Exprimiendo Java Web Start

Java Web Start es la apuesta de Sun Microsystemsä para lanzar a Java de una vez por todas dentro del mundo de las apliccaiones de escritorio. La capacidad de ejecutar aplicaciones desde un navegador web y de hacer transparente al desarrollador y al cliente el control de versiones y de dependencias hacen de Java Web Start una herramienta de valor incalculable. En este artículo se muestra como sacar el máximo partido a esta tecnología en entornos empresariales donde los requerimientos de rendimiento y mantenibilidad son de gran importancia.

¿Qué se puede encontrar en este artículo?

Este artículo trata de mostrar algunas técnicas para sacarle el máximo partido a Java Web Start dentro de un entorno empresarial. En los primeros apartados se hace una pequeña introducción a la tecnología y se expone un ejemplo muy simple de su uso. Finalmente se muestra como ejecutar múltiples aplicaciones en una misma máquina virtual y como aprovechar el mecanismo de carga dinámica de aplicaciones, que nos brinda la especificación *JNLP*, dentro de nuestras aplicaciones empresariales sin la necesidad de utilizar un navegador web para ello.

¿Qué no se puede encontrar en este artículo?

Este artículo no es una descripción exhaustiva de Java Web Start y *JNLP*, de ningún modo se trata de una guía definitiva de estas tecnologías y ninguna de ellas se trata en profundidad. Aunque se realiza una pequeña introducción, para poder seguir este artículo es recomendable que el lector esté familiarizado con estas tecnologías, en especial con la estructura del descriptor *JNLP* y el funcionamiento básico de la carga de aplicaciones con Java Web Start. En [4,5,6,15,16,17,18] se puede encontrar más información sobre todos estos conceptos.

Introducción a Java Web Start

Java Web Start es la implementación de referencia de la especificación *JNLP* (JSR 56, Java Networking Launching Protocol)[1] que define como ejecutar aplicaciones Java remotamente desde un entorno de red cualquiera.

Java Web Start revoluciona el concepto tradicional que tenemos de las aplicaciones. Normalmente cuando se quiere ejecutar una aplicación que no se encuentra instalada en un equipo, se descarga del servidor, se instala en dicho equipo y por último se ejecuta. Java Web Start intenta simplificar al máximo todo este proceso de modo que el usuario lo único que tiene que hacer para lanzar una aplicación sea simplemente pinchar en un enlace de su navegador, a partir de ese momento, todo el proceso relacionado con la descarga, instalación y ejecución del programa se realiza de una manera transparente.

A pesar de su parecido, una aplicación de Java Web Start no tiene nada que ver con un Applet. Java Web Start sólo utiliza el navegador como medio para que el usuario pueda ejecutar las aplicaciones. Una vez que el usuario pincha en un enlace de una aplicación, ésta se ejecuta en la máquina virtual del cliente como cualquier otra aplicación.

Java Web Start no forma parte del navegador web, es una aplicación independiente y por lo tanto no requiere del navegador para su funcionamiento. Una vez que el usuario pincha en un enlace para ejecutar una aplicación, puede continuar navegando o cerrar el navegador sin que esto interfiera en el funcionamiento de la aplicación que ha sido lanzada. Además, Java Web Start va guardando en una caché interna las aplicaciones que va ejecutando el usuario, de modo que éste pueda lanzarlas posteriormente sin la necesidad de abrir el navegador o incluso ejecutarlas localmente sin conectarse a ninguna red.

Las aplicaciones Java Web Start siguen el modelo de seguridad de la plataforma Java 2 por lo que la integridad de los datos que obtenemos a través de la red está garantizada. Como veremos, comúnmente las aplicaciones que se ejecuten han de estar debidamente firmadas y se requiere siempre que el usuario autorice su ejecución.

Java Web Start viene incluido de serie dentro en el *JRE* a partir de su versión 1.4. La última versión es la 1.2 (beta) que viene con el *JRE* 1.4.1 también beta. Como curiosidad reseñar que el sistema operativo OS X de Macintosh ya trae preinstalado soporte para aplicaciones Java Web Start. Aunque técnicamente es necesario que se encuentre instalado al menos un *JRE* dentro de la máquina cliente para poder ejecutar aplicaciones Java Web Start, lo cierto es que éstas se pueden configurar de manera que el *JRE* utilizado se descargue automáticamente si no se encuentra disponible con lo que se consigue una transparencia absoluta para el cliente.

Java Web Start no es la única implementación de la especificación *JNLP*. Una alternativa muy popular es *OpenJNLP* [10], una implementación *Open Source* de la especificación que está desarrollada completamente en Java y que utilizaremos en el último apartado de este artículo.

Ventajas y desventajas de Java Web Start

Como ya he dicho anteriormente, Java Web Start revoluciona por completo el concepto tradicional de aplicaciones. Las ventajas que ofrece tanto a los desarrolladores de las mismas como a los usuarios son muchas y muy importantes:

- **Transparencia** : El usuario no necesita pasar por un proceso traumático de descarga e instalación de la aplicación para poder ejecutarla. Únicamente tiene que pinchar un enlace en su navegador y la aplicación se descarga, se instala y se ejecuta de manera automática. Además, Java Web Start se encarga de crear los accesos directos correspondientes en el escritorio y menú de inicio del usuario.
- **Mantenibilidad** : Para los desarrolladores y administradores de sistema, Java Web Start es una bendición. Ahora ya no es necesario copiar la misma aplicación a todos los usuarios de una red cada vez que se realiza una pequeña modificación en la misma, sino que con actualizarla en el servidor web es suficiente para que los usuarios puedan utilizar la última versión de la misma.
- **Control de versiones** : Java Web Start se encarga automáticamente de realizar el control de versiones de las aplicaciones. Antes de ejecutar una aplicación, Java Web Start comprueba en el servidor web que no exista una versión más avanzada de la misma, en cuyo caso actualizará la vieja versión por la nueva automáticamente. Esto beneficia tanto a los usuarios que siempre ejecutan la última versión de su software, como a los desarrolladores que no tienen necesidad de distribuir las nuevas versiones a los usuarios o crear algún sistema interno de control de versiones.
- **Independencia del servidor web y del navegador** : Java Web Start puede funcionar en cualquier servidor web tan sólo añadiendo el tipo MIME correspondiente a los ficheros con extensión *.jnlp*, por otra parte, también funcionará en cualquier navegador aunque en algunos habrá que configurar el programa asociado a los ficheros con dicha extensión.
- **Independencia del sistema operativo** : Aunque Java Web Start no está disponible para todos los sistemas operativos para los que la plataforma Java se encuentra disponible, *OpenJNLP* que como dijimos está escrito en Java y que es una inciativa *Open Source* si que es totalmente independiente del sistema operativo.
- **Automatiza la gestión de *JRE*s** : Cada aplicación puede decidir que *JRE* quiere utilizar para ejecutarse, es más, si ese *JRE* no existiese en el equipo del cliente, Java Web Start se encarga automáticamente de su descarga e instalación en el sistema.
- **Transparencia al desarrollador** : No es necesario modificar las aplicaciones existentes para que aprovechen esta tecnología. Para hacer una vieja aplicación compatible con Java Web Start, tan sólo hay que crear un pequeño descriptor XML con las características de la aplicación y colocarla en un servidor web. Las aplicaciones pueden seguir ejecutándose del modo tradicional sin ningún problema.
- **Ejecución local de las aplicaciones** : Java Web Start a diferencia de tecnologías como JSP/Servlets no necesita la red para ejecutar las aplicaciones. La red tan sólo es un medio para obtener dichas aplicaciones y sus actualizaciones. Una vez descargada una aplicación, ésta se ejecuta de manera local y tan sólo accede a la red si lo necesita para su funcionamiento.

Como toda tecnología, Java Web Start no está exenta de problemas:

- **Una máquina virtual por aplicación** : Este es quizás el problema más importante, aunque posteriormente veremos una posible solución. La especificación *JNLP* establece que cada aplicación se ha de ejecutar en una máquina virtual diferente. Obviamente esto es un gran obstáculo para entornos con recursos limitados y donde sea necesario ejecutar múltiples aplicaciones diferentes simultáneamente obligando a un consumo de recursos y de memoria innecesario.
- **Problemas de flexibilidad** : Java Web Start tiene varias limitaciones de flexibilidad : no se pueden pasar algunos parámetros a la máquina virtual ya que se comprometería la seguridad y la portabilidad (ejemplo: los parámetros que comienzan con -X), algunas opciones sólo se pueden configurar desde el ordenador del usuario (como el tipo de máquina virtual a utilizar, registrar la salida, etc.)
- **No soporta los *JRE* 1.1 e inferiores**: Java Web Start basa su funcionamiento en el modelo de seguridad de la plataforma Java 2 por lo que no existe soporte para versiones anteriores.

Utilizando Java Web Start

En este apartado vamos a ver con una aplicación sencilla el uso de Java Web Start. El código fuente de la aplicación es el siguiente y todos los ficheros necesarios para ejecutar este ejemplo se encuentran en [26].

```
import javax.swing.*;
import java.awt.event.*;
import java.util.*;

public class Main {

    private static int count;
    private static List buttons = new ArrayList();
    private JButton button = new JButton();

    public Main() {
        JFrame frame = new JFrame();
        JButton button = new JButton();
        buttons.add(button);
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                count++;
                Iterator it = buttons.iterator();
                while (it.hasNext()) {
                    ((JButton)it.next()).setText("clicks = " + count);
                }
            }
        });
        button.setText("clicks = " + count);
        frame.getContentPane().add(button);
        frame.setSize(300,300);
        frame.setLocation(400,300);
        frame.setVisible(true);

        // Importante. Si se hace un exit se cerrar? el loader
       frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }

    public static void main(String[] args) {
        System.out.println("[Main] main class executed");
        new Main();
    }
}
```

Como se puede apreciar, se trata de un ejemplo muy sencillo que muestra una ventana con un botón que al pulsarlo incrementa un contador. El contador es estático para poder comprobar fácilmente si nuestro programa se ejecuta en máquinas virtuales diferentes cuando lo lanzamos varias veces y que en el siguiente apartado utilizaremos para ver como las aplicaciones se ejecutan en la misma máquina virtual.

Una vez que hayamos creado nuestra aplicación y comprobado que funciona correctamente en modo local crearemos el fichero jar que contendrá la aplicación. Para ello simplemente ejecutamos la siguiente línea:

```
  jar -cvf main.jar *.class
```

El siguiente paso es la creación del descriptor *JNLP*. Este descriptor es un fichero XML que contiene información sobre nuestra aplicación y sobre como ha de lanzarla Java Web Start. El descriptor de nuestra aplicación es muy sencillo y no presenta ningún problema incluso a los lectores no familiarizados con esta tecnología, sin embargo el número de parámetros y opciones que soporta dicho descriptor es bastante grande, por lo que no dude en consultar las referencias al final de este artículo para obtener una información más detallada sobre el mismo.

Lo vemos a continuación :

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

<jnlp spec="1.0+" codebase="http://localhost:8080/jnlp/" href="jnlp.jnlp">

  <information>
```

```
    <title>Ejemplo de JNLP</title>
    <vendor>JavaHispano</vendor>
    <homepage href="http://www.javahispano.com"/>
    <description>Ejemplo de JNLP</description>
    <description kind="short">
      Esta aplicación es un pequeño ejemplo de la tecnología JNLP
    </description>
    <icon href="images/javahispano.jpg"/>
    <offline-allowed/>
  </information>

  <security>
    <all-permissions/>
  </security>

  <resources>
    <j2se version="1.4+"/>
    <jar href="lib/main.jar"/>
  </resources>

  <application-desc main-class="Main">
  </application-desc>

</jnlp>
```

Como se puede observar la mayoría de los campos son autodescriptivos. Quizás los más interesantes sean la etiqueta <jnlp> cuyos atributos especifican donde se encuentra el fichero *JNLP*, la etiqueta <jar> que permite especificar los diferentes archivos que componen nuestra aplicación, la etiqueta <j2se> cuyo atributo version especifica cuál es la máquina virtual que se usará para ejecutar la aplicación y la etiqueta <application-desc> cuyo atributo main-class especifica cual es la clase principal de la aplicación.

Java Web Start puede ejecutar las aplicaciones en dos modos diferentes. El primero, es el modo restringido, en el que las aplicaciones se ejecutan en un sandbox, modo en el que sólo pueden hacer uso de determinados recursos del sistema. El API de Java Web Start permite la utilización de diversos servicios programáticamente para poder saltarse algunas limitaciones de este modelo y permitir de este modo el acceso a ficheros, al portapapeles, a la descarga de archivos, etc.

El segundo modo, es el modo de confianza y es el que utilizaremos en este y el resto de ejemplos. En este modo las aplicaciones consiguen el acceso a todos los recursos del sistema. Para que una aplicación pueda conseguir dicho acceso, previamente ha de tener todos los ficheros jar de los que conste firmados digitalmente, de este modo, cuando el usuario quiera ejecutar la aplicación le aparecerá un certificado donde se solicita acceso no restringido al sistema, el usuario es el que decide si debe confiar o no en la fuente que emite el certificado.

El proceso de firma de los ficheros jar es muy sencillo. Lo primero que hay que hacer es crear una clave de autenticación con la herramienta keytool, por ejemplo:

```
  keytool -genkey -keystore myKeyStore -alias myself
```

Esta herramienta nos pedirá información acerca de la clave y del emisor del certificado que le aparecerá al usuario al ejecutar la aplicación. La clave se almacenará en el almacén de claves que especifiquemos. Una vez creada la clave, tan sólo nos queda firmar todos nuestros ficheros jar, en este caso:

```
  jarsigner -keystore myKeyStore main.jar myself
```

Por último, para pedir acceso no restringido al sistema habría que añadir las siguientes líneas al descriptor jnlp:

```
<security>
  <all-permissions/>
</security>
```

Bien, la aplicación ya está preparada por completo, tan sólo falta configurar el servidor web. En este ejemplo y en el resto, utilizaré Apache Tomcat [19], aunque también se puede utilizar cualquier otro servidor. En el servidor que se utilice habrá que añadir soporte para el tipo MIME para los ficheros *JNLP*, en este caso, Tomcat ya lo trae incluido por lo que no hay que configurar absolutamente nada.

Para este y el resto de ejemplos crearé la siguiente estructura dentro del servidor web:

```
raiz del servidor
|____ jnlp
       |____ jnlp.jnlp
       |____ lib
       |____ main.jar
```

Esta estructura se puede crear de muchas formas, ya sea creando un contexto partícular, un simple directorio a partir del raiz, etc.

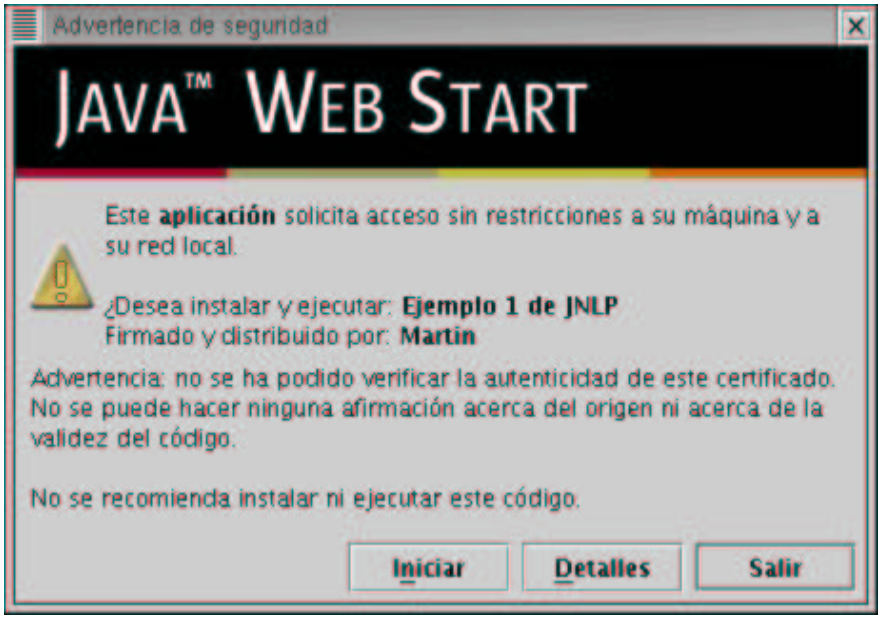
Una vez arrancado el servidor web, se debería acceder a la aplicación escribiendo el enlace (suponiendo que usamos Apache Tomcat) :

```
htp://localhost:8080/jnlp/jnlp.jnlp
```

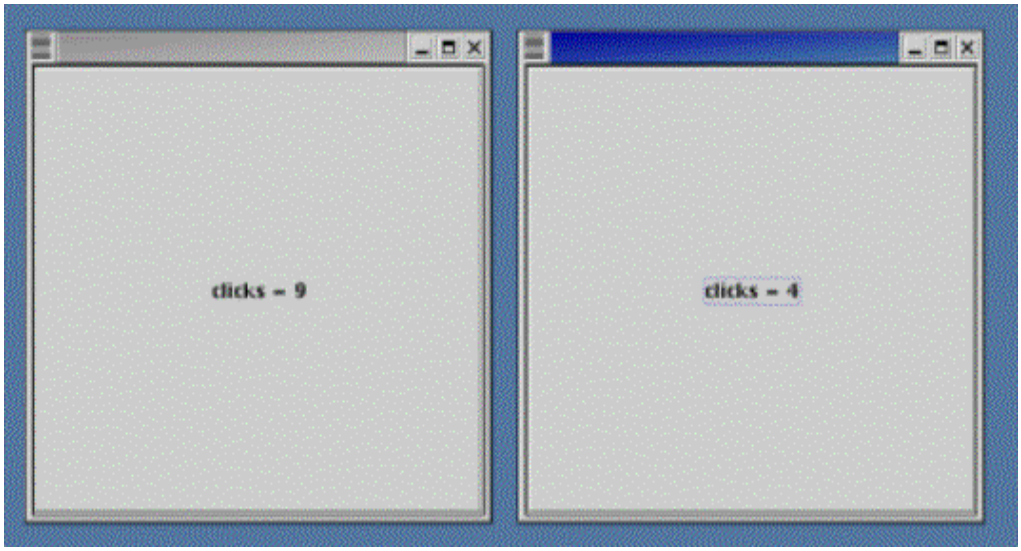
Según el navegador que se utilice para ejecutar el ejemplo quizás sea necesario configurarlo para que asocie el tipo MIME *JNLP* con la aplicación Java Web Start u *OpenJNLP* en caso de utilizar este último. Si por cualquier razón no se es capaz de configurar el navegador para ejecutar aplicaciones *JNLP*, siempre se pueden lanzar desde la línea de comandos, por ejemplo con Java Web Start tendríamos que escribir los siguiente:

```
directorio_de_Java_Web_Start/javaws http://localhost:8080/jnlp/jnlp.jnlp
```

Una vez hecho esto, si es la primera vez que se ejecuta la aplicación, aparecerá una alerta de seguridad en la que se le pregunta al usuario si quiere confiar en dicha aplicación y en la fuente que emite el certificado. En el mensaje hay una alerta de que no se puede verificar la autenticidad del certificado, esto es totalmente normal ya que no se ha comprado dicho certificado a ninguna autoridad de certificación. Para lanzar la aplicación definitivamente hay que pulsar el botón Iniciar.



Uno de los problemas de Java Web Start que ya mencioné anteriormente es que cada aplicación se ejecuta en una máquina virtual diferente. Una manera de comprobarlo es ejecutar dos veces la aplicación de ejemplo de este apartado y ver como al pulsar el botón de una de ellas el otro botón no se ve modificado, esto se debe a que las aplicaciones se están ejecutando cada una en su máquina virtual. En el siguiente apartado se verá una manera simple de sobrepasar este inconveniente y poder de este modo ejecutar gran cantidad de aplicaciones en la misma máquina virtual.



Ejecutando múltiples aplicaciones en la misma máquina virtual

Como se ha visto, una de las ventajas de Java Web Start es que permite lanzar aplicaciones desde la web de una manera transparente. A poco que pensemos, una de las consecuencias de esto es la posibilidad de crear portales empresariales que engloben aplicaciones de muy diferentes tipos.

A menudo, en las empresas encontramos aplicaciones de muy diversa índole. Es muy sencillo que en una misma empresa existan aplicaciones nativas (ya sean del sistema operativo o viejas aplicaciones creadas por la empresa), aplicaciones Java y aplicaciones basadas en tecnología web (JSP, Servlets, ASP, PHP, etc.). La diversidad de todas estas tecnologías hace que sea muy díficil la creación de un portal personalizado donde cada usuario pueda ejecutar estas aplicaciones y conseguir una alta mantenibilidad del sistema.

Java Web Start nos ofrece una buena posibilidad para realizar un portal de este estilo. Las aplicaciones web no plantean ningún problema, las aplicaciones Java tampoco son un problema ya que esta tecnología nos permite lanzarlas directamente desde el navegador mientras que las aplicaciones nativas pueden ejecutarse utilizando un pequeño lanzador de aplicaciones realizado en Java y que también podrá ejecutarse sin problemas desde el navegador. Las ventajas de un portal de este estilo son inmensas: mantenibilidad, centralización de la información, control sencillo de los permisos de acceso, personalización del contenido, etc.

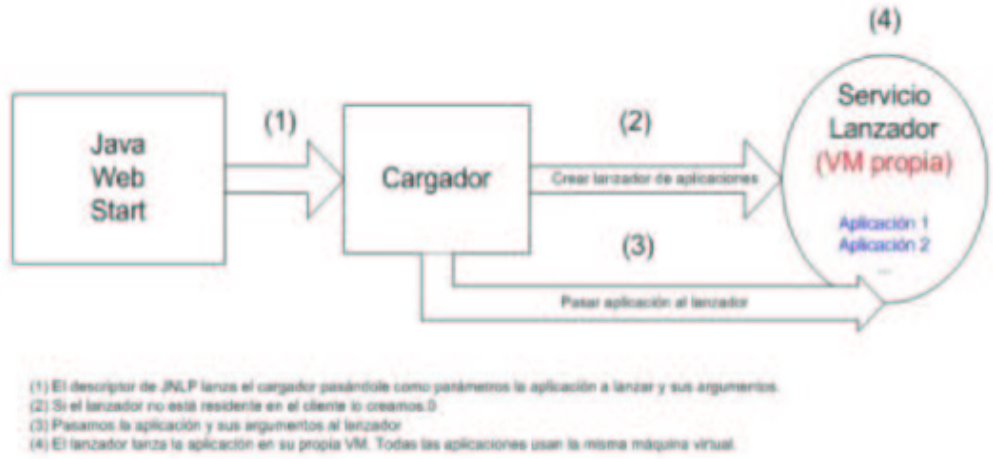
Sin embargo todavía nos queda un problema: cada aplicación se ejecutará en una máquina virtual diferente, algo que es inadmisible cuando el número de aplicaciones a ejecutar es relativamente grande y el número de recursos es limitado.

En este apartado se muestra una posible solución a este problema basada en el uso de un lanzador de aplicaciones. Este lanzador actuará como un demonio de sistema que se quedará a la espera de aplicaciones Java para ejecutar. Cuando una aplicación Java quiera ejecutarse, el cargador iniciará un nuevo hilo en su máquina virtual y la lanzará.

Para conseguir hacer esto es necesario centralizar el acceso a las aplicaciones, es decir, antes, teníamos que cada descriptor *JNLP* se utilizaba para ejecutar una aplicación diferente, ahora cada descriptor *JNLP* se utilizará para ejecutar siempre nuestro lanzador de aplicaciones y a éste se le pasará como parámetro la aplicación que se quiera lanzar. Siguiendo con el ejemplo del apartado anterior el descriptor *JNLP* quedaría del siguiente modo:

```
<application-desc main-class="Loader">
  <argument>Main</argument>
</application-desc>
```

En este caso el lanzador se corresponde con la clase Loader. El primer argumento que recibe dicho lanzador es la clase principal de la aplicación que se quiere ejecutar, en este caso Main. El resto de parámetros de la aplicación se pasarían también como argumentos, eso sí, el primero siempre ha de ser la clase principal. El siguiente esquema muestra gráficamente el funcionamiento de este lanzador de aplicaciones:



Básicamente:

- Si es la primera vez que se ejecuta nuestro cargador de aplicaciones, éste se queda residente en equipo del usuario esperando por aplicaciones para ser lanzadas. En este caso depués de registrar el cargador se lanzaría la aplicación que se iba a ejecutar.
- Si el lanzador de aplicaciones ya se encuentra residente, entonces se le avisa de que se quiere ejecutar una nueva aplicación, posteriormente el lanzador ejecutará dicha aplicación en un nuevo hilo de su máquina virtual.

Para implementar este lanzador de aplicaciones residente existen muchas alternativas. En este caso se ha utilizado RMI[20,21,22,23,24,25] principalmente por su sencillez; otras alternativas podrían haber sido utilizar sockets o utilizar directorios compartidos.

De aquí en adelante se mostrará el código fuente del cargador de aplicaciones que se puede encontrar en [26]. La explicación se va realizando por partes para que sea más sencilla su comprensión.

Como objeto remoto que es el lanzador de aplicaciones ha de implementar una interfaz remota:

```
import java.rmi.*;

public interface Loader extends Remote {

  public void launchApplication(String[] args) throws Exception;
    public void shutdown() throws RemoteException;
  }
```

Como se puede ver, el cargador es muy simple, tiene métodos para lanzar aplicaciones y para retirarse del sistema. Ahora voy a describir más a fondo la implementación del cargador. Primero empezaré con el método main:

```
  public static void main(String[] args) {
    if (System.getProperty("shutdown-registry") != null) { _____________________ *1
      shutdownRegistry();
      System.exit(0);
    }
    else if (System.getProperty("own-vm") != null) {       _____________________ *2
      executeAppInOwnVM(args);
    }
    else {
      try {
        createRegistry();                                  _____________________ *3
        try {
          executeApp(args);
        }
```

```
      catch (Exception e) {
        e.printStackTrace();
      }
    }
    catch (ExportException ee) {                                    *4
      System.out.println("[Loader] registry already created");
      executeApp(args);
      System.exit(0);
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

- · Lo primero que se hace en *1 es comprobar si lo que se quiere es cerrar el cargador, esto podría corresponderse con la típica opción de salir del sistema en un portal empresarial.
- · En *2 se comprueba si la aplicación ha de ejecutarse en su propia máquina virtual ya que puede que no queramos que alguna aplicación en concreto comparta la máquina virtual en la que se ejecutará con el resto de aplicaciones.
- · Si no se cumple ninguna de las dos condiciones anteriores, en *3 el cargador intenta hacerse residente en el sistema y una vez lo haya conseguido ejecuta la aplicación.
- · En caso de que ya se encuentre residente (*4) se ejecuta la aplicación.

En los siguientes puntos se muestra el código de los métodos más importantes:

```
public static void createRegistry()
  throws RemoteException, ExportException,InterruptedException {

  System.out.println("[Loader] craeting registry");
  registry = LocateRegistry.createRegistry(PORT);
  System.out.println("[Loader] registry created");
}
```

El método createRegistry() que se ve arriba simplemente intenta crear un registro RMI en el puerto especificado del equipo del cliente. En ese registro es donde guardaremos el lanzador de aplicaciones.

El método executeApp() que aparece por debajo de estas líneas es el que se encarga de ejecutar la aplicación y añadir el cargador al registro RMI si es necesario:

```
private static void executeApp(String[] args) {
  try {
    registry = lookupRegistry();                                    *1
  }
  catch (Exception e) {
    e.printStackTrace();
    return;
  }

  Loader loader = null;

  try {
    loader = lookupLoader();                                        *2
    loader.launchApplication(args);
  }
  catch (NotBoundException nbe) {
    System.out.println("[Loader] loader not bound");
    try {
      bindLoader();                                                 *3
      loader = lookupLoader();
      loader.launchApplication(args);
    }
    catch (Exception e) {
      e.printStackTrace();
      return;
    }
  }
  catch (Exception e) {
    e.printStackTrace();
```

```
      return;
    }
  }
```

- · Lo primero que se hace es intentar localizar el registro (*1) donde se debería encontrar el lanzador de aplicaciones, si el registro no se encuentra se finaliza la ejecución del programa.
- · El siguiente paso es buscar el lanzador de aplicaciones dentro del registro (*2), si lo encontramos se intentará lanzar la aplicación.
- · En caso de que no se cumpla la condición del punto anterior, se añade el lanzador de aplicaciones al registro (*3), se busca para asegurarse de que se ha cargado correctamente y finalmente se intenta lanzar la aplicación. En caso de que en alguno de estos dos últimos puntos produzca una excepción el programa finalizará.

A continuación se pueden ver estos métodos más en detalle:

```
  public static Registry lookupRegistry()
    throws RemoteException {

    System.out.println("[Loader] looking for registry");
    Registry registry = LocateRegistry.getRegistry(PORT);
    System.out.println("[Loader] registry found successfully");
    return registry;
  }

  public static Loader lookupLoader()
    throws RemoteException, NotBoundException, MalformedURLException {

    System.out.println("[Loader] looking for loader");
    Loader loader = (Loader)Naming.lookup("//localhost:"+PORT+"/loader");
    System.out.println("[Loader] loader found successfully");
    return loader;
  }

  public static void bindLoader()
    throws RemoteException, AlreadyBoundException, MalformedURLException {

    System.out.println("[Loader] binding loader");
    Naming.rebind("//localhost:"+PORT+"/loader",new LoaderImpl());
    System.out.println("[Loader] loader bound on registry");
  }
```

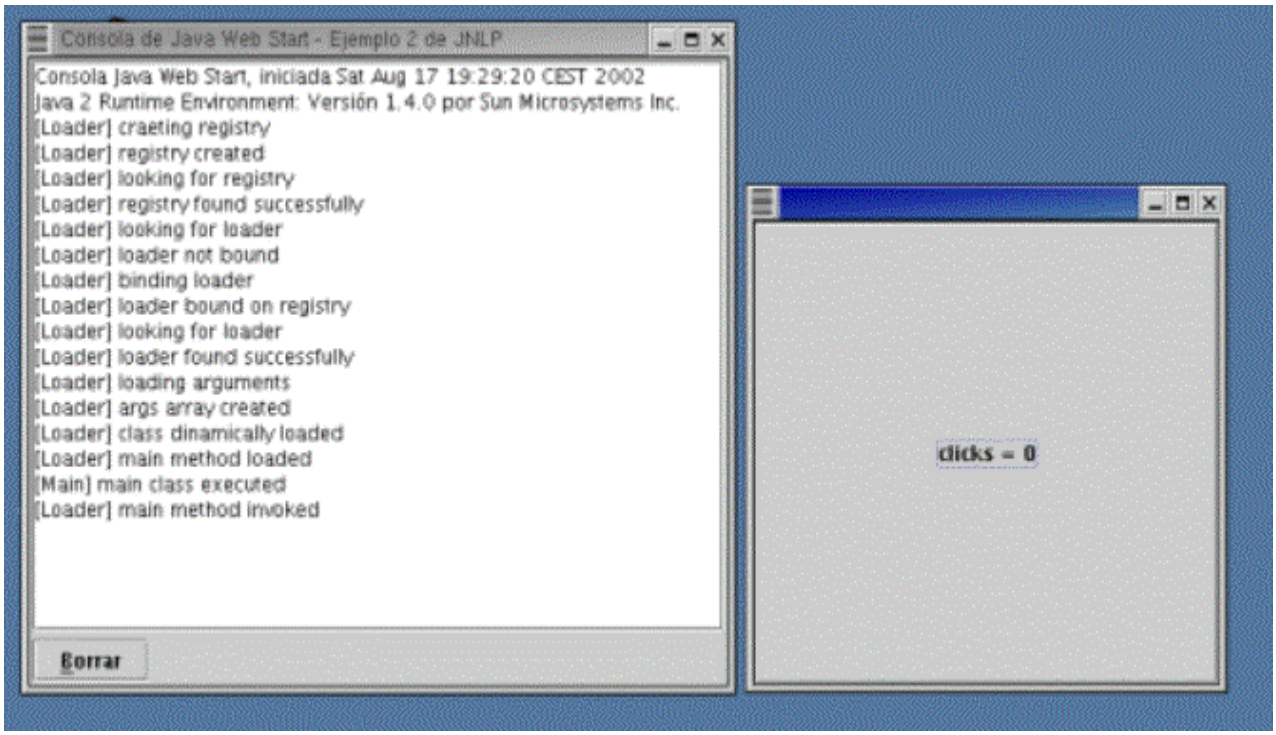
Como se puede apreciar, el código es muy simple y hacen uso de los mecanismos básicos de RMI[] para registrar y buscar los diferentes objetos.

El método launch(String[] args) es el verdadero encargado de lanzar la aplicación que se quiere ejecutar. Este método lo llama el cargador desde su máquina virtual, la misma que ejecuta todas las aplicaciones. Para ejecutar la aplicación, simplemente se obtiene el método main de la clase que queremos ejecutar utilizando el API Reflection y se llama a dicho método pasándole los argumentos necesarios; cualquier otra alternativa (llamada a un método concreto, a un constructor, a un inicializador estático, etc.), también habría sido posible y se realizaría de modo muy similar.

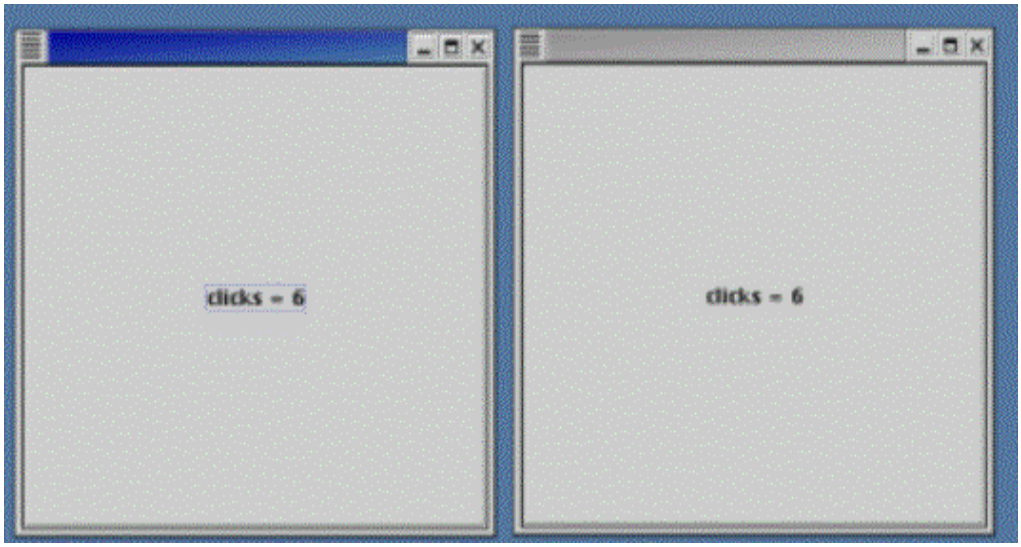
```
  public static void launch(String[] args) throws Exception {

    if (args[0].length == 0) {
      throw new LoaderException("Not enough arguments");
    }
    System.out.println("[Loader] loading arguments");
    String classname = args[0];

    String[] newArgs = new String[args.length-1];
    if (newArgs.length != 0) {
      System.arraycopy(args,1,newArgs,0,newArgs.length);
    }

    System.out.println("[Loader] args array created");
    Class appClass = Class.forName(classname);
    System.out.println("[Loader] class dinamically loaded");

    Method mainMethod = appClass.getMethod("main", new Class[]{String[].class});
    System.out.println("[Loader] main method loaded");
```

```
    mainMethod.invoke(null, new Object[]{newArgs});
    System.out.println("[Loader] main method invoked");
  }
```

El conjunto de archivos que componen el ejemplo de este apartado se encuentra en [26]. Lo primero que hay que hacer es configurar el servidor web como se vio en el apartado anterior para que el fichero *ejemplo2.jnlp* y las librerías que contienen el lanzador de aplicaciones (*loader.jar*) y la aplicación que ejecutaremos (*main.jar*) sean accesibles. Por seguir con la estructura que se vio en el apartado anterior, el fichero *ejemplo2.jnlp* irá bajo el directorio jnlp mientras que los ficheros *main.jar* y *loader.jar* se colocarán bajo el directorio jnlp/lib/. Una vez configurado todo correctamente lo único que hay que hacer es arrancar nuestro navegador web y acceder al fichero *ejemplo2.jnlp*, momento en el que el lanzador de aplicaciones se hará residente para posteriormente lanzar la aplicación.



En la consola de Java Web Start se puede ver como se va realizando todo el proceso de creación y configuración del registro para acabar lanzando la aplicación. La aplicación lanzada es la misma que vimos en el apartado anterior, es decir, una ventana con un botón que muestra el valor de una variable estática que actúa como contador. Si se vuelve a lanzar el fichero *ejemplo2.jnlp* desde el navegador se verá como ahora ya no se crea el registro y se lanza directamente la aplicación.



Falta reseñar algunos puntos:

- El tiempo de carga de la segunda aplicación y posteriores es mucho menor que el de la primera ya que no es necesario crear una nueva máquina virtual para ejecutarla, por lo tanto tenemos una ganancia importante en tiempo de lanzamiento.
- Al ejecutarse todas las aplicaciones en una misma máquina virtual hay que tener especial cuidado con las variables estáticas. En el ejemplo se ve claramente este efecto ya que al abrir varias aplicaciones se observa como al pulsar en uno de los botones el resto de contadores del resto de aplicaciones también se actualizan.

Aprovechando el mecanismo de carga dinámica de aplicaciones que ofrece *JNLP*

Hace tiempo, me toco trabajar en un proyecto interesante, se trataba la creación de múltiples aplicaciones para gestionar una empresa y una de ellas era un escritorio desde el que se pudiesen lanzar todas estas aplicaciones. El principal problema al que nos enfrentábamos era el intentar que los usuarios pudiesen acceder a todas las aplicaciones y que éstas se ejecutasen en la misma máquina virtual para de este modo aprovechar más los recursos disponibles en los clientes y e incluso tener la posibilidad de compartir estructuras de datos entre las aplicaciones.

Hace unos días, uno de mis amigos de esa empresa en la que estuve me comentó que ahora se encontraban con un pequeño problema. Con el paso del tiempo, la cantidad de programas que se han ido añadiendo a ese escritorio ha sido muy grande, en el que el tamaño de la totalidad de aplicaciones hace complicada su actualización y mantenimiento. Habían pensado en Java Web Start por la flexibilidad que ofrece pero no veían la forma de utilizarlo ya que su escritorio es una aplicación Swing y no está pensado para ser ejecutado desde un navegador web, además no quieren tirar por la borda todo el trabajo que hicimos y quieren mantener el mismo lanzador de aplicaciones por lo que hacer un equivalente en navegador web no es una opción viable.

El objetivo que se persigue es que las aplicaciones se actualicen ellas mismas cada vez que sean invocadas, consiguiendo de este modo una transparencia absoluta al usuario y un ahorro considerable de mantenimiento para el equipo de desarrollo, y todo esto manteniendo el lanzador que existía previamente.

Por suerte, existe una solución muy sencilla que es aprovechar todos estos mecanismos de carga dinámica de aplicaciones que ofrece la especificación *JNLP*, esta solución pasa por utilizar *OpenJNLP*. *OpenJNLP* es un desarrollo *Open Source*, que está formado por un cargador de aplicaciones que viene a ser el equivalente a Java Web Start y por un conjunto de librerías que implementan la especificación de *JNLP*.

Hasta ahora, en el escritorio desde el que se lanzan las aplicaciones cada vez que se pulsaba en el icono de una de dichas aplicaciones, ésta se lanzaba en un hilo diferente. Con el nuevo planteamiento, en lugar de ejecutar la aplicación directamente, lo que se hará será realizar una llamada a una función de una de las librerías de *OpenJNLP* que se encargará de comprobar y actualizar la aplicación con nuevas versiones en el caso de existiesen y ejecutar la aplicación.

En [10] está el enlace desde donde se puede descargar *OpenJNLP*, una vez descargado es necesario añadir al CLASSPATH las librerías *openjnlp-lib.jar* y *openjnlp-extra.jar*. Es muy importante también bajar el Java Web Start Developer?s Pack [3], que contiene la librería *jnlp.jar* que también es necesario añadir al CLASSPATH.

El ejemplo se compone de una ventana que contiene un botón, cada vez que se pulsa el botón se carga la aplicación que hemos utilizado hasta ahora en todos los ejemplos, es decir, la que se encuentra en el fichero *main.jar*. El objetivo es ver como si se actualiza esta aplicación en el servidor el usuario siempre carga la última versión de manera transparente.

En el siguiente trozo de código se encuentra la parte en la que se lanza la aplicación al pulsar el botón, he suprimido toda la parte del interfaz gráfico ya que no tiene demasiado interés.

```
import org.nanode.jnlp.*;                                          *1
import org.nanode.launcher.cache.FileCache;
import org.nanode.launcher.cache.Cache;

private void launch() {

  try {
    final Cache cache = FileCache.defaultCache();                  *2
    final URL url = new URL("http://localhost:8080/jnlp/jnlp.jnlp");
    new Thread() {
      public void run() {
        try {
          JNLPParser.launchJNLP(cache,url,true);                   *3
        }
        catch (ParseException pe) {
          pe.printStackTrace();
        }
      }
    }.start();
  }
  catch (MalformedURLException murle) {
    murle.printStackTrace();
  }
}
```

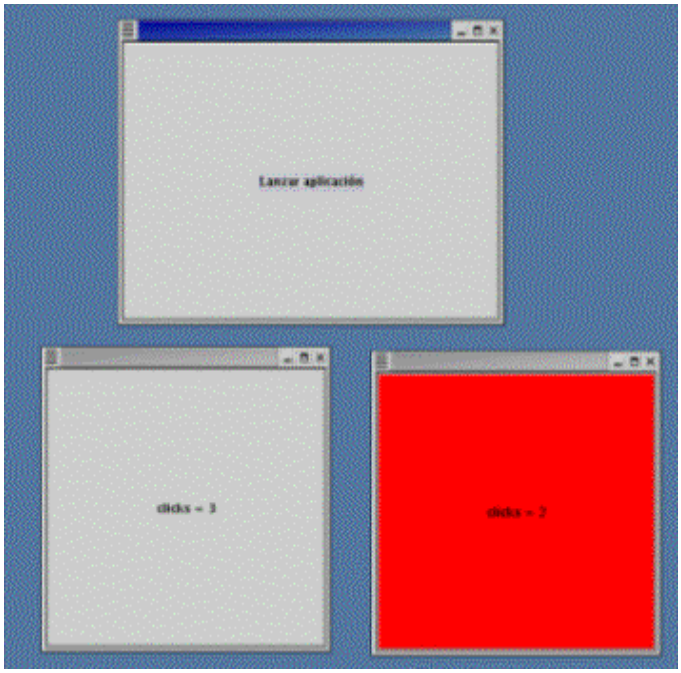
El proceso es muy sencillo:

- Lo primero que hay que hacer es importar las clases necesarias de la librería OpenJnlp (*1).
- Una vez hecho eso hay que establecer la caché de aplicaciones (*2), que es donde *OpenJNLP* irá almacenando las aplicaciones que un usuario va ejecutando para poder ejecutarlas cuando no exista una versión más actualizada en el servidor. La implementación por defecto de la caché de aplicaciones se basa en ficheros. El directorio donde se guardan dichas aplicaciones es el *.jnlp/cache/vendor/title* a partir del directorio que tenga como valor el atributo user.home, donde vendor y title se corresponden con los campos del descriptor jnlp.
- Para finalizar el proceso se lanza la aplicación (*3). El método launch se encarga automáticamente de comprobar si existen actualizaciones de la aplicación que especificamos en la url que se le pasa como parámetro y ejecuta dicha aplicación en un nuevo hilo.

El conjunto de archivos que componen este ejemplo se encuentra en [26]. Lo primero que hay que hacer es configurar nuestro servidor web como se vio en los anteriores apartados para que el fichero *ejemplo3.jnlp* y las librerías que contienen las diferentes versiones de la aplicación que se va a ejecutar (*main.jar* y *main2.jar*) sean accesibles.

Los dos ficheros jar contienen la misma aplicación que se ha utilizado hasta ahora como ejemplo salvo que la segunda versión dibuja el botón de color rojo. El proceso de prueba es el siguiente:

- Primero se ejecuta la clase Desktop.class con el comando java -cp path_a_librerías_jnlp Desktop. Esta clase se encargará de lanzar el descriptor *JNLP ejemplo3.jnlp*, que utiliza el fichero *main.jar* por lo que deberá aparecer la aplicación con el botón normal.
- A continuación, para poder observar todo lo que se ha comentado es necesario sobreescribir en el servidor el fichero *main.jar* con la nueva aplicación, *main2.jar*.
- Por último se vuelve a ejecutar la clase Desktop.class como se explicó en el primer punto. En este caso, como la aplicación ha sido modificada, se bajará la nueva versión y el botón aparecerá de color rojo.



En este caso, y a diferencia del apartado anterior, a pesar de ejecutarse todas las aplicaciones en la misma máquina virtual, cuando se pulsa en uno de los botones no se actualiza el contador en el resto de ventanas. ¿Por qué sucede esto si el contador es estático? Esto se debe a que *OpenJNLP* utiliza un cargador de clases (*ClassLoader*) diferente para cargar cada aplicación y en el lenguaje Java dos instancias de una misma clase que hayan sido cargadas por distintos cargadores de clase se comportan exactamente igual que si fueran clases diferentes.

El apartado anterior también podría haberse adaptado para que se produjese este efecto pero se ha dejado así por simplicidad y para mostrar que a veces es necesario tener cuidado con este tipo de variables estáticas.

Retomando el tema del escritorio empresarial, está claro que la solución a los problemas de mis amigos, y por extensión, de toda la gente que quiera aprovechar las ventajas de la carga dinámica de aplicaciones que ofrece *JNLP* es muy sencilla. En lugar de lanzar las aplicaciones de la manera tradicional (creando una instancia de la aplicación en un nuevo hilo y ejecutándola), se puede utilizar *OpenJNLP* para lanzar estas aplicaciones, de modo que el proceso de actualización y ejecución de las mismas se automatiza completamente. Además esta solución permite aprovechar todo el código que haya sido realizado y no obliga a crear un cargador de aplicaciones diferente, ni migrar hacia una especie de "escritorio web", sino que permite mantener esos sistemas ya disponibles, como el escritorio empresarial de este ejemplo, con tan sólo modificar la forma con la que cargan las aplicaciones.

Las ventajas de esta aproximación son grandísimas en cuanto a mantenibilidad y facilidad de despliegue de aplicaciones. Con esta solución, modificar una aplicación determinada no implica actualizarla en todos nuestros usuarios sino que la actualización se hará de una manera simple y transparente, al tiempo que nuestros usuarios siempre utilizan la última versión de nuestro software.

Referencias

- 1. JSR 56 - Java Network Launching Protocol and API specification, http://jcp.org/jsr/detail/056.jsp
- 2. Java Web Start, http://java.sun.com/products/javawebstart
- 3. Java Web Start Developer?s Pack, http://java.sun.com/products/javawebstart/download-jnlp.html
- 4. Java Web Start Architecture, http://java.sun.com/products/javawebstart/architecture.html
- 5. Official Java Web Start FAQ, http://java.sun.com/products/javawebstart/faq.html
- 6. Foro en SUN sobre Java Web Start, http://forum.java.sun.com/forum.jsp?forum=38
- 7. Java Web Start descarga del código fuente, http://www.sun.com/software/communitysource/javawebstart/download.html
- 8. Java Web Start en Mac OS X, http://developer.apple.com/java/javawebstart/
- 9. Unofficial Java Web Start/*JNLP* FAQ, http://www.vamphq.com/jwsfaq.html
- 10. Open*JNLP*, http://openjnlp.nanode.org/

- 11. Rachel, http://rachel.sourceforge.net/
- 12. Fontanus *JNLP* Wrapper, http://zydego.fontanus.net/jnlp/wrapper/
- 13. Java URL, http://www.amherst.edu/~tliron/javaurl/
- 14. Juniper, http://sourceforge.net/projects/juniper/
- 15. Deploying Sofware with *JNLP* and Java Web Start, John Zukowski, http://developer.java.sun.com/developer/technicalArticles/Programming/jnlp/
- 16. Java Web Start to the Rescue, Raghavan N. Srinivas, http://developer.java.sun.com/developer/technicalArticles/JavaLP/javawebstart/
- 17. Developing and distributing Java applications for the client side, Steven Kim, http://www-106.ibm.com/developerworks/java/library/j-webstart/
- 18. Packaging *JNLP* Applications in a Web Archive, http://java.sun.com/products/javawebstart/1.2/docs/downloadservletguide.html
- 19. Apache Tomcat, http://jakarta.apache.org/tomcat/
- 20. SUN?s RMI Tutorial, Ann Wollrath y Jim Waldo, http://java.sun.com/docs/books/tutorial/rmi/
- 21. Java Remote Method Invocation (RMI), http://java.sun.com/products/jdk/rmi/
- 22. RMI Architecture and Functional Specification, ftp://ftp.java.sun.com/docs/j2se1.4/rmi-spec-1.4.pdf
- 23. Getting Started Using Java RMI, http://java.sun.com/j2se/1.4/docs/guide/rmi/getstart.doc.html
- 24. Java Remote Method Invocation ? Distributed Computing in Java, http://java.sun.com/marketing/collateral/javarmi.html
- 25. Fundamentals of RMI, http://developer.java.sun.com/developer/onlineTraining/rmi/RMI.html
- 26. ódigo fuente y ejemplos de este artículo, http://www.javahispano.com/download/ejemplos/jws.tar.gz

Martín Pérez Mariñán es desarrollador Java desde hace tres años. **SUN Certified Programmer** for Java 2 Platform y **SUN Certified Developer** for Java 2 Platform, además es ingeniero de sistemas por la universidad de La Coruña. En su vida laboral ha tocado casi todas las APIs contenidas en Java y actúalmente está trabajando para **INTECNO del Grupo DINSA** desarrollando proyectos empresariales con J2EE.
Cuando no está metido en su trabajo intenta realizar aportaciones a publicaciones escritas u online y el resto del tiempo lo dedica a su novia y al futbol.
Para cualquier duda o tirón de orejas, e-mail a: martin@kristalnetworks.com