

# **OPEN SERVICE GATEWAY INITIATIVE USING JAVA EMBEDDED SERVER**

## **(JES 2.0)**

### **Abstract**

OSGi defines a model for service platforms where applications written in Java can be installed and executed. The applications typically consist of a number of components - "bundles" - that together solve a problem. These bundles are provided by different vendors and must be configured to cooperate on a service platform (Framework). A large number of service platforms can be remotely administrated by an operator responsible for their continuous operation. Existing services can be upgraded and services can be installed without physical access to the platform. Great emphasis is placed on securing the platform against unauthorized access. The platform must be capable of running autonomously without requiring user interaction.

### **What is OSGI?**

It is java technology based solution specified by **Open Service Gateway Initiative**. The main function of this framework is to delivery of multiple services over wide-area networks to local networks and devices.

### **OSGi Specification:**

- OSGi Specification 1.0, May 2000.
- OSGi Specification 2.0, October 2001.

### **OSGi Implementations:**

- JES2.0, Sun Microsystems
- E-Box, Ericsson
- SMF3.0, IBM
- OSCAR, R. Hall
- JSPservlet, PageBox
- mBserver5.1, ProSyst
- OpenRG, Jungo

### JES2.0 from Sun Microsystems:

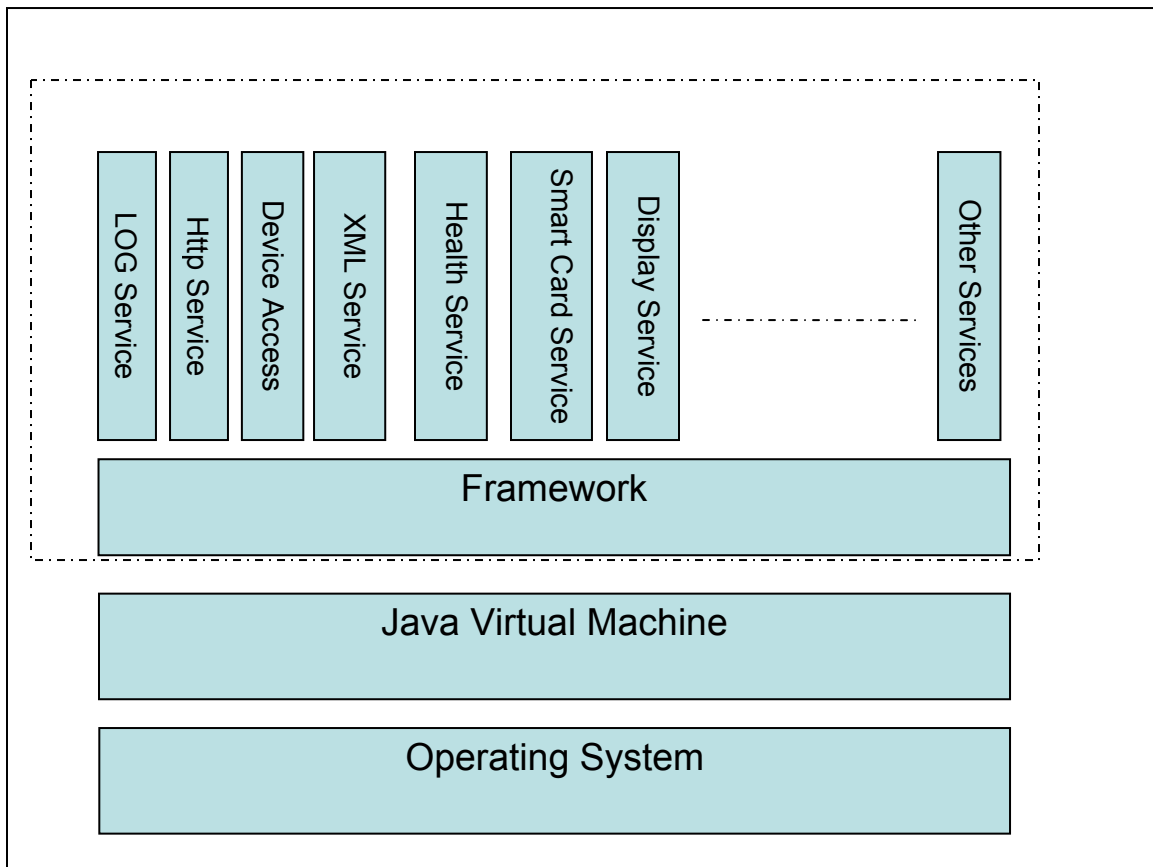
Sun Microsystems is a founding member of the OSGi consortium ([www.osgi.org](http://www.osgi.org)). They have defined an open standard for connecting future generations of networked consumer and small business devices to Internet services.

Advantages:

- Platform Independence
- Vendor independence
- Future-Proof
- Integration

Operating model:( fig 1 at the end)

Architecture:



The above figure shows the residential gateway architecture. It usually consists of several layers. The foundation of the execution environment is made by the operating system and java platform. The OSGi software runs top of the Java runtime environment.

It consists of

**Framework:** it acts as the underpinning common environment for hosting a set of bundles.

- Managing the life cycle of bundles and resolving interdependencies among bundles and
- Making classes and resources available from a bundle
- Maintaining a registry of services
- Firing events and notifying listeners when a bundle's state changes, when a service is registered or unregistered, or when the framework is launched or raises an error

**Bundles:** these are the software components plugged into the framework. A bundle may provide zero, one, or multiple services.

**Service:** a service does something useful and is implemented in the Java programming language. Examples for service are Web server, calculating taxes, application that reports electricity consumption and so on.

To develop a service, you usually define an interface that says what the service does, along with corresponding implementation classes that spell out how the service is to be performed.

**Example:** following is the service that plays digital music

**Service Interface:**

Package player.service;

//imports

public interface DigitalPlayer {

// Plays music read from the stream

public void play(InputStream musicStream);

}

Class that implements the above interface may look like:

```
Package player.impl.mp3;
Import player.service.DigitalPlayer;
//other imports
public class MP3Player implements DigitalPlayer{
public void play(InputStream musicStream){
//read MP3 encoded bytes from stream,
//send to the appropriate codec
// available in this service implementation.
}
}
```

The separation of interface and implementation ensures the service interface to the callers remains stable while the implementation undergoes changes.

### **Bundle:**

A bundle is a form of packaging for the services. It is also a functional unit with life cycle operations and class loading capability.

- A bundle is a JAR file that contains class files and resources such as images, HTML pages, and other data files
  - Bundle = interface + implementation +resources
  - Interface: what the service does
  - Implementation: how the service is to perform
- Services are delivered as bundles

### **Bundle Life cycle (fig 2 at the end)**

#### **The “Hooks” to the framework:**

Because bundles are to be installed into the framework, standard interfacing mechanisms are defined within each bundle from which the framework learns how to host the bundle.

They are the manifest and the activator of the bundle.

1. Manifest:

The manifest file is the standard entry in a JAR file. An additional set of manifest headers have been defined in the OSGi Service Gateway Specification from which the framework can gain knowledge about the bundle to host it successfully.

Example manifest headers:

Bundle-Activator: player.impl.Activator

Export-Package: player.service

Import-Package: http.service

## 2. The Bundle Activator:

The activator is implemented by a bundle to perform customized operations at the time when the bundle is started and stopped. It implements the start and stop methods of the `org.osgi.framework.BundleActivator` interface,

```
Public interface BundleActivator {  
Public void start (BundleContext context);  
Public void stop (BundleContext context);  
}
```

### **Summary:**

- OSGi is about delivering services over WAN to LAN
- A bundle is a packaging unit
- A bundle is a functional module
- A bundle can be installed, started, updated, stopped, and uninstalled.

## “HOME, SWEET HOME” BUNDLE

### 1 Create the Directory

C:\home

### 2 Use Notepad to create a file named Manifest that contains only one line and one blank line:

Bundle-Activator: home.SweetHome

Save this file as “*Manifest*” (with quotations)

### 3 Write an Activator

Open text pad

Now create the java file as shown:

```
package home;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class SweetHome implements BundleActivator
{
    public void start(BundleContext ctx)
    {
        System.out.println(“Home, Sweet Home”);
    }
    public void stop(BundleContext ctx)
    {
        System.out.println(“I’ll be back”);
    }
}
```

Save the file as SweetHome.java in c:\home

### 4 Compile the Activator Class

From the start button -> **run**

Type *cmd* and enter.

**type** set CLASSPATH = c:\jes2.0\lib\framework.jar

### 5 compilation ( go to dos prompt )

c:\> javac home/sweetHome.java

### 6 create jar file

c:\> jar cmf home/Manifest home.jar home/\*.class

### 7 Inspect the content of home.jar

c:\> jar tf home.jar

## **8 Running the Bundle**

Go to start, programs, Java Embedded Server 2.0

You will come to the prompt

> Type 'help' to see the list of the commands

## **9 Install the bundle**

➤ install c:\home\home.jar

## **10 Examine**

➤ bundles

## **11 Start Bundle**

➤ start 1

## **12 Type bundles**

➤ bundles

## **13 Stop bundle**

➤ stop 1

## **14 Uninstall the bundle**

➤ uninstall 1

➤ bundles

➤ shutdown.

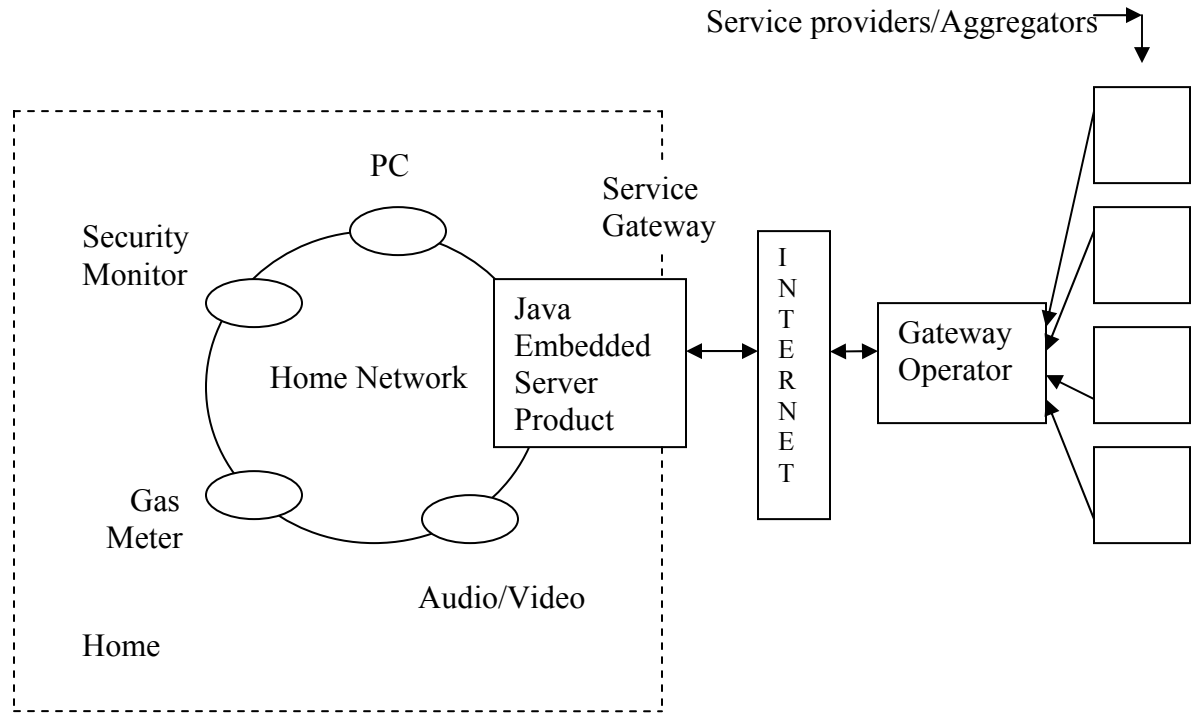


Fig 1: Operational Model

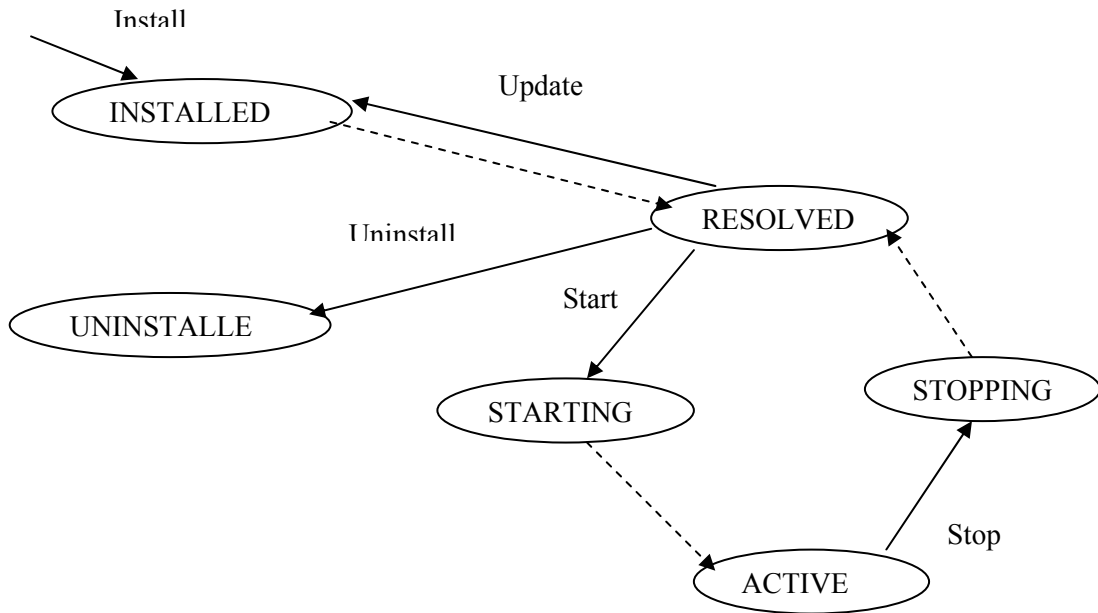


Fig 2