TABLA DE CONTENIDO

[TABLA DE CONTENIDO 1](#_Toc503814633)

[Crear Modulo OSGI – Liferay – Gogo Shell 2](#_Toc503814634)

[*OSGI (Open Services Gateway Initiative)* 2](#_Toc503814635)

[*OSGI Architecture* 3](#_Toc503814636)

[*Arrancando Liferay - Gogo Shell* 3](#_Toc503814637)

[*Creando módulo OSGI* 3](#_Toc503814638)

[*Modelos de dependencias* 3](#_Toc503814639)

[*Ciclo de vida* 4](#_Toc503814640)

[Bundle Manifest 4](#_Toc503814641)

[**System Bundle** 5](#_Toc503814642)

[Modulo OSGI - Código 6](#_Toc503814643)

[*Clase Activator* 6](#_Toc503814644)

[Compilar y publicar la clase Activator. 7](#_Toc503814645)

[Correr y probar la clase Activator. 7](#_Toc503814646)

Crear Modulo OSGI – Liferay – Gogo Shell

### *OSGI (Open Services Gateway Initiative)*

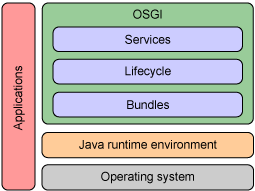
Framework creado en Marzo de 1999 **orientado a Servicios** e implementado en Java, que define la forma de crear módulos y la forma en que estos interactúan entre sí en tiempo de ejecución.

La **OSGi Alliance** es un consorcio de empresas tecnológicas a nivel mundial que trata de asegurar la interoperabilidad de las aplicaciones y servicios. Algunos ejemplos de miembros: Motorola, Nokia, Mitsubishi Electric Corporation, Vodafone Group Services, LinkedIn, LG Electronics...

Este framework provee al desarrollador de un entorno Java gestionado y seguro que permite el despliegue de aplicaciones denominadas **bundles**. El framework de OSGI permitirá la **descarga, instalación y borrado de bundles en tiempo de ejecución.**

OSGi intenta solventar los problemas del tradicional "classloader" de la máquina virtual y de los servidores de aplicaciones Java. Para ello, en OSGI **cada bundle tiene su propio classpath separado del resto de classpath de los demás módulos.**

La arquitectura OSGI se divide en capas, tal y como se muestra en la siguiente figura, las cuales se detallan a continuación:asd

[](http://www.albertcoronado.com/wp-content/uploads/2016/05/osgi2.gif)

* **Bundles**: Componentes OSGI creados por los desarrolladores.
* **Servicios**: Capa encargada de conectar distintos bundles de manera dinámica.
* **Ciclo de vida**: API que permite instalar, iniciar, parar, actualizar y desinstalar bundles sin necesidad de reiniciar el framework.
* **Módulos**: Capa que define cómo importar/exportar código fuente de un bundle.
* **Seguridad**: Capa que administra los aspectos de seguridad del framework.
* **Entorno de ejecución**: Específica qué métodos están disponibles en la plataforma.

### *OSGI Architecture*

OSGI(Open Services Gateway initiative) es la solución que aporta Java a la programación modular. Un módulo es un componente auto-contenido con una funcionalidad perfectamente delimitada dentro de un sistema mayor.

Los módulos OSGI pueden ser arrancados y parados en caliente (Sin tener que para y reiniciar un servidor), pueden comunicarse entre ellos y además pueden recibir avisos cuando otro módulo es arrancado o parado.

En el siguiente ejemplo vamos a ver como crearíamos un módulo OSGI y como interactuaría con el contenedor OSGI.

### *Arrancando Liferay - Gogo Shell*

Lo primero que necesitamos para crear nuestra aplicación OSGI es un contenedor de módulos OSGI. En este caso mediante uso del Liferay 7.0 Gogo Shell.

Esto nos abre una consola desde donde podemos manejar el contenedor: Consultar los módulos ejecutando el comando ‘lb'(List Boundles), arrancar/parar módulos con ‘start <module ID>’/’stop <module ID>’, instalar un nuevo módulo con ‘install <path al .jar>’ o ver el listado de comandos con ‘help’ entre otros.

### *Creando módulo OSGI*

Nuestro primer módulo (Bundle) OSGI, será básicamente, una aplicación que se arrancará, parará y mantendremos escuchando cuando se activen o desactiven nuevos módulos.

### *Modelos de dependencias*

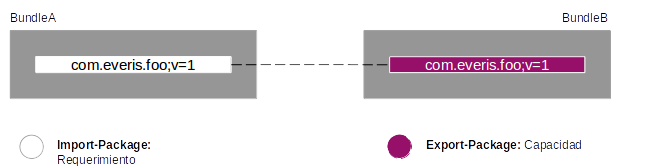
* El sistema debe resolver las dependencias del bundle, posiblemente en múltiples alternativas válidas de las cuales el sistema automáticamente **resolverá las dependencias con la mejor de las opciones posibles.**
* El modelo de resolución de dependencias dinámico de OSGI siempre encuentra el bundle que mejor encaja con cada invocación **en tiempo de ejecución.**
* OSGI recomienda la **dependencia a través de APIs.**

Un aspecto fundamental del framework de OSGI es la gestión de las dependencias entre bundles. Estas dependencias se expresan en el **MANIFEST.MF** y se pueden clasificar como **requerimientos y capacidades**.

El framework resuelve las dependencias durante la **fase de Resolución**. El framework debe resolver ser capaz de localizar una capacidad para cada requerimiento para poder declarar un bundle como **Resolved**.

Los paquetes se reflejan en el osgi.wiring.package. Un **Import-Package** se mapea en el osgi.wiring.package como un requerimiento mientras que un **Export-Package** se mapea como una capacidad.

* **Import-Package:** com.everis.foo; version=1
* **Export-Package:** com.everis.foo; version=1

[](https://github.com/maldiny/OSGI-en-Castellano/blob/master/Imagenes/Osgi-wiring.png)

### *Ciclo de vida*

### Bundle Manifest

* **Bundle-Activator:** Especifica el nombre de la clase que será invocada durante el arranque y la parada del bundle a los métodos respectivamente. Esta clase deberá implementar los métodos de BundleActivator start() / stop(). p.e.j: com.acme.fw.Activator
* **Bundle-Category:** Listado de categorías separados por coma al que pertenece el bundle. p.e.j.: osgi, test, nursery
* **Bundle-ClassPath:** Listado de jar separados por coma que serán cargados en el classpath del bundle durante su arranque. p.e.j: /jar/http.jar,.
* **Bundle-ManifestVersion:** Define el listado de reglas de especificación del bundle. p.e.j.: 2
* **Bundle-Name:** Define el nombre del bundle. p.e.j: Firewall
* **Bundle-SymbolicName:** Especifica el nombre identificativo único para el contenedor de bundles. p.e.j: com.acme.daffy
* **Bundle-Version:** Número de la versión del bundle. p.e.j: 1.1
* **Export-Package:** Listado de paquetes exportados por el bundle especificando la versión de forma opcional pero recomendable. p.e.j: org.osgi.util.tracker;version=1.3
* **Import-Package:** Listado de paquetes importados por el bundle necesarios para la ejecución del mismo. p.e.j: org.osgi.util.tracker,org.osgi.service.io;version=1.4

**System Bundle**

* El system bundle es el bundle que representa el **framework** del contenedor.
* Siempre se le asignará como el bundle con **identificador cero (0).**
* Su ciclo de vida no es como el de resto de bundles.
  + **start**: No realiza nada porque el System bundle ya se encuentra en funcionamiento.
  + **stop**: Realiza la parada de todo el framework.
  + **update**: Para y reinicia el framework en otro hilo.
  + **uninstall**: Esta operación no se puede realizar. Retorna un BundleException.
* Provee un **servicio de tipo “Package Admin”** que provee acceso a las librerías internas del sistema.
* El **StartLevel** de este bundle es el **0** y no puede ser modificado.

Modulo OSGI - Código

### *Clase Activator*

Clase que implementa la interface BundleActivator, que sólo tiene dos métodos, start() y stop(), a los que se llamará cuando el framework arranque o pare nuestro bundle.

Los métodos reciben como parámetro un BundleContext, con el que nuestro Activator podría interactuar con el framework OSGI y a través de él con otros bundles

**package** com.wlopera.osgi;

**import** org.osgi.framework.BundleActivator;

**import** org.osgi.framework.BundleContext;

**import** org.osgi.framework.ServiceEvent;

**import** org.osgi.framework.ServiceListener;

**public** **class** Activator **implements** BundleActivator, ServiceListener {

@Override

**public** **void** start(BundleContext context) **throws** Exception {

System.***out***.println("Iniciar servicio OSGI!!");

context.addServiceListener(**this**);

}

@Override

**public** **void** stop(BundleContext context) **throws** Exception {

System.***out***.println("Finalizar servicio OSGI!!");

context.removeServiceListener(**this**);

}

/\*\*

\* Implementa ServiceListener.serviceChanged().

\* Imprime los detalles del servicio Framework

\* **@param** evento

\*\*/

**public** **void** serviceChanged(ServiceEvent event)

{

String[] objectClass = (String[])

event.getServiceReference().getProperty("objectClass");

**if** (event.getType() == ServiceEvent.***REGISTERED***)

{

System.***out***.println(

"Tipo de servicio: " + objectClass[0] + " registered.");

}

**else** **if** (event.getType() == ServiceEvent.***UNREGISTERING***)

{

System.***out***.println(

"Tipo de servicio" + objectClass[0] + " unregistered.");

}

**else** **if** (event.getType() == ServiceEvent.***MODIFIED***)

{

System.***out***.println(

"Tipo de servicio " + objectClass[0] + " modified.");

}

}

}

Notas: Implementa ServiceListener: Permite monitorear el ciclo de vida del bundle.

### Compilar y publicar la clase Activator.

### 

### Correr y probar la clase Activator.

### 

### 