EUSKO JAURLARITZA
GOBIERNO VASCO

Programa Saiotek 2006

SMARTLAB

Entorno de Trabajo Inteligente Colaborativo y Programable

E1.3 Vigilancia Tecnológica

Plataforma OSGi

Tecnológico Fundación Deusto
Teknologikoa Deustu Fundazioa

HISTORIAL DE CAMBIOS

Versión	Descripción	Autor	Fecha	Comentarios
V0.1	Versión Inicial	Iker Larizgoitia	11/05/2007	

TABLA DE CONTENIDOS

1 OSGI

1.1 Descripción general

En 1999, un grupo de empresas de la industria informática formaron un consorcio mundial con el objetivo de crear una plataforma middleware universal. Este consorcio ha dado lugar a un conjunto de especificaciones, implementaciones de referencia, pruebas y certificaciones bajo el nombre de OSGi (Open Services Gateway initiative).

A lo largo de los últimos años, OSGi está sufriendo un impulso considerable sobre todo en sistemas de integración software y domóticos. Las especificaciones OSGi defininen una arquitectura SOA dentro de una máquina virtual de Java para integración de sistemas heterogéneos. La plataforma OSGi proporciona mecanismos estandarizados para la colaboración de componentes. Además reduce la complejidad global de construcción, mantenimiento y despliegue de aplicaciones.

La plataforma de servicios ofrece las funciones necesarias para poder añadir y eliminar servicios dinámicamente sin necesidad de resetear el sistema. También proporciona mecanismos de descubrimiento que facilitan la colaboración. Dentro de las especificaciones existen también una serie de servicios comunes básicos (Servidores HTTP, configuración, logs, seguridad, XML, …). Sin embargo, debido a su arquitectura basada en plug-ins existen muchos más servicios ya desarrollados por la comunidad investigadora.

1.2 Arquitectura OSGi

OSGi es un contenedor de aplicaciones orientado a servicios que proporciona funcionalidades avanzadas para la gestión del ciclo de vida de las aplicaciones desplegadas en el mismo. La especificación de OSGi define un framework que permite a los desarrolladores crear sus soluciones y desplegarlas proporcionándoles las herramientas necesarias para realizar este proceso de la forma más sencilla.

En OSGi las aplicaciones reciben el nombre de *bundle.* Los *bundles* pueden ser cargados, iniciados y parados en el entorno de forma dinámica pudiendo proporcionar uno o más servicios al mismo de forma que éstos puedan ser utilizados por otras aplicaciones. El framework proporciona además un entorno común de ejecución para las mismas, lo que

significa que las aplicaciones no sólo pueden utilizar servicios unas de otras sino que incluso pueden compartir código entre ellas. El framework proporciona todo el control de dependencias entre *bundles* y servicios que proporcionan para facilitar la reutilización y cooperación entre los mismos. Esto hace que el entorno de ejecución sea más ligero, ya que sólo es necesario mantener una copia de las librerías en memoria y toda la plataforma se ejecuta sobre una única máquina virtual de Java, reduciendo el consumo de recursos de computación.

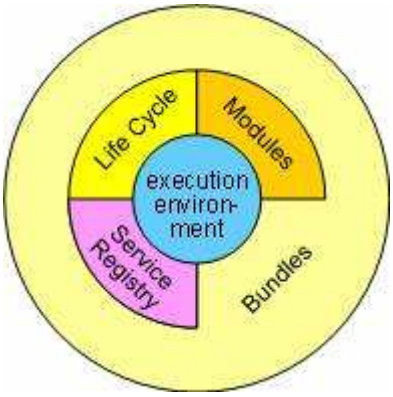

Figura 1.- Arquitectura de OSGi

A nivel general OSGi tiene las siguientes capas (ver Figura 1):

- *Entorno de ejecución*: define la especificación del entorno Java en el que se ejecutará el framework. Aunque la mayoría son válidos (J2SE, CDC, CLDC, MIDP,…) en OSGi también se define un entorno mínimo para los bundles OSGi.
- *Capa de Módulos*: define las políticas de carga de las clases, basado en Java, pero añadiendo modularización.
- *Capa de ciclo de vida*: define el modelo de instalación, arranque, parada, actualización y desinstalación de los bundles.
- *Capa de Service Registry*: el *Service Registry* provee de un modelo de cooperación para los bundles que tiene en cuenta el dinamismo de los mismos. El Service Registry provee un mecanismo general para compartir objetos entre bundles. También están definidos un número de eventos que anuncian el registro y desregistro de eventos.

Sobre estas capas se despliguen los diferentes servicios, los cuales pueden ser de varios tipos: framework, sistema, protocolo,… Las especificaciones OSGi se aplican fácilmente porque la plataforma es una pequeña capa que permite a múltiples componentes Java colaborar dentro de una única máquina virtual. También provee de un modelo de seguridad

que permite crear áreas de permisos entre componentes, facilitando así el modelo de acceso y cooperación. Adoptar la especificación OSGi puede, por lo tanto, reducir el coste de desarrollo de aplicaciones, ya que siguiendo su arquitectura, los componentes pueden ejecutarse en multitud de dispositivos.

1.3 OSGi y AmI

Las características de la especificación OSGi hacen muy adecuada esta plataforma para los entornos de Inteligencia Ambiental. OSGi permite crear un entorno de ejecución donde dispositivos heterogéneos pueden contectarse entre sí de una forma rápida y sencilla. La estrategia a seguir para conseguir esto es que cada dispositivo sea controlado por un *bundle* que implemente los protocolos de comunicación y manejo correspondientes. Los *bundles* son cargados e iniciados en el entorno OSGi exponiendo una serie de servicios que podrán ser utilizados por otras aplicaciones para actuar sobre ese dispositivo.

Los dispositivos que forman parte de un entorno de Inteligencia Ambiental pueden aparecer o desaparecer en cualquier momento, por lo que la plataforma de integración debe ser flexible ante estas situaciones. OSGi permite que el usuario pueda añadir, iniciar, parar o eliminar *bundles* del entorno de forma sencilla y sin la necesidad de detener todo el sistema. Así, la aparición de nuevos dispositivos en el entorno se podrá resolver añadiendo el *bundle* correspondiente al dispositivo en el framework OSGi. Además es posible detectar que nuevos dispositivos han sido añadidos y hacer accesible su funcionalidad a la aplicación de control.

Sobre estos dispositivos se podrán desplegar en la plataforma OSGi los servicios de alto nivel que se encargarán de controlar y gestionar el entorno de Inteligencia Ambiental. La utilización de los dispositivos es muy sencilla una vez que los *bundles* correspondientes han sido desplegados en el framework OSGI. Las aplicaciones de control podrán incluir aplicaciones de razonamiento semántico o motores de reglas que se encargarán de dotar de inteligencia al entorno compuesto de dispositivos. Además, el framework OSGi proporciona servicios de alto nivel para la realización de tareas más complejas.

A pesar de todas las ventajas comentadas anteriormente, OSGi es una plataforma centralizada; para que los servicios ofrecidos por los *bundles* puedan ser descubiertos a través del registro de servicios estos deben estar en el mismo entorno de ejecución, es decir, sobre desplegados sobre la misma instancia de la plataforma OSGi. La especificación

no contempla el descubrimiento y acceso a servicios entre distintas instancias de OSGi. Sin embargo, esto puede ser resuelto a través de la utilización del *bundle* desarrollado en el proyecto R-OSGi que permite que varias instancias de OSGi, ejecutándose en distintas máquinas, puedan descubrir y ejecutar servicios unas de otras.

1.4 OSGi a fondo

1.4.1 Servicios del Núcleo

1.4.1.1 Module Layer

El soporte en la plataforma Java para empaquetado y despliegue de componentes es limitado. El framework OSGi provee de una solución estandarizada y genérica para la modularización en Java.

Bundles

La unidad de modularización es el Bundle. Un bundle es un conjunto de clases Java y recursos que juntos proveen de un servicio. Las características principales de los bundles son:

- Se despliegan como archivos Java ARchive (JAR) y contienen todos los recursos de cualquier tipo necesarios para dar alguna funcionalidad.
- Tienen un fichero llamado manifiesto (en META-INF/MANIFEST.MF) que sirve para describir el contenido y características del bundle. Este manifiesto contiene información que el framework necesita para instalar y activar el bundle. Debe seguir una estructura concreta y se pueden añadir nuevos campos si fuera necesario.
- Se puede incluir información adicional bajo el directorio OSGI-OPT.

A continuación se muestra un ejemplo de las propiedades que pueden ir en el manifiesto.

```
Bundle-Activator: com.acme.fw.Activator
Bundle-Category: test
Bundle-Classpath: /jar/http.jar,.
```

```
Bundle-ContactAddress: 2400 Oswego Road, Austin, TX 74563
Bundle-Copyright: OSGi (c) 2002
Bundle-Description: Network Firewall
Bundle-DocURL: http:/www.acme.com/Firewall/doc
Bundle-Localization: OSGI-INF/l10n/bundle
Bundle-ManifestVersion: 2
Bundle-Name: Firewall
Bundle-NativeCode: /lib/http.DLL; osname = QNX; osversion = 3.1
Bundle-RequiredExecutionEnvironment: CDC-1.0/Foundation-1.0
Bundle-SymbolicName: com.acme.daffy
Bundle-UpdateLocation: http://www.acme.com/Firewall/bundle.jar
Bundle-Vendor: OSGi Alliance
Bundle-Version: 1.1
DynamicImport-Package: com.acme.plugin.*
Export-Package: org.osgi.util.tracker;version=1.3
Fragment-Host: org.eclipse.swt; bundle-version="[3.0.0,4.0.0)"
Import-Package: org.osgi.util.tracker,org.osgi.service.io;version=1.4
Require-Bundle: com.acme.chess
```

Como se puede ver, la mayoría son autoexplicativas, los detalles se pueden encontrar en la propia especificación de OSGi.

Entorno de Ejecución

Cada bundle puede tener restricciones de ejecución en ciertos entornos. Estos entornos se especifican en el manifiesto y pueden ser los siguientes:

```
CDC-1.0/Foundation-1.0
OSGi/Minimum-1.1
JRE-1.1
J2SE-1.2
J2SE-1.3
J2SE-1.4
J2SE-1.5
JavaSE-1.6
PersonalJava-1.1
```

PersonalJava-1.2 CDC-1.0/PersonalBasis-1.0 CDC-1.0/PersonalJava-1.0

Arquitectura de carga de clases

Varios bundles pueden compartir una única máquina virtual. Dentro de esta máquina virtual, los bundles pueden ocultar paquetes y clases a otros bundles, así como compartirlos. Las clases se cargan a partir de *class loaders.* Cada bundle cuenta con uno, que abarca un espacio concreto de clases (classspace), que son todas las clases que un class loader puede instanciar. Estas clases pueden cargarse de:

- El class loader padre (normalmente los paquetes java.*)
- Los paquetes importados.
- Los bundles requeridos.
- El classpath del propio bundle.
- Fragmentos que estén unidos.

Un espacio de clases debe ser consistente, nunca dos clases podrán tener el mismo nombre, sin embargo, sí se permiten varias versiones de la misma clase en espacios diferentes.

Antes de cargar un bundle es necesario *resolverlo*, es decir, todas las restricciones que conforman el espacio de clases (sobre todo las que se importan) debe existir antes de que el código del bundle se instale o ejecute.

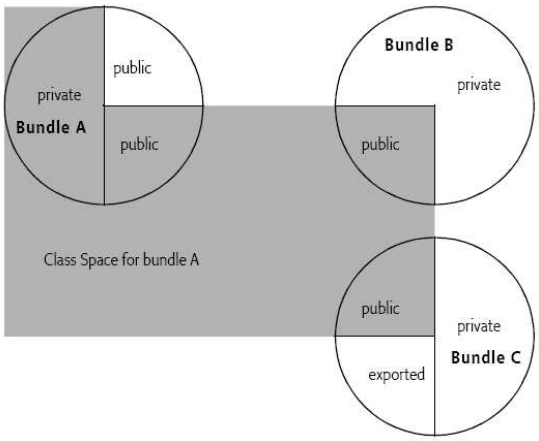

Figura 2.- Ejemplo de espacio de clases para el Bundle A

Estas restricciones se modelan mediante los atributos del manifiesto Import-Package, Export-Package y DynamicImport-Package, los cuales se pueden personalizar con números de versiones concretas, nombres de clases…etc.

1.4.1.2 Life Cycle Layer

Esta capa provee de un API para la gestión del ciclo de vida de los bundles.

Cuando creamos un bundle y queremos instalarlo en el framework, lo primero que se hace es resolverlo, de lo cual se encarga el Module Layer visto en el apartado anterior. Todo bundle cuenta con un objeto Activator que es el que implementa la interfaz con los métodos de arranque(start) y parada(stop).

El ciclo de vida de un bundle pasa por los estados que se indican a continuación:

- *Installed*: El bundle ha sido correctamente instalado.
- *Resolved*: Todas las referencias que utiliza el bundle están disponibles. El bundle está preparado para arrancar.
- *Starting*: El bundle está arrancando, su método start ha sido invocado.
- *Active*: El bundle está activo.
- *Stopping*: El bundle se está deteniendo.

- *Uninstalled*: El bundle se ha desinstalado y su ciclo de vida ha terminado.

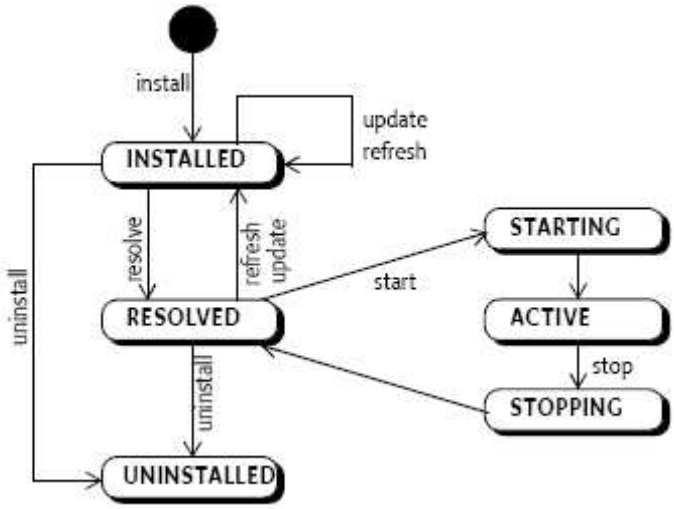

Figura 3.- Posibles estados de un bundle

A la hora de definir el ciclo de vida de un Bundle hay que tener en cuenta:

- La clase que funciona de Activator debe indicarse en el manifiesto.

- El método start debe preparar los recursos que un bundle necesita, normalmente registrar los servicios que proporciona y arrancar los threads pertinentes.

- El método stop debe deshacer todo lo que hizo el start para liberar los recursos. No hace falta desregistrar servicios porque el propio framework se encarga de ello.

1.4.1.3 Service Layer

Esta capa define un modelo colaborativo dinámico que está integrado con el ciclo de vida anterior. Es un conjunto de servicios que permite publicar, buscar y enlazar servicios. Un servicio es un objeto Java normal que se registra bajo uno o más interfaces contra el Service Registry. Los bundles pueden registrarlos, buscarlos o recibir notificaciones cuando su registro cambia.

La forma natural de operar es definir una interfaz para un servicio, implementarla en un bundle y después registrarla en el framework para que esté accesible por el resto. Una vez registrada (bien como interfaz o bien como clase concreta) el resto de bundles podrán acceder a ella buscándola en el ServiceRegistry.

Los servicios se pueden registrar y desregistrar mientras el bundle esté activo. Cuando se registra un servicio se obtiene un objeto ServiceRegistration que será como el ticket que se usará para controlar el registro.

Para cada servicio que se registra se pueden definir una serie de propiedades en pares atributo/valor, a los cuales se pueden aplicar filtros de búsqueda basados en sintaxis LDAP. Estas propiedades se pueden ampliar, pero hay unas por defecto:

objectClass - String[]

service.description - String

service.id - Long

service.pid - String

service.ranking - integer

service.vendor – String

El proceso de registro, desregistro y cambio de las propiedades es capaz de generar eventos que otros bundles pueden monitorizar para realizar acciones específicas en cada caso.

Hay que poner especial atención a las referencias de objetos que se pasan entre bundles, es conveniente saber que cuando un servicio se desregistra, todas las referencias que pertenen a su class loader deben ser liberadas o el contexto del bundle no será limpiado por el garbage collector. Para facilitar esta tarea se pueden utilizar técnicas como el interfaz ServiceFactory o el uso de indirección en los objetos de implementación.

ServiceFactory

Utilizando ServiceFactory se puede personalizar los objetos que se devuelven cuando un bundle solicita un servicio. De esta manera y si la funcionalidad es compatible, a cada bundle se le proporciona una instancia distinta del servicio. De esta manera puede saberse cuándo se deja de usar para liberarla adecuadamente.

Los bundles que solicitan servicios deben liberarlos cuando ya no los utilicen más. Si no lo hacen esas referencias puede que no sean consistentes a lo largo del tiempo, porque el

servicio ha desaparecido. Asimismo, cada bundle puede desregistrar los servicios que proporciona cuando así lo considere. Si no lo hace, el framework automáticamente los desregistrará al parar el bundle.

1.4.1.4 Package Admin Service

Los bundles pueden exportar paquetes a otros bundles. El Package Admin Service sirve para controlar el ciclo de vida de los paquetes compartidos y qué acciones tomar cuando los bundles de los que se dependen se actualizan o se desintalan.

1.4.1.5 Start Level Service

Este servicio se utiliza para determiner el orden de activación de los bundles y poder definir niveles de ejecución. Con este servicio se puede gestionar qué bundles se pueden cargar o descargar en un nivel determinado y sirve por ejemplo para definir modos de ejecución seguros, para poner pantallas de arranque, dar prioridad a ciertos bundles o controlar bundles que necesitan otros para funcionar.

1.4.1.6 URL Handlers Service

El servicio URL Handler se utiliza para registrar nuevos esquemas de URL y definir cómo convertirlos en objetos Java específicos. Este servicio puede sustituir al que tiene Java por defecto, adaptándolo a las necesidades de dinamismo de la plataforma.

1.4.2 Servicios de la plataforma

1.4.2.1 Log Service

El servicio de Log proporciona un mecanismo de propósito general para escribir mensajes de log en la plataforma. Consiste en dos servicios, uno para escribir (LogService) y otro para leer(LogReader).

Este servicios tiene la particularidad de que los eventos del Framework son registrados automáticamente. Además el resto de servicios puede registrarse a cambios en el log mediante la interfaz Log Listener.

1.4.2.2 Http Service

Este servicio permite registrar servlets (y también recursos) en la propia plataforma para que sean accedidos desde el exterior. En estos servlets se pueden implementar opciones de control remoto, interfaces remotas, o cualquier otra funcionalidad. La ventaja es que podemos acceder a OSGi mediante peticiones HTTP-GET y POST y de ahí redireccionar a los servicios internos que queramos.

Se podría considerar este servicio como una puerta de entrada HTTP a la funcionalidad de la plataforma OSGi.

1.4.2.3 Device Access specification

OSGi tiene el potencial de ser un punto de encuentro para servicios y dispositivos de muchos fabricantes y distribuidores diferentes. Las tecnologías de conexión más recientes (USB, IEEE 1394, …) permiten la inserción y extracción de dispositivos de manera dinámica en el sistema. Esto hace que sea difícil configurar todos los aspectos necesarios para cada uno de los dispositivos.

La especificación Device Access trata de definir la coordinación, detección automática y vinculación a dispositivos en OSGi. Esta especificación se centra el proceso de vinculación de los dispositivos a la plataforma, sin definir, deliberadamente, el mecanismo de descubrimiento, los interfaces de los dispositivos o las categorías de dispositivos existentes.

Esta especificación permite funcionalidades que van desde la capacidad de operación plug&play, carga dinámica de drivers, múltiples representaciones para cada dispositivo, árboles de dispositivos o incluso dispositivos con múltiples configuraciones.

Los componentes principales de este servicio son:

- DeviceManager: es el encargado de controlar todo el proceso de enlace de dispostivos y drivers. No es un servicio OSGi, sino simplemente un objeto que se debe controlar los servicios Device que se registran para asignarles un Driver apropiado.

- Driver service: un driver que es capaz de controlar cierto hardware o comunicación hard.

- Device service: un servicio que representa un dispositivo. Cada dispositivos deberá tener enlazado un driver. Es responsabilidad del DeviceManager hacer esa unión. Para conseguirlo se puede ayudar del DriverLocator y el DriverSelector

- DriverLocator service: servicio responsable de localizar y cargar los bundles de los drivers apropiados a los Device que se registran.

- DriverSelector: servicio que ayuda en la selección de un driver concreto cuando existen varias posibilidades.

Las ventajas de usar este servicio es que proporciona una arquitectura estándar para integrar dispositivos en OSGi y además separa los dispositivos de los drivers con los que se utilizan.

1.4.2.4 Configuration Admin Service Specification

Este servicio facilita las labores de configuración de las propiedades de los diferentes bundles del sistema. Consiste en una serie de interfaces que los servicios deben implementar para ser avisados cuando la configuración que tengan asociada cambie.

Hay dos alternativas, el interfaz ManagedService para servicios con un único conjunto de propiedades y el ManagedServiceFactory, utilizado cuando se generarán varias instancias de un mismo servicio y cada una necesita un conjunto propio de propiedades.

El requisito fundamental para poder utilizar el Configuration Admin Service es que los servicios tengan definido un PID único, ya que es así como se asociarán los objetos de configuración.

Cuando un bundle necesita información de configuración debe registrar uno o más objetos ManagedService con un PID como propiedad de servicio. Si tiene propiedades por defecto, éstas deben ser incluidas como propiedades del mismo. Se incluirán automáticamente la primera vez que se cree el objeto de configuración correspondiente. Al registrar un ManagedService, se invocará al método update del mismo, bien con valor nulo (si no había configuración previa) o bien con el diccionario de propiedades existentes.

Los objetos de configuración se recuperan a través del servicio ConfigurationAdmin, el cual proporcionar métodos para crear, modificar y borrar la configuración.

Otra funcionalidad a tener en cuenta es el servicio de eventos. El Configuration Admin Service es capaz de lanzar eventos cuando la configuración cambia (CM_UPDATED) o es eliminada (CM_DELETED). Los servicios que quieran escuchar estos eventos tienen que registrarse en el framework bajo el interfaz ConfigurationListener.

1.4.2.5 Metatype Service Specification

El servicio de metadatos permite a los programadores de servicios OSGi definir los atributos de forma procesable por un ordenador. El objetivo es poder especificar el tipo de información que puede ser utilizada como parámetros en forma de atributos-valor.

Esta información se representa en XML y se recomienda que esté ubicada bajo el directorio OSGI-INF/metatype.

Este servicio tiene varias aplicaciones. Por ejemplo, mediante la definición de los atributos se pueden generar dinámicamente interfaces de usuario que gestionen dichos atributos. También se puede utilizar junto con el servicio de configuración (Configuration Admin) para el mismo propósito.

El esquema de organización de los atributos sigue la estructura de LDAP, al igual que otras características del framework OSGi.

El servicio de metadatos se presenta como una manera sencilla de describir los atributos de los servicios y poder trabajar sobre ellos de forma automática. Sin embargo, tiene sus limitaciones, ya que el propio modelo de atributos es muy simple, no permitiendo tipos recursivos, tipos mixtos o tipos anidados.

1.4.2.6 Preference Service Specification

Algunas veces los bundles necesitan guardar cierta información persistente que sobreviva al ciclo de vida del mismo. Esta información puede ser bien de cada usuario, si estamos hablando de aplicaciones finales, o bien de sistema.

El servicio de preferencias proporciona un modelo más avanzado que el tradicional usado en java.util.Properties, añadiendo cierta funcionalidad al mismo:

- Soporte para jerarquías de preferencias.

- Soporte para mútiples tipos de datos (no sólo para Strings).
- Soporte para almacenamiento distribuido.
- Soporte para creación de espacios de usuario.

El objetivo de este servicio es, por lo tanto, proporcionar un mecanismo sencillo de acceso a las preferencias almacenadas en el sistema. Deliberadamente no se centra en el mecanismos de almacenamiento y las consecuencias de implementación que pueda tener (transacciones, acceso remoto, atomicidad) para primar la facilidad de uso.

Este servicio únicamente soporta el almacenamiento de datos escalares y arrays de bytes. Aunque en principio el tamaño de estos datos no es una restricción, se recomienda que no se utilice como medio de almacenamiento masivo.

Otro aspecto a considerar es que las preferencias están pensadas para que se utilicen en el ámbito del bundle que se creó, aunque se pueden pasar referencias entre bundles para el acceso a las mismas. No hay que confundir las preferencias con las opciones de configuración, para lo cual se usará el Configuration Admin Service.

1.4.2.7 Wire Admin Service

En los entornos orientados a servicios existen varias alternatives a la hora de enlazar unos servicios con otros. En los modelos más flexibles, el objetivo es que el productor y el consumidor de datos o eventos conozcan lo mínimo posible el uno del otro para que los cambios en cada uno no perjudiquen al otro.

En OSGi existen tres aproximaciones para conectar servicios. El primero de ellos consiste en permitir al consumidor que elija al productor, siguendo el tradicional patrón Observer. El segundo, más acorde con la filosofía de OSGi, consiste en hacer que el productor localice a sus consumidores mediante el llamado patrón whiteboard. El Wire Admin Service proporciona el tercero, el cual trata de ir un paso más allá a la hora de reducir el acoplamiento entre productores y consumidores.

El modelo de funcionamiento es muy sencillo a la vez que potente. Existen productores (interfaz Producer) y consumidores (interfaz Consumer) que se asocian mediante la creación de objetos Wire. Un Wire es un objeto que asocia mediante sus correspondientes PID dos

servicios, cada uno con uno de los roles indicados antes. La creación de un Wire se puede realizar sin que los servicios estén presentes, siendo el propio Wire Admin el encargado de notificar cuando cada Wire se pueda utilizar. Un Wire no es válido hasta que ambos extremos estén operativos.

Cada extremos guarda una lista de los Wire a los que está enlazado para poder trabajar con ellos. Cuando un Producer tenga datos que notificar, actualizará el correspondiente Wire. Para controlar esta actualización se permite la definición de filtros que gestionen cuándo hay que llevarla acabo.

Los datos que se envían entre el Consumer y el Producer deben ser compatibles, es decir, el tipo de datos que se genera tiene que se igual al que se recibe. Para ello, al crear un Consumer y un Producer se define una serie de *flavors*, que no es más que los tipos de datos que cada uno es capaz de procesar.

Este modelo soporta tanto mecanismo Push del Producer al Consumer como mecanismo Pull, el que el consumidor solicita los datos al Wire (nunca al Productor directamente).

Otra característica interesante es la opción de definir objetos compuestos y marcarlo en las propiedades del productor. Cuando hay multitud de datos procesables, crear un Wire para cada uno puede ser inviable. Definiendo una composición, se utiliza un Wire para multiplexar varios datos y para desambiguar se introducen en sobres (interfaz Envelope) que se identifican por un ámbito (scopes).

Además este servicio proporciona una serie de eventos, con los cuales gestionar el ciclo de vida de los Wire. Entre otros hay eventos para la creación, la conexión, la actualización, el borrado, desconexión, o lanzamiento de excepciones dentro de un Wire.

1.4.2.8 I/O Connector Service

Las comunicaciones son una parte fundamental de la plataforma OSGi, por lo que es necesario un API de comunicaciones flexible y extensible, que soporte multitud de situaciones: redes distintas, conexiones, firewalls,…

Heredado de otras experiencias parecidas en el mundo Java, como es el Connector framework de Java ME, se presenta como una alternativa perfecta para el entorno OSGi.

A nivel general, el Connector framework consiste en proporcionar interfaces de

comunicación generales cuya implementación se proporcionará en función de una URI que defina el protocolo o mecanismo deseado.

El modelo adoptado en OSGi sigue prácticamente la misma filosofía que el Connector framework. La principal diferencia es la manera de conseguir las implementaciones para realizar las conexiones, que en OSGi se obtienen a partir de un servicio llamado Connector Service en vez de a través de una clase estática como se hacía en Java ME.

Este modelo conserva sus característica de simplicidad de uso y configuración, así como el soporte para múltiples implementaciones y posibilidad de extender con nuevos esquemas de comunicación sin más que implementando un nuevo esquema.

1.4.2.9 Initial Provisioning

Una plataforma como OSGi tendrá casi con seguridad algún mecanismo que permita que un operador pueda configurarla y gestionarla. La especificación de Inicial Provisioning intenta definir cómo un agente de gestión puede acceder a la plataforma y poder así gestionarla, sin explicitar el protocolo o mecanismos concretos que se utilizan.

1.4.2.10 UPnP Device Service

Este servicio define cómo unir UPnP al entorno OSGi. El objetivo es conseguir que un servicio OSGi pueda ser exportado como un dispositivo UPnP y al revés, que un dispositivos UPnP preexistente en la red local se integre en OSGi a través de un servicio.

Si se conoce el modelo de funcionamiento de UPnP, este servicio no hace más que establecer la correspondencia y funcionamiento general en OSGi.

1.4.2.11 Declarative Service Specification

Para poder utilizar un servicio en OSGi tenemos que tener claro el modelo de funcionamiento que se basa principalmente en los mecanismos de publicación, búsqueda y enlazado de servicios.

Este modelo es muy apropiado para favorecer la comunicación y colaboración de los diferentes servicios, pero puede provocar efectos secundarios cuando el tamaño del sistema a desarrollar empieza a aumentar.

Los problemas principales de este modelo son:

- Tiempo de arranque: cada bundle tiene que registrar de forma activa sus servicios, requiriendo que todas sus referencias estén presentes de antemano. En sistemas grandes esto puede provocar retrasos de inicialización demasiado elevados.

- Tamaño en memoria: Cuando se registra un servicio, éste es cargado en memoria. Si no se utiliza es un espacio que se está malgastando, aparte de la sobrecarga que supone tener que crear el class loader correspondiente.

- Complejidad: en un entorno donde los servicios se consideran dinámicos (pueden aparecer y desaparecer en cualquier momento) el modelo de programación en sistemas complejos puede resultar difícil de mantener.

El uso de un modelo declarativo, por lo tanto, va a simplificar las tareas de desarrollo de servicios OSGi, simplificando el modelo de programación utilizado. A continuación se realiza un resumen del funcionamiento de este modelo.

El modelo se basa en el concepto de *componente*. Un componente no es más que una clase Java con ciertas características. Estos componentes tendrán una descripción en XML que definirá cómo debe gestionarse en OSGi. Con este XML muchos de los aspectos que se hacían antes a mano (registro de servicios, comprobación de dependencias, escuchador de servicios, …) los realiza el Declarative Service automáticamente.

Las diferencias principales con el mecanismo normal de creación de servicios son las siguientes:

- Se elimina el concepto de BundleActivator, el framework leerá el XML con los componentes que estén definidos y los creará en función de la política que allí esté definida. Esta política puede ser *immediate*, se crea el componente nada más resolver todas sus dependencias, *delayed*, el servicio es registrada, pero no se crea la instancia hasta que alguien lo solicita o *factory,* se registra una factoría para crear las diferentes instancias.

- Cada componente debe implementar dos métodos, uno de activación y otro de desactivación, que se invocarán durante su ciclo de vida.

- Las referencias a otros servicios se declaran en el XML con ciertas propiedades (si

son obligatorias, opcionales o la cardinalidad permitida) y es el framework el que se asegura de que estén correctamente enlazadas antes de activar el componente. La gestión de las referencias a servicios que desaparecen mientras el componente está activo también se pueden gestionar automáticamente sin necesidad de registrarse al framework para escuchar los eventos.

En definitiva, muchas de las tareas comunes a realizar con los servicios se automatizan en el XML, con lo cual los componentes resultantes son más sencillos de desarrollar a la vez que más ligeros en código y procesamiento. Por todo esto el modelo declarativo es una alternativa a considerar para el desarrollo de sistemas complejos en OSGi.

1.4.2.12 Event Admin Service

Muchos de los servicios de OSGi tienen que trabajar con eventos, bien como consumidores o como productores de los mismos. Este servicio provee de un mecanismo de comunicación entre bundles basado en los sistemas publish/subscribe de eventos.

A nivel general, este servicio permite las características típicas de los sistemas de eventos: publicación y suscripción a eventos categorizados en tipos (topics), filtrado de eventos, envío síncrono y asíncrono de los mismos,…

Como hemos dicho anteriormente, la arquitectura se basa en el patrón Publish-Subscribe. Este patrón desacopla los extremos de la comunicación mediante la creación de un canal de eventos. Los elementos que intervienen en el esquema del Event Admin son:

- *Evento* (Event):

  Define una situación que un bundle quiere notificar. Se compone de un tipo y unas propiedades.

  El tipo se utilizada para categorizar los eventos. Su nombre debe ser neutro de la implementación, o los datos asociados al mismo. Se utiliza como un primer mecanismo de filtrado para saber cómo despacharlos a los diferentes consumidores. Los tipos se organizan en un espacio de nombres jeráquico, siendo más específicos según vamos de izquierda a derecha en la jerarquía. Los nombres distinguen entre mayúsculas y minúsculas. La sintaxis recomendada es la siguiente:

fully/qualified/package/ClassName/ACTION

Las propiedades son pares atributo valor que proporcionan más información sobre el evento. El atributo debe ser de tipo String, pero el valor puede ser cualquier objecto Java, aunque se recomienda limitarlo a tipos primitivos o como mucho la clase String. En cualquier caso, los objetos que se pasan en los eventos deberían ser inmutables.

- *Event Handler*

  Este es el interfaz bajo el cual que debe registrar un servicio que quiera escuchar eventos del EventAdmin. Este servicio se configura con dos propiedades topic-scope y event-filter que indican los eventos a los que se suscribe y el filtro que hay que aplicar. Los filtros siguen la sintaxis LDAP. Un ejemplo de ambos es el siguiente:

  topic-scope → org/osgi/service/log/LogEntry/LOG_WARNING

  event-filter → (bundle.symbolicName=com.acme.*)

  Con esta configuración, por ejemplo, se recibirían los eventos LOG_WARNING de cualquier bundle cuyo symbolicName empiece por com.acme.

- *Event Publisher*

  Los publicadores de eventos no tienen ninguna característica especial. Simplemente necesitan una referencia al servicio EventAdmin y publicar un evento en la manera que consideren más apropiada.

- *EventAdmin*

  Este es el servicio que implementa el canal de eventos. La especificación contempla los dos mecanismos típicos de envío de eventos, el síncrono y el asíncrono. En el síncrono hay que tener cuidado con posibles monitores que se creen en las secuencias, porque se pueden provocar interbloqueos. En el modelo asíncrono, las implementaciones deberán considerar cómo envían y en cuántos threads, pero deben asegurar que los eventos se distribuyen en estricto orden de llegada al canal.

En resumen, este servicio proporciona un canal de eventos con opciones de filtrado y clasificado de eventos sencillo y homogéneo dentro de la plataforma OSGi.

1.4.2.13 XML Parser Service

Ante el uso tan extendido de XML en casi todas las plataformas actuales, es necesario también que OSGi provea de un mecanismo para poder acceder a los diferentes parsers de una forma que esté integrada con su filosofía.

Este servicio no define más que la manera de obtener instancias de parsers XML (tanto SAX como DOM) a través de un servicio OSGi mediante factorías. La implementación de los parses puede ser cualquiera.

1.4.2.14 Position Specification

Este servicio trata de definir una interfaz para integrar un sistema de localización GPS como servicio OSGi y que esté disponible para todos los servicios. Simplemente define los métodos que debería tener y su mecanismo de funcionamiento recomendado, el cual se basa en la utilización del WireAdmin service para utilizar el servicio.

1.4.2.15 Service Tracker Specification

El entorno de programación dinámico que proporciona OSGi obliga a los programadores a asegurarse antes de usar un servicio de que sigue operativo en el framework. Esta tarea puede convertirse en un problema, además de ser muy propensa a producir errores.

Esta especificación define una clase de utilidad, el ServiceTracker que facilita el seguimiento del ciclo de vida de los servicios que utiliza un determinado Bundle. Esta clase se autoregistra a los eventos de los servicios, facilitando así las tareas de gestión de los servicios que lo utilicen.

Simplemente es necesario crear esta clase con un determinado filtro que indique qué servicios serán vigilados por esta clase. Cuando se quiera usar el servicio se le solicita al Tracker que lo devuelva.

También podemos personalizar el proceso, bien extendiendo el ServiceTracker o bien creando un ServiceTrackerCustomizer. Con cualquiera de estos dos métodos lo que se consigue es tener control de los eventos de adición, modificación y eliminación de los servicios en el framework, pudiendo añadir lógica adicional a dichos eventos.

1.5 Implementaciones de OSGi

Siendo OSGi uno especificación abierta, cuenta con varias implementaciones, tanto Open Source como propietarias. Por las características del proyecto SMARTLAB, únicamente nos fijaremos en las implementaciones Open Source. A continuación hacemos un resumen de las implementaciones Open Source más destacadas:

Apache Felix - (incubator.apache.org/felix/) el proyecto Felix ha entrado dentro de los proyectos de incubación de la Apache Software Foundation y será la propuesta de implementación de las especificaciones de OSGi de Apache. El objetivo de Felix es implementar totalmente la Release 4 de OSGi. Actualmente incluye el framework y los servicios estándar, así como otras tecnologías interesantes de integrar, pero todavía necesita más trabajo para un soporte completo de OSGi. A pesar de estar incompleta, las partes que están implementadas son bastante estables.

Eclipse Equinox - (www.eclipse.org/equinox/) Desde el punto de vista de la implementación, Equinox es una implementación del framework de OSGi y un conjunto de bundles que implementan varios de los servicios opcionales. Está dedicado sobre todo a su utilización en Eclipse, pero también está abierto a la implementación de todos los demás aspectos de la especificación de OSGi.

Knopflerfish - (www.knopflerfish.org) El objetivo de Knoplerfish es desarrollar y distribuir un código fuente, herramientas y aplicaciones basadas en OSGi de fácil utilización. Este proyecto tiene varios desarrolladores asignados al mismo desde Gatespace Telematics, empresa que se encarga de la distribución. Existe asimismo una versión comercial llamada Knoplerfish Pro, con soporte completo para productos comerciales.

Newton Project - (www.codecauldron.org) El proyecto Newton es un framework de ejecución distribuido para la instanciación dinámica y consiguiente gestión de complejos sistemas OSGi /SCA en entornos empresariales. Basado en el sistema de descripción SCA

(Service Component Architecture), Newton es capaz de dinámicamente desplegar y mantener la disponibilidad de bundles OSGi y dinámicamente enlazarlos con otros componentes. El proyecto Newton fue donado a Open Source por Paremos (www.paremus.com).

ProSyst Open Source mBedded Server Equinox Edition - (www.prosyst.com**)** Esta implementación está basada en Eclipse Equinox pero contiene algunas características adicionales y bundles sobre ella. Esta implementación es una buena opción para proyectos Open Source y de investigación, ya que no hay ningún coste de licencia.

A continuación se hace una comparativa de las tres especificaciones OpenSource más utilizadas y se analiza qué servicios de OSGi tienen implementados. Como los servicios no son más que bundles, al final se podría completar cualquiera de las implementaciones con los bundles de otra, siempre y cuándo estos hayan seguido el mecanismos estándar de desarrollo de servicios en OSGi.

Tabla 1.- Comparativa frameworks Felix, Eclipse y Knopflerfish

	Felix Oscar	eclipse	knopflerfish
Versión estable	R4	R4	R4
Log	✓	✓	✓
Http	✓	✓	✓
Device Access	✓	✓	✗
Configuration Admin	✗	✗	✓
Metatype Service	✗	✓	✓
Preference Service	✓	✓ (extendido)	✓
Wire Admin	✓ (no completo)	✗	✗
I/O Connector	✗	✗	✓ (http,socket, datagram)
Initial Provisioning	✗	✗	✗
UPnP Device	✓ (desarrolado por terceros)		
Declarative Service	✗	✓	✓
Event Admin Service	✗	✓	✓
XML Parser Service	✓ (desarrollado por terceros)		
Position Specification	✗	✗	✗
Service Tracker	✓	✓	✓

Cada implementación incorpora además una serie de bundles extra con servicios de soporte, que en función de los requisitos es recomendable consultar.

Un recurso muy útil es el repositorio de bundles que se aloja en la propia página de OSGi (http://www2.osgi.org/Repository/HomePage). En el se pueden buscar multitud de bundles que están ya implementados y listos para utilizar.

2 REFERENCIAS

[1] OSGi Alliance – http://www.osgi.org