Escuela Técnica Superior de Ingeniería **Informática**

etsinf

UNIVERSIDAD POLITECNICA DE VALENCIA

Proyecto final de carrera

Diseño e implementación de una arquitectura software OSGi para gestionar repositorios remotos de componentes

Autor:
Sergio Cava Montesinos

Director:
Joan Fons i Cors

Nacho Mansanet Benavent

Mayo de 2011

Tabla de contenido

Apartado 1
Introducción

Contexto histórico

Hoy en día la información se considera uno de los mayores bienes de los que disponemos y a su vez el trabajo intelectual tiende a ser cada vez más importante en relación con el trabajo físico. Las transacciones en torno a esta información son cada vez más comunes en el mercado. La expansión y la creación de nuevos tipos de instituciones que trabajan con información como, por ejemplo, universidades, bibliotecas, patentes de empresas comerciales, etc. se considera indicativa del grado de evolución tecnológica alcanzado por una civilización.

Curiosamente, esto pone de relieve la importancia adquirida por los nuevos medios de los que disponemos en relación con los sistemas de distribución descentralizada, tales como Internet, cuando el precio de la distribución de información tiende a cero con cada vez más eficientes herramientas para distribuir información y la creciente cantidad de la misma que se distribuye a una cada vez mayor base de clientes.

Una de estas herramientas que ha permanecido en auge durante la última década es el teléfono móvil. Gracias a los avances en la escala de integración de las arquitecturas informáticas, cuya circuitería se ha visto reducida hasta unas cuantas milmillonésimas partes de metro (entre 32 y 65 nm en el caso de los procesadores) se ha podido crear la nueva y predominante generación de teléfonos móviles inteligentes, también conocidos como *smartphones*.

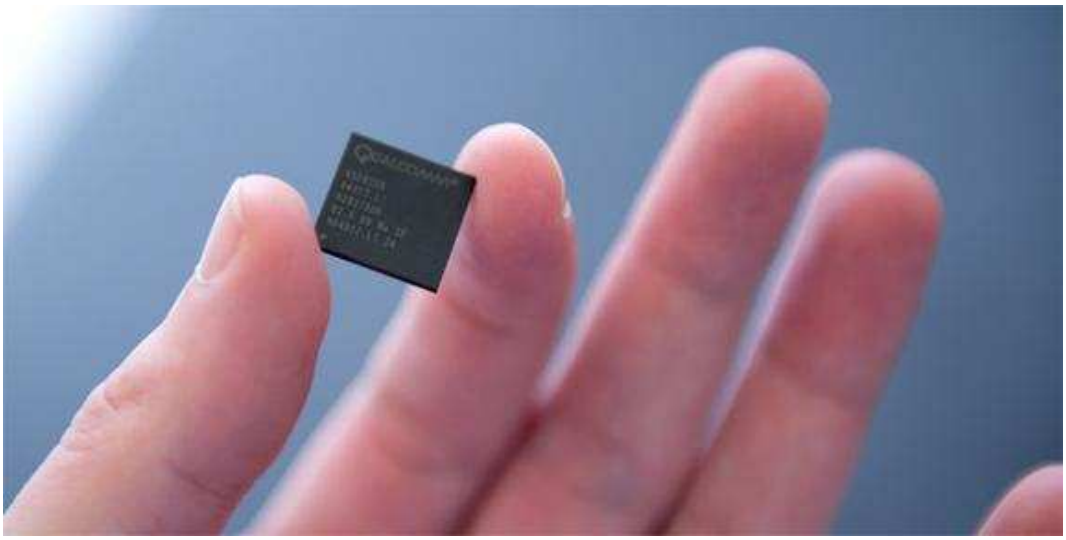


Ilustración 1. Procesador Qualcomm ARM Cortex A8, 1 GHz con tecnología de 45 nm

Estos teléfonos tienen características similares a las de un ordenador personal y se distinguen por muchas propiedades, que incluyen, entre otras, pantallas táctiles, un sistema operativo así como conectividad a Internet. Uno de los puntos fuertes del acceso a Internet a través de un teléfono móvil es, por evidente que suene, que puedes acceder desde casi cualquier lugar ya sea a través del propio proveedor de telefonía móvil, tanto por su red estándar como por la red 3G, así como a través de otras tecnologías inalámbricas como pueden ser el Wi-Fi e incluso el bluetooth.

Tal es la aceptación en el mercado de los dispositivos móviles del uso de smartphones que los analistas de la industria dividen este mercado en dos categorías: smartphones y todo lo demás (ordenadores portátiles, PDAs, tabletas digitales, etc.). De acuerdo con un reciente estudio llevado a cabo por la empresa Gartner, los smartphones se han llevado el 19% del total de ventas de dispositivos móviles a nivel mundial en el año 2010, incrementando así las ventas en un 51% y subiendo año tras año.

Los *smartphones* permiten, a su vez, la instalación de programas para incrementar el procesamiento de datos y la conectividad. Estas aplicaciones pueden ser desarrolladas por el fabricante del dispositivo, por el operador o por un tercero. Todas estas características hacen de los teléfonos móviles de hoy en día una herramienta portátil, liviana y muy potente cuyo límite lo pone la propia imaginación.

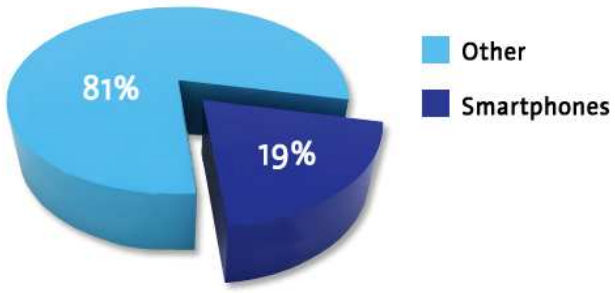


Ilustración 2. Estado de las ventas en el mercado de los dispositivos móviles a nivel mundial

Ahora imaginemos una aplicación para móvil que sea capaz de buscar, instalar, ejecutar y desinstalar otras aplicaciones o partes de ella, a medida que las circunstancias lo requieren, sirviéndose de las facilidades que ofrece internet, ya sea de manera automática y transparente para el usuario o bajo petición explícita del mismo. Podría tratarse de una aplicación topográfica a la que le indicásemos que queremos medir cierta área del terreno, tras lo cual la aplicación buscaría en la nube de repositorios en internet los módulos necesarios para medir áreas y al intentar instalarlos vería que esos módulos dependen de un tercero que recoge las coordenadas del GPS, con lo que también se instalaría. No haría falta reiniciar el dispositivo ni la aplicación para empezar a usar las nuevas funcionalidades y el usuario final no ha tenido que perder el tiempo en resolver complejas dependencias ni su dispositivo necesita tener la memoria cargada con módulos que no vaya a usar.

Esa es la idea principal del proyecto: un núcleo de aplicación (o *kernel*) que facilite la instalación modular así como otras tareas haciendo uso de repositorios en internet, cuya utilidad principal será su aplicación en tecnologías móviles y sistemas pervasivos.

De unos años hasta ahora la tecnología para estos ámbitos que ha destacado sobre las demás es OSGi (Open Service Gateway Initiative), debido principalmente al respaldo de un numeroso grupo de empresas internacionales. OSGi es un estándar con una arquitectura orientada a servicios. Define un modelo cooperativo donde las aplicaciones pueden dinámicamente descubrir y utilizar otros servicios, es decir, puede detectar cuándo un servicio está disponible o deja de estarlo. Es posible también una administración flexible y remota de estas aplicaciones así como de los servicios que ellas proveen.

Siendo OSGi la tecnología aún era necesaria una plataforma en la que trabajar con repositorios y, habiendo varias, una de las más potentes y versátiles además de ser de código abierto resultó ser Apache Felix, siendo la plataforma finalmente elegida para trabajar.

Apache Felix nos permite el ensamblaje dinámico de aplicaciones con paquetes (*bundles*) de OSGi y nos ofrece una base sobre la que empezar a trabajar con repositorios.

Objetivo del proyecto

El objetivo de este proyecto persigue diseñar e implementar un núcleo de aplicación o *kernel* capaz de gestionar las funcionalidades del entorno y los recursos haciendo uso de repositorios remotos. Este núcleo ha de ser capaz de buscar paquetes o módulos en repositorios, resolver dependencias entre ellos, instalarlos, ejecutarlos, desinstalarlos y actualizarlos. Las funcionalidades descritas aportan al entorno facilidades y ventajas entre las que destacan:

- Extensibilidad: al tratarse de un sistema modular pueden instalarse, desinstalarse y actualizarse paquetes sin afectar al funcionamiento principal de la aplicación, aportando nuevas funcionalidades o desechándolas en caso de ser necesario.
- Flexibilidad: su capacidad de ser extensible nos permite gestionar de manera eficaz los recursos empleados, así como los repositorios remotos nos posibilitan la instalación de recursos en cualquier momento y lugar tal y como los vamos necesitando.
- Transparencia: el kernel desarrollado es el encargado de buscar paquetes, resolver dependencias, instalar el recurso más adecuado disponible o desinstalar lo necesario para que el usuario simplemente disponga de las funcionalidades que ha requerido sin siquiera darse cuenta de que por detrás ha habido un proceso de búsqueda e instalación/actualización.

Estructura del documento

Apartado 1: Introducción

En este apartado se pretende situar al lector en el marco tecnológico actual con respecto a los dispositivos móviles, explicando cómo hemos llegado hasta él sin olvidarnos, además, de las expectativas que se prevén en un futuro próximo. Asimismo se plantean los objetivos perseguidos con la realización del proyecto. Finalmente se realiza una concisa introducción a los contenidos que serán abordados a lo largo del documento.

Apartado 2: Contexto tecnológico

El siguiente apartado tiene como objetivo enumerar las características más importantes de OSGi e introducir al lector en los fundamentos de esta tecnología y de los repositorios remotos con el objetivo de servir como base para ilustrar la solución propuesta y cómo gracias a esta tecnología se pueden implementar aplicaciones modulares que pueden agregarse o eliminarse de forma transparente al usuario.

Apartado 3: Diseño arquitectónico

En el tercer apartado abordaremos el análisis realizado de los requisitos necesarios para llevar a cabo la solución al problema que se nos plantea en el proyecto, mostrando los distintos diseños y diagramas de clases de las soluciones propuestas así como las funcionalidades de sus interfaces.

Apartado 4: Implementación del caso de estudio

Este apartado se centrará en la implementación de la aplicación explicando las partes más destacadas del código paso a paso y cómo se ha hecho para alcanzar finalmente las funcionalidades deseadas.

Apartado 5: Conclusiones del proyecto

Aquí pretendo dejar plasmadas mis sensaciones a nivel personal relativas al desarrollo del proyecto final de carrera.

Apartado 6: Referencias

El último apartado incluye las referencias bibliográficas en las que me he apoyado para la realización del proyecto.

Apartado 2

Contexto tecnológico

Introducción a la plataforma

OSGi nos plantea una solución a un problema con el que nos encontramos frecuentemente en el mundo del desarrollo de software hoy en día.

El problema

La complejidad del software está creciendo a un ritmo alarmante.

Hoy en día gran parte de esta complejidad es debida a que los actuales ciclos de producción exigen ser cada vez más cortos y se requieren muchas más funcionalidades, además de ser necesario el desarrollo de variaciones del mismo producto para así adaptarse a distintos sistemas operativos o distintos dispositivos de hardware. Estas tendencias han provocado que los gastos de los fabricantes relativos al desarrollo de software sean porcentualmente mayores con respecto al cómputo global de los gastos en cualquiera de sus proyectos.

Hoy en día, el desarrollo de software consiste en gran medida en adaptar funcionalidades existentes para que funcionen en un nuevo entorno. En los últimos 20 años, han sido desarrolladas y puestas al alcance de todos un gran número de librerías y soluciones software estándares, muchas de las cuales son ampliamente utilizadas en los productos actuales. Sin embargo, el uso de estas librerías no está exento de problemas. La integración de muchas librerías diferentes puede ser desconcertante, ya que muchas de ellas han aumentado su complejidad y requieren de sus propias librerías para funcionar, aunque las funcionalidades que estas últimas le aporten no sean necesarias para el producto.

Esta tendencia hace que los productos de software hayan de someterse a un ciclo de pruebas muy exigente. Añadámosle que las liberías evolucionan de manera asíncrona para ver con más claridad por qué el desarrollo de software es tan costoso hoy en día.

Hoy en día los entornos de desarrollo de software se centran en la escritura de nuevos programas, en lugar de la integración de programas existentes en nuevos sistemas. Es un hecho, además, que la integración de código existente se ha convertido en gran parte de la labor de los desarrolladores de software, por lo tanto es necesario que existan herramientas que normalicen los aspectos de integración de software para que la reutilización de componentes existentes se convierta en fiable, robusta y barata.

La solución

La tecnología OSGi permite a Java aprovechar las ventajas de los sistemas modulares dinámicos. La plataforma de servicios OSGi proporciona tal funcionalidad a Java que lo convierte en el entorno principal para la integración de software y por lo tanto para desarrollo. Java proporciona la portabilidad que se requiere para poder soportar productos en muchas plataformas diferentes. La tecnología OSGi facilita las bases estandarizadas que posibilitan la construcción de aplicaciones a partir de pequeños, reutilizables y modulares componentes. La plataforma de servicios OSGi suministra funciones a fin de poder cambiar la composición de forma dinámica en el dispositivo sin necesidad de reiniciarlo. Para minimizar el acoplamiento, así como para que éste sea pertinentemente manejado, la tecnología OSGi provee una

arquitectura orientada a servicios que permite que sus componentes se descubran unos a otros de forma dinámica y así puedan colaborar entre sí.

OSGi

Sobre la alianza OSGi

La alianza OSGi (The OSGi Alliance) fue fundada por Ericsson, IBM, Motorola, Sun Microsystems y otros en marzo de 1999 (antes de convertirse en una corporación sin fines de lucro era llamada “the Connected Alliance”).

Entre sus miembros encontramos más de 35 compañías de muy diferentes areas de negocio, como por ejemplo Nokia, Oracle, Red Hat, Samsung, Telefónica, Siemens, o las ya citadas IBM, Ericsson, Motorola o Sun Microsystems (entre muchas otras).

La Alianza tiene una Junta Directiva que prevé la gobernanza global de la organización. Los oficiales de OSGi tienen diversas funciones y responsabilidades para respaldar a la Alianza.

El trabajo técnico se lleva a cabo dentro de los grupos de expertos (GE), fletado por el Consejo de Administración, y el trabajo no técnico se lleva a cabo en varios grupos de trabajo y comités.

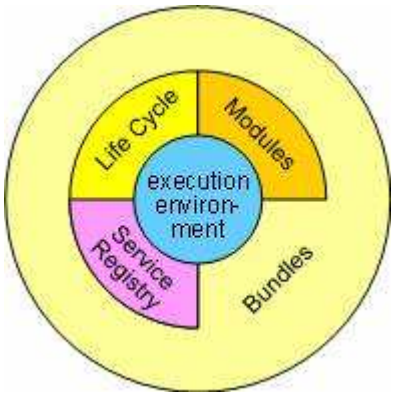
El trabajo técnico realizado en los grupos de expertos incluye el desarrollo de especificaciones, implementaciones de referencia y las pruebas de cumplimiento. Estos grupos de expertos han realizado cuatro versiones principales de las especificaciones OSGi (como la del 2007).

Hay grupos de expertos dedicados al mundo de las empresas, de los móviles, de los vehículos y de plataformas centrales.

The Enterprise Expert Group (EEG) es el grupo de expertos más nuevo y se dedican al desarrollo de aplicaciones para servidores.

En noviembre de 2007, el Residential Expert Group (REG) comenzó a trabajar en especificaciones que permitiesen ser administradas de forma remota. En octubre de 2003 Nokia, Motorola, IBM, ProSyst y otros miembros de OSGi formaron un grupo de expertos en tecnología móvil (Mobile Expert Group, o MEG) para definir una plataforma de servicios de información móvil para la próxima generación de teléfonos móviles inteligentes, ahora conocida como R4.

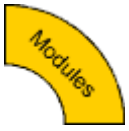
El framework de OSGi



El *framework* es el componente básico de la especificación OSGi. Ofrece un entorno estándar para las aplicaciones (llamadas bundles). El framework se divide en una serie de capas:

- L0: Entorno de ejecución.
- L1: Módulos.
- L2: Gestión del ciclo de vida.
- L3: Registro de servicios.

La capa L0, conocida como el entorno de ejecución, es la especificación del entorno de Java. Las configuraciones de Java 2 y perfiles como J2SE, CDC, CLDC, MIDP, etc. son entornos de ejecución válidos. La plataforma OSGi ha estandarizado también un entorno de ejecución basado en la *Foundation Profile* y en una pequeña variación que especifica los requisitos mínimos de un entorno de ejecución para que sea útil para los bundles de OSGi.



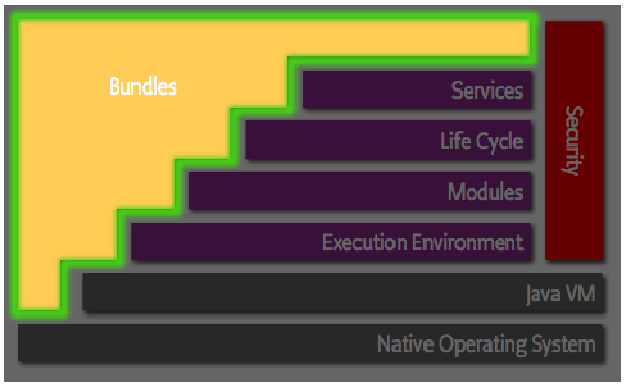
La capa L1 -módulos- define las directivas de carga de clases. Aunque basado en Java, el framework de OSGi añade un potente modelo de carga dinámica de clases, añadiéndole modularidad al sistema. En Java hay normalmente un único *classpath* o variable de entorno que contiene todas las clases

y recursos. Esta capa de OSGi agrega clases privadas para los módulos así como control de enlaces entre ellos.

La capa de gestión del ciclo de vida L2 permite que los bundles puedan ser dinámicamente instalados, ejecutados, detenidos, actualizados y desinstalados, además de resolver las dependencias de los módulos en tiempo de carga. Esta capa es utilizada por los bundles en la clase de carga pero además añade una API para manejar los módulos en tiempo de ejecución. Más adelante la veremos más detalladamente.

La capa L3 añade el Servicio de Registro. El servicio de registro proporciona un modelo de cooperación para bundles dinámicos. Los bundles pueden cooperar a través de los medios tradicionales de intercambio, pero este sistema no es el más compatible con la instalación y desinstalación dinámica de código. El servicio de registro proporciona un modelo global para compartir objetos entre bundles. Se define una serie de eventos para manejar el intercambio de información entre ellos. Esta capa es de gran utilidad a la hora de gestionar los servicios.

Bundles



Un bundle es un grupo de clases Java y de otros recursos acompañados con un archivo de manifiesto (MANIFEST.MF) en el que se detallan todos sus contenidos, así como sus dependencias en otros servicios necesarios para proporcionar al grupo de clases Java incluídas comportamientos más sofisticados, hasta el punto de considerer al grupo entero como un único componente.

Debajo tenemos un ejemplo tipo de un archivo MANIFEST.MF con cabeceras OSGi:

```
Bundle-Name: Hola Mundo

Bundle-SymbolicName: com.ejemplo.holamundo

Bundle-Description: Un bundle Hola Mundo

Bundle-ManifestVersion: 2

Bundle-Version: 1.0.0

Bundle-Activator: com.ejemplo.Activator

Export-Package: com.ejemplo.holamundo;version="1.0.0"

Import-Package: org.osgi.framework;version="1.3.0"
```

El significado de cada una de las cabeceras del ejemplo es el siguiente:

Bundle-Name: Define un nombre para el bundle.

Bundle-SymbolicName: La única cabecera requerida. Con ella se asigna un identificador único al bundle, basado en la convención de nombres de dominio inverso (también utilizada por los paquetes Java).

Bundle-Description: Una descripción de la funcionalidad del bundle.

Bundle-ManifestVersion: Indica la especificación de OSGi a usar para leer el bundle.

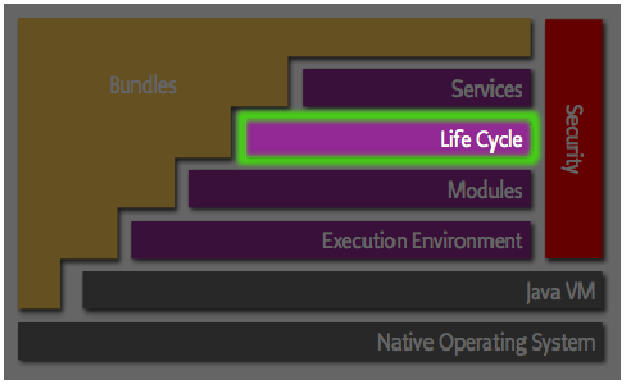
Bundle-Version: Designa un número de version al bundle.

Bundle-Activator: Indica el nombre de la clase que ha de ser invocada una vez el bundle sea activado.

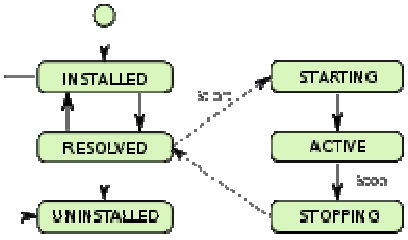
Export-Package: Define qué paquetes Java contenidos en el bundle serán puestos a disposición de los demás.

Import-Package: Indica los paquetes Java requeridos para poder cumplir con las dependencias necesarias para que el bundle funcione.

Ciclo de vida



Como ya hemos visto, la capa del ciclo de vida da la capacidad dinámica de instalación, ejecución, detención, actualización y desinstalación a los bundles. Éstos, a su vez, dependen de la capa de módulos para resolver sus dependencias , pero añadiendo un API para manejarlos en tiempo de ejecución. La capa del ciclo de vida introduce dinámicas que normalmente no son parte de una aplicación, y sus operaciones están completamente protegidas por la arquitectura de seguridad. Los diferentes estados por los que puede pasar un bundle son los siguientes:



ıstración 3. Ciclo de vida de un bundle.

Así pues, los diferentes estados por los que puede pasar un bundle (INSTALLED, RESOLVED, STARTING, ACTIVE, STOPPING y UNINSTALLED) son descritos a continuación:

INSTALLED

Se ha instalado correctamente el bundle.

RESOLVED

Todas las clases que el bundle necesita están disponibles. Este estado indica que el bundle está preparado para ser ejecutado o que ya se ha detenido.

STARTING

Está comenzando la ejecución del bundle, se ha hecho una llamada al método BundleActivator.start pero todavía no se ha recibido su respuesta. Cuando el bundle tenga una política de activación permanecerá en este estado hasta que sea activado de acuerdo con la misma.

ACTIVE

El bundle ha sido activado satisfactoriamente y está en ejecución. El método start de su activador (BundleActivator.start) ha sido llamado y se ha recibido su respuesta.

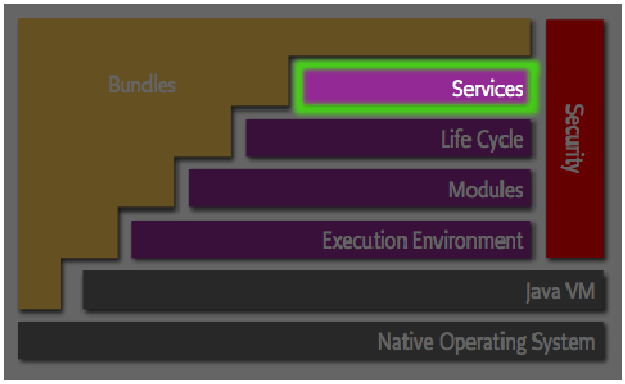
STOPPING

Está siendo detenido el bundle. Se ha hecho una llamada al método BundleActivator.stop pero todavía no ha sido devuelta ninguna respuesta.

UNINSTALLED

El bundle ha sido desinstalado y ya no podrá cambiar a ningún otro estado.

Servicios



¿Qué son?

Un servicio OSGi es una instancia de un objeto java registrada en un framework OSGi con unas propiedades dadas. Cualquier objeto java puede ser registrado como servicio, siempre que implemente una interfaz conocida.

El cliente de un servicio es siempre un bundle de OSGi, o lo que es lo mismo, una pieza de código java que puede ser iniciada a través de la interfaz BundleActivator.

Cada bundle puede registrar cero o más servicios propios, y a su vez puede utilizar cero o más servicios ajenos. No hay, pues, limitación alguna en el número de servicios, más allá de la limitación dada por la capacidad de la memoria de nuestra máquina o de las políticas de seguridad de java.

Tanto la publicación como el registro y uso de servicios pueden ser restringidos mediante el uso de políticas de seguridad de java.

Registrar un objeto como un servicio es muy sencillo:

```
Long      i     = new Long(20);

Hashtable props = new Hashtable();

props.put("descripción", "Un valor long");

bc.registerService(Long.class.getName(), i, props);
```

Nota: un servicio puede ser registrado con muchas interfaces distintas. En ese caso, el objeto debe implementar todas las interfaces.

¿Para qué pueden usarse?

El concepto de servicio tiene una finalidad muy general. Algunos ejemplos son:

- Exportar funcionalidades de un bundle a otros.
- Importar funcionalidades de otros bundles.
- Registrar oyentes (listeners) para eventos de otros bundles.
- Exponer aparatos externos, tales como aparatos UPnP o incluso hardware, a otros bundles OSGi.
- Exponer código java corriendo en OSGi a una red externa.
- Configurar bundles.

Generalmente el uso de servicios es el más indicado para que los bundles se comuniquen entre ellos.

¿Cómo acceder a los servicios?

Los servicios son siempre accedidos a través de un ServiceReference, que únicamente apunta a un objeto de servicio.

Para acceder a un servicio hay que seguir el siguiente procedimiento:

1. Crea un ServiceReference.
2. Obtén el objeto de servicio usando BundleContext.getService() y asígnaselo al ServiceReference del primer paso.

Los ServiceReference se obtienen normalmente:

1. Usando BundleContext.getService().

   Este método devuelve el servicio de mayor rango de la clase dada, o null.

2. Usando BundleContext.getServices().

   Este método devuelve todos los servicios que concuerden con el contexto, o null.

3. A través del retorno de un oyente ServiceListener.
4. Usando la clase ServiceTracker.

En todos los casos excepto el primero se le permite al cliente especificar un conjunto de propiedades que el servicio debe poseer. Tales propiedades se especifican usando filtros LDAP. Cero o más servicios pueden coincidir con un filtro dado.

Los filtros suelen contener un nombre de clase y un grupo de propiedades. En el siguiente ejemplo se busca un servicio HTTP que tenga la propiedad port (puerto) igual a 80:

```
(&(objectclass=org.osgi.service.http.HttpService)(port=80))
```

Tan pronto haya un ServiceReference disponible, el objeto de servicio puede ser accedido usando un casting:

```
HttpService http = (HttpService)bc.getService(sr);
```

Después de esto el objeto de servicio puede ser accedido como se accedería a cualquier otro objeto java. De hecho es un objeto totalmente normal de java, y más concretamente es el mismo objeto que el bundle que registró el servicio creó.

¿Cómo liberar un servicio?

Tan pronto como deje de ser necesario un servicio, el cliente debe liberarlo llamando al método BundleContext.ungetService():

```
bc.ungetService(sr);
```

Todos los servicios son liberados automáticamente cuando el bundle cliente se detiene. No es necesario, por tanto, ningún tipo especial de servicio de limpieza en BundleActivator.stop().

Es necesario remarcar que el servicio debería ser liberado únicamente cuando no se le dé ningún uso. Ciertos servicios podrían mantener un estado para cada bundle que los usa, y liberar este servicio también liberaría tal estado, lo cual podría ser, o no, lo

que se pretendía. Es por ello que se debe consultar la documentación de cada servicio.

Apartado 3

Diseño arquitectónico

Arquitectura del caso de estudio

Una vez conocido el funcionamiento básico de OSGi empezaremos a desarrollar uno de los posibles modelos a seguir para lograr la funcionalidad que buscamos. En este caso se ha diseñado un modelo basándonos en el patrón de interfaz-implementación de Java que nos permite la reutilización, especialización y, en el caso especial de OSGi, la publicación, descubrimiento y captación de servicios.

Primera solución propuesta

Una primera solución fue la bautizada como "appcontrol", consistente en tres módulos de funciones bien distinguidas.

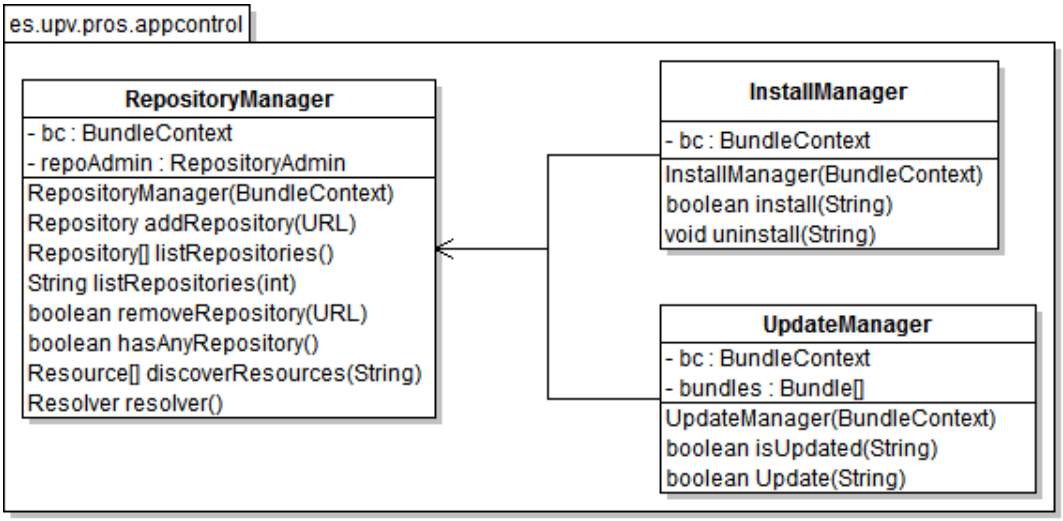


Ilustración 4. Diagrama de clases de appcontrol

- **RepositoryManager**

Es la clase encargada de llevar el control de repositorios, ya sean locales o remotos. A través de esta clase se pueden manejar al

gusto la lista de repositorios que almacena, añadiendo nuevos o eliminando existentes cuando fuese necesario. Cumple, además, con funciones vitales: a través de ella se pueden hacer búsquedas de clases que queramos encontrar en los repositorios y, en caso de haber dependencias, dispone de un método capaz de buscar de manera concurrente los módulos necesarios para una correcta instalación y funcionamiento de aquella clase que precisáramos.

- **InstallManager**

Gracias a esta clase es posible instalar y desinstalar módulos en el framework, pasándole a sus correspondientes métodos un nombre simbólico sobre el que trabajar la instalación o desinstalación. Esta clase depende de la clase RepositoryManager para realizar a través de esta última las búsquedas en los repositorios y resolver las posibles dependencias. Si finalmente ha conseguido instalar lo que se le ha pedido nos devolvería un valor afirmativo (true), y en caso contrario, uno negativo (false). Si bien cumple con el objetivo primigenio, las limitaciones de este módulo fueron fundamentales para realizar ciertas modificaciones en el modelo posterior que ilustra la solución final escogida de la cual hablaremos más adelante.

- **UpdateManager**

Con esta última clase se pretende dotar a la aplicación de herramientas para mantener los módulos instalados al día. Dispone de métodos para actualizar los bundles del framework en caso de haber nuevas versiones disponibles en los repositorios, y por tanto también depende del RepositoryManager para llevar sus tareas a cabo.

Con “appcontrol” tenemos, pues, una aplicación que una vez agregada a nuestro framework es capaz de instalar, desinstalar y actualizar nuevos módulos. Fue en este momento cuando nos preguntamos si había alguna manera de mejorar este servicio, de hacerlo un poco más amigable y transparente si cabe.

Solución definitiva

Ya que el entorno de trabajo de OSGi se basa en contextos de ejecución a través de los cuales se concede el acceso a muchos otros métodos con los que actuar e interrelacionar los distintos módulos, se pensó como un posible sustituto a “appcontrol” un sistema que actuara como una capa superior a estos contextos, extendiendo la usabilidad de los mismos. Así fue como nació el concepto del contexto distribuido “DistributedBundleContext”. La usabilidad del “BundleContext” de OSGi y esta nueva clase sería casi la misma en la práctica, pero con el contexto de ejecución distribuido se conseguiría franquear las limitaciones de los sistemas locales haciendo uso de los repositorios remotos para obtener nuevos servicios que no estén disponibles de forma local, esta vez de forma totalmente transparente al sistema.

Esta nueva solución requiere un nuevo modelo de clases más complejo como podemos apreciar en la ilustración 5.

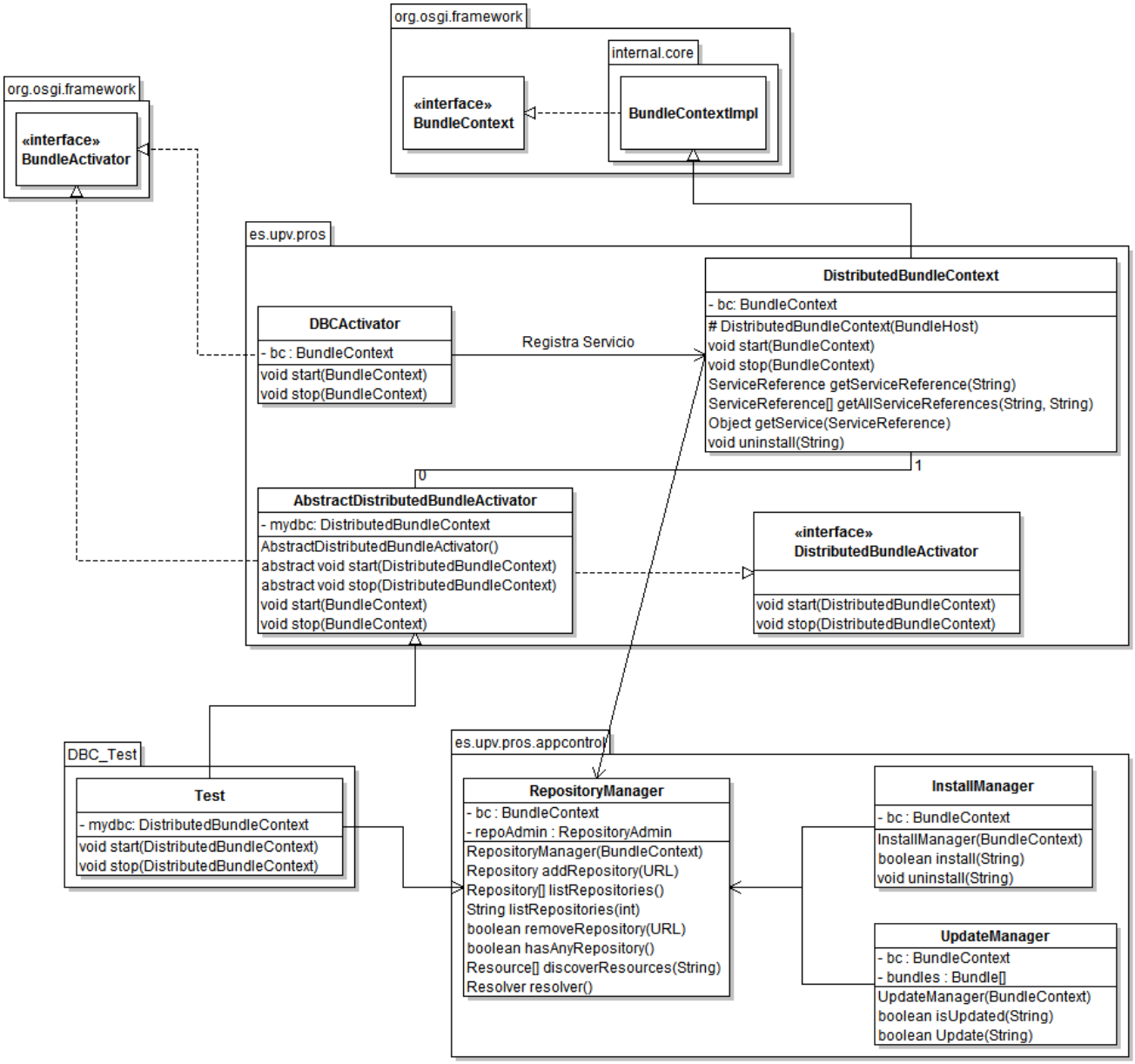


Ilustración 5. Diagrama de clases del DBC.

Interfaces

Las interfaces empleadas son las siguientes:

- **BundleContext**

  Es el contexto de ejecución de un bundle en el framework. El contexto se emplea para conceder acceso a otros métodos para que el bundle pueda interactuar con el framework

  Los métodos de BundleContext le permiten al bundle en cuestión:

  - Suscribirse a eventos publicados por el framework.
  - Registrar objetos de servicio con el servicio de registro del framework.
  - Adquirir referencias a servicios (ServiceReferences) del servicio de registro del framework.
  - Obtener y liberar objetos de servicio para un servicio referenciado.
  - Instalar nuevos bundles en el framework.
  - Obtener la lista de bundles instalados en el framework.
  - Obtener el objeto Bundle de uno de sus bundles instalados.
  - Crear archivos que se pueden almacenar en una memoria persistente provista para el bundle por el framework.

  Se creará un objeto BundleContext y le será proporcionado al bundle asociado al contexto cuando comience su ejecución usando el método BundleActivator.start(BundleContext). El mismo objeto BundleContext le será proporcionado al bundle asociado al contexto cuando sea detenido usando el método BundleActivator.stop(BundleContext).

El objeto BundleContext es usado generalmente de forma privada por su bundle asociado y no está diseñado para un uso compartido con otros bundles en el entorno de OSGi.

Al objeto Bundle asociado a un objeto BundleContext se le llama bundle de contexto.

El objeto BundleContext sólo es válido durante la ejecución de su bundle de contexto, es decir, en el periodo durante el cual el bundle de contexto está en los estados STARTING, STOPPING y ACTIVE. Si se intentase hacer uso del BundleContext fuera de estos periodos, sería lanzada una IllegalStateException.

El objeto BundleContext no ha de ser reutilizado nunca una vez su bundle de contexto haya sido detenido.

El framework es la única entidad capaz de crear objetos BundleContext, y éstos son válidos únicamente en el framework que los creó.

- **BundleActivator**

Sirve para personalizar el comienzo y la parada de un bundle.

Es una interfaz que ha de ser implementada cuando queremos que un bundle pueda ser iniciado o detenido. El framework puede crear instancias del BundleActivator de un bundle tal como lo requiera. Si una instancia del método BundleActivator.start es ejecutada satisfactoriamente se puede asegurar que el método BundleActivator.stop de la misma instancia será llamado cuando el bundle sea detenido. El

framework no debe llamar a un objeto BundleActivator de forma concurrente.

El BundleActivator ha de quedar reflejado en la cabecera Bundle-Activator del manifiesto (MANIFEST.MF). Cada bundle puede especificar un único BundleActivator en el archivo de manifiesto. Los bundles fragmentados no deben, por tanto, tener un BundleActivator.

La forma de la cabecera del manifiesto es:

Bundle-Activator: nombre-de-la-clase

Donde nombre-de-la-clase es un nombre completo de clase de Java.

La clase BundleActivator especificada debe tener un constructor público sin parámetros de entrada para que un objeto de su clase pueda ser creado a través del método Class.newInstance().

- **DistributedBundleActivator**

Es la interfaz análoga a BundleActivator, sólo que esta se centra en dar la funcionalidad de comienzo y parada a la clase DistributedBundleContext, la cual es instanciada a través de su parámetro de entrada.

Si una instancia del método DistributedBundleActivator.start es ejecutada satisfactoriamente se puede asegurar también que el método BundleActivator.stop de la misma instancia será llamado cuando el bundle sea detenido.

De la misma forma que sucede con su clase análoga, aquella clase que implemente a DistributedBundleActivator deberá aparecer reflejada en la cabecera Bundle-Activator del manifiesto.

Como también podemos apreciar en la Ilustración 5, además de interfaces e implementaciones de las mismas, hay una clase cuyo caso es especial, pues se trata de la extensión de la clase que implementa a BundleContext. Esta clase extendida es la que queda registrada como servicio en el entorno y será la que ofrezca las funcionalidades de instalación y desinstalación de nuevos servicios.

Para conocer un poco mejor el funcionamiento interno de la aplicación pasaremos a explicar los principales métodos ilustrándolos con el código en Java correspondiente.

Apartado 4
Implementación
del caso de estudio

Implementación de las clases

La clase que maneja los repositorios es **RepositoryManager**, que basa su funcionamiento en el uso de un objeto de la clase RepositoryAdmin. Es durante la creación del RepositoryManager cuando se obtiene el servicio del RepositoryAdmin de la siguiente manera:

```
public RepositoryManager(BundleContext context)
{
	bc = context;
	try
	{
		ServiceReference referencia =
	bc.getServiceReference(RepositoryAdmin.class.getName());
		if (referencia != null)
		{
			repoAdmin =
		(RepositoryAdmin)bc.getService(referencia);
		}
		else
		{
			throw new Exception("RepositoryAdmin not
available.");
		}
		} catch (Exception e) {e.printStackTrace();}
}
```

La clase **DBCActivator** es la que registra en el entorno el servicio de tipo DistributedBundleContext. Al ser la clase activadora ha de implementar a BundleActivator, extendiendo sus métodos start y stop.

```
public class DBCActivator implements BundleActivator
{
	private BundleContext bc;

	public void start(BundleContext context)
						throws Exception
	{
		bc = context;
		BundleHost bundle = (BundleHost) bc.getBundle();
		DistributedBundleContext newDBC =
				new DistributedBundleContext(bundle);

	context.registerService(DistributedBundleContext.class.
getName(), newDBC, null);
	}

	public void stop(BundleContext context) throws
Exception
	{
		bc = null;
	}

}
```

Como podemos observar primero crea un objeto de tipo DistributedBundleContext para luego registrarlo como un servicio más del entorno.

Inspirada en el activador de bundles clásico se ha implementado la clase abstracta **AbstractDistributedBundleActivator**, que viene a ser la clase puente entre el programador final que quiera aprovechar las ventajas del entorno distribuido y el mismo entorno. Para ello simplemente tendrá que extender esta clase en lugar de extender a BundleActivator. En AbstractDistributedBundleActivator se crean dos métodos análogos a los de BundleActivator, start y stop, pero cuyo parámetro de entrada pasa de ser un

BundleContext a un DistributedBundleContext. Al ser ambos métodos abstractos, toda aquella clase no abstracta que extienda a ésta se verá obligada a implementarlos. De esta manera se consigue, mediante un mecanismo prácticamente idéntico a la creación de activadores de bundles clásicos, dotar al entorno con capacidades de 'computación en la nube'.

La mayor parte del peso de la implementación de la solución propuesta en este proyecto recae sobre la clase **DistributedBundleContext**, ya que es la que lleva a cabo las tareas de instalación, actualización y desinstalación de los servicios requeridos por el usuario tanto a través de los repositorios como del mismo entorno local.

Dispone de métodos concretos para cada una de las tareas que ofrece. Es con el método sobrescrito *getServiceReference* con el que se obtienen los servicios solicitados de forma transparente al usuario, ya provengan del mismo entorno –por encontrarse instalados en él en el momento de su solicitud- o de los repositorios. Este método lo que hace es devolvernos la referencia al servicio (*ServiceReference*) de la clase que responde a la cadena del parámetro de entrada. Es en caso de no estar instalado el servicio que se le requiere cuando lo buscará en los repositorios y lo instalará, devolviéndonos igualmente una referencia al mismo. Si se diese el caso de que no lo encontrase ni de forma local ni remota o la clase indicada no registra ningún servicio nos devolverá un valor nulo (*null*).

Para ello primero necesitaremos diferenciar el nombre simbólico del bundle que buscamos del nombre del paquete al que pertenece. Conociendo ya los estándares marcados por Java en cuanto a nomenclaturas, sabemos que lo que va después del último punto en

el nombre completo es la clase, y lo anterior el paquete al que ésta pertenece. Así, efectuamos lo siguiente:

```
public ServiceReference getServiceReference(String classname)
      {
            java.util.StringTokenizer stkn = new
StringTokenizer(classname,".");
            int divider = stkn.countTokens();
            String paquete = "";
            for (int i = 1; i<divider && stkn.hasMoreTokens();
i++)
            {
                  paquete+=stkn.nextToken();
                  if ((i+1)<divider)
                        paquete+=".";
            }
            String symbolicname = stkn.nextToken();
```

Una vez recogidas ambas variables buscamos alguna coincidencia en el entorno local entre los servicios disponibles:

```
Bundle[] bundles = bc.getBundles();
boolean resultado = false;
for (int i=0; i<bundles.length; i++)
{
      if (bundles[i].getSymbolicName().equals(symbolicname))
      {
            try {
                  bundles[i].start();
                } catch (BundleException e)
                    {
                      e.printStackTrace();
                    }
            ServiceReference sr[] =
                              bundles[i].getRegisteredServices();
            if (sr != null)
```

```
                return sr[0];
            return null;
      }
}
```

En la matriz que hemos llamado ‘sr’ hemos recogido los servicios registrados en nuestro entorno por el bundle cuyo nombre simbólico se corresponde con el que estamos buscando, siempre y cuando lo haya. Si lo hubiera y registrase servicios, nos lo devuelve, pero si a pesar de haber un bundle con el nombre simbólico que buscamos éste no registra ningún servicio nos devolverá *null*.

Una vez comprobado el entorno local miraremos si se puede encontrar el servicio deseado en los repositorios, realizando la búsqueda pertinente sin olvidarnos de manejar la resolución de dependencias. A su vez es importante comprobar cuál de las versiones que nos encontremos en los repositorios es la *mayor* o más actual, para elegir siempre que sea posible la instalación de la misma.

```
Resolver resolver = RepositoryManager.resolver();
String busqueda = "(registeredservice="+symbolicname+")";
if (paquete.length()>0)
      busqueda =
              "(registeredservice="+paquete+"."+symbolicname+")";
      Resource[] resource = null;
      try {
            resource =
                  RepositoryManager.discoverResources(busqueda);
          } catch (Exception e1)
            {
              e1.printStackTrace();
             }
```

```
      int mayor = 0;
      for (int j = 1; j<resource.length; j++)
      {
            mayor =
resource[mayor].getVersion().compareTo(resource[j].getVersion())
>= 0 ? mayor : j;
      }

      try
      {     resolver.add(resource[mayor]);
            resultado = true;
      }
      catch (java.lang.ArrayIndexOutOfBoundsException e)
      { System.out.println(busqueda+" no ha devuelto ningún
resultado."); resultado = false;}
      if (resultado)
      if (resolver.resolve())
      {
            System.out.println("Instalando "+symbolicname+". ");
            resolver.deploy(true);
```

Una vez realizadas las tareas de búsqueda, selección, instalación y ejecución de los servicios que buscábamos –o en su defecto la búsqueda y no resolución de los mismos- realizaremos la misma tarea que habíamos hecho al principio del método: buscar en el entorno local los servicios registrados por los bundles con el fin de poder devolver la referencia a ellos una vez instalados:

```
bundles = bc.getBundles();
            for (int i=0; i<bundles.length; i++)
            {

      if(bundles[i].getSymbolicName().equals(symbolicname))
                  {
                        try {
                              bundles[i].start();
                              } catch (BundleException e)
```

```
                                    {
                                        e.printStackTrace();
                                    }
                            ServiceReference sr[] =
                                    bundles[i].getRegisteredServices();
                            if (sr != null)
                                    return sr[0];
                            return null;
                        }
                }
        }
        else
        {
                System.out.println("No ha sido posible resolver las
dependencias.");
                Requirement[] reqs =
                                resolver.getUnsatisfiedRequirements();
                for (int i = 0; i < reqs.length; i++)
                {
                    System.out.println("Imposible resolver: " +
reqs[i]);
                }
        }
        return null;
}
```

Apartado 5
Conclusiones del proyecto

Valoración personal

Para poder hablar de mis valoraciones personales veo necesario contextualizar el proceso íntegro. Como estudiante de último curso tienes la posibilidad de elegir sobre qué desarrollarás el proyecto final de carrera. No es una decisión fácil. Es por eso que acudí al que ha sido mi profesor y director de proyecto, Joan Fons, para pedir su opinión y orientación al respecto, y me habló de esta tecnología de la que yo no había oído hasta entonces: OSGi. Pronto vi su potencial y me llamó la atención. Me pareció justo lo que buscaba: un reto. Mi proyecto no iba a tratar sobre algo que ya hubiese probado tener conocimientos en la carrera, sino que iba a requerir de una fase de investigación intensa para poder comprender el funcionamiento de esta nueva tecnología para mí.

Así sucedió que los primeros meses fueron un estira y afloja con OSGi para poder hacerme un esquema mental de su funcionamiento, teniendo en cuenta que no era sencillo encontrar documentación amigable y completa. Además de haber pocos tutoriales, la mayoría de ellos resultaron estar incompletos, lo que dejaba ciertas lagunas. Poco a poco, cogiendo un poco de aquí y otro poco de allí, explorando, haciendo mis propias pruebas en java, depurando errores y consultando con Joan y Nacho cuando me bloqueaba, terminé por familiarizarme lo suficiente con OSGi como para poder seguir en paralelo la fase de investigación y empezar “la acción”.

En resumidas cuentas ha sido un reto, con sus vicisitudes, sin duda enriquecedor.

Apartado 6

Referencias

Referencias

[1] *OSGi Tutorial: A Step by Step Introduction to OSGi Programming. Based on the Open Source Knopflerfish OSGi Framework*. **Sven Haiges**. Octubre del 2004.

[2] *OSGi: Open Services Gateway Interface.* **Dr. Diego López de Ipiña González de Artaza**, Universidad de Deusto.

[3] *OSGi Best Practices*. **Peter Kriens**, **OSGi Alliance Technical Director**. 2007 JavaOneSM Conference, Session TS-1419, 2007. IBM Corporation and aQute SARL.

[4] *ProSyst Framework Equinox Edition Package. OSGi APIs.*
http://dz.prosyst.com/pdoc/mbs_equinox_2.0/um/framework/api/osgi/index.html

[5] *ProSyst Documentation Center. OSGi bundles.*
http://dz.prosyst.com/pdoc/mbs_equinox_2.0/um/framework/bundles/prosyst/osgi/introduction.html

[6] *The OSGi Alliance.*
http://www.osgi.org

[7] *Wikipedia, the free encyclopedia.*
http://en.wikipedia.org/wiki/OSGi

[8] *Apache Felix Documentation.*
http://felix.apache.org/site/documentation.html

[9] *R-OSGi: Distributed Applications through Software Modularization*. **Jan S. Rellermeyer, Gustavo Alonso, and Timothy Roscoe,** Department of Computer Science, ETH Zurich, Switzerland.