Beans	Os mostramos como construir un Entity Bean básico y desplegarlo en el servidor J2EE de referencia. Lo usaremos como base de buenas prácticas J2EE
Repositorio CVS en Windows	Os mostramos como montar un servidor para el control de versiones CVS en Windows asi como acceder a él a través de WinCVS
Generador automático de Webservices	Os mostramos como crear un servicio Web a partir de una clases, gracias a generadores automáticos de código y NetBeans
Instalar JBoss	Os mostramos como instalar en servidor gratuito de aplicaciones JBOSS así como a automatizar su arranque y parada.
CMP Entity Beans y MySql	Os mostramos como crear un Entity Bean con persistencia controlada por el servidor, configurado para usar MySql
EJB´s y Orion	Recreación de la guía paso a paso de como crear una aplicación Web con EJB´s y Servlets y su despliegue con ANT sobre Orion
Introducción a ANT	En el mundo Java, la compilación, verificación e instalación de aplicaciones se ha normalizado con este potente paquete llamado ANT.

Nota: Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento.

Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores.

En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo.

Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador rcanales@adictosaltrabajo.com para su resolución.

www.AdictosAlTrabajo.com Opimizado 800X600