OSGI Development

Introduction

Henry R.P

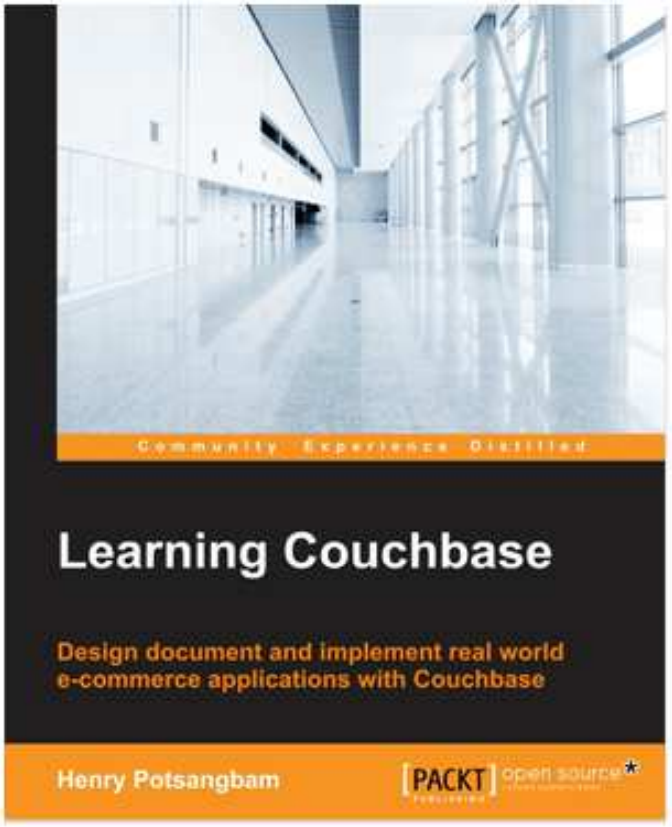
IT Architect / Consultant & Corporate trainer

- **TOGAF – Certified**
- **Mapr Certified Hadoop Cluster Administrator**
- **IBM Certified Application Developer**
- **IBM Certified Solution Designer**
- **SAP Certified Development Consultant .**
- **CIPM – Certificate in Project Management.**

HP

Training & Consulting on:

- **OSGI** - Eclipse IDE/ Equinox/ Virgo/ Spring DM / Felix / Karaf
- **NOSQL & Bigdata** – Couchbase, Hadoop, Cassandra, MongoDB,CDH, BigInsight
- **Predictive Analytics – R & SAS**
- **EAI**:- Mule / Fuse ESB /Spring Integration/ JBI / Apache Camel /Talend/ Apache Service Mix
- **Portal**:- Liferay, SAP Netweaver.
- **Application server**:- WAS, Tomcat , WebLogic, Jboss
- **Architecture**: EA, TOGAF, CoBIT etc.
- **JEE Framework**



Author

henry@thinkopensource.in

HP

Clientele

HP

Introduce Yourself.

- Name
- Year of Experience.
- Skills Level
  - Java / Maven /Web Services
  - OSGI
- Expectation, if any.

Schedule

Time	
10.00 – 11.15 AM	Session I
11.15 AM to 11.30 AM	Tea Break
11.30 AM to 1.00 PM	Session II
1.00 PM to 2.00 PM	Lunch Break
2.00 PM to 3.15 PM	Session III
3.15 PM to 3.30 PM	Tea Break
3.30 PM to 6.00 PM	Session IV

HP

Outline

Introduction

- OSGI
- Modular system
- Others OSGI Implementation

OSGi Bundle Overview

- Modules
- The OSGi Service Platform
- OSGi Bundles
- Activators
- The Eclipse Plug-in Development Environment (PDE)
- Eclipse Run Configurations

OSGI Implementation

- Equinox Overview
- Felix Overview
- Karaf

-OSGI - Additional Topics

Development IDE + Maven

Apache karaf

- Introduction to Apache Karaf
- Karaf Consoles
- Application Logging
- Provisioning Applications
- Deploying Applications

OSGI Layers:

- Bundle Lifecycle

  Service Layer
- Native Code Libraries

Bundle Advance:

- Background: Classes and Class Loaders
- Typical Class Loader Delegation
- Bundle Class Loading
- Bundle Dependencies
- A New Level of Visibility

o Require-Bundle vs. Import-Package

o Bundle Version Numbering

HP

Outline

Service Layers – SCR

- Service Component
- Declarative Services & Extension Points
- Dynamic Services
- Dynamic Services Big Picture
- Service Providers
- Service Consumers
- Using Service Trackers

HTTP Service

- Web container embedded in Equinox
- Embed Equinox inside Web Container
- Advantages and disadvantages of both these architectures

REST

- Introduction Rest services based on Jersey. REST API
- Introduction to Apache Shiro :

▪ Authentication and Authorization in Restful Services

Testing OSGi based Applications

▪ OSGi Mocks

▪ Pax Exam 2.4

▪ Troubles shooiting OSGI application with Karaf

▪ Best Practices:
OSGI Best Practices

▪ Designing web services APIs' for CRUD operations in REST resources.(URI pattern , bundle structure etc).

Introduction
Why Modular?
OSGI – Java Modular System.
Introduction To OSGI

Modular

- No doubt, a computer is complex system.
- How do you handle this complexity, when designing large systems?

Modular Systems

- Break the large system into more smaller, understandable units
- These small units are called modules.

- Benefits of modular systems
  - Reuse
  - Abstraction
  - Devision of labour
  - Ease of repair

Modular Systems :- I am a hard disk

- I've been made up of many smaller parts. I need all of them to work properly. We work as a single unit. (self contained)
- You can store important information inside me and you can retrieve them later. Thats what I do. I am not doing unrelated things.(highly cohesive)
- Talk to me using our language (common interface shared between other hard disks).
- I don't care about how other modules perform their work internally. I talk to their interface(loosecoupling)

Modular Systems

In the software world

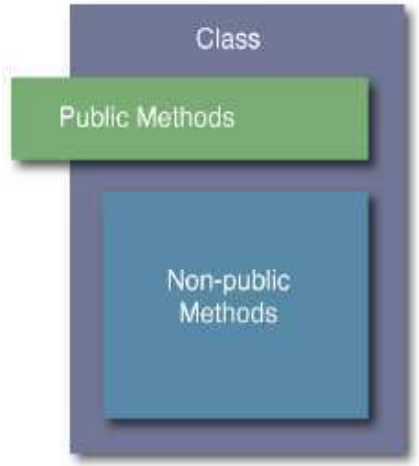
- Same theories can be applied to the software world also.

- Dividing a complex software system into small parts/modules.
  - Allows us to understand the system easily.
  - Allows us to develop part by part by different team.
  - Allows us to reuse already developed modules.

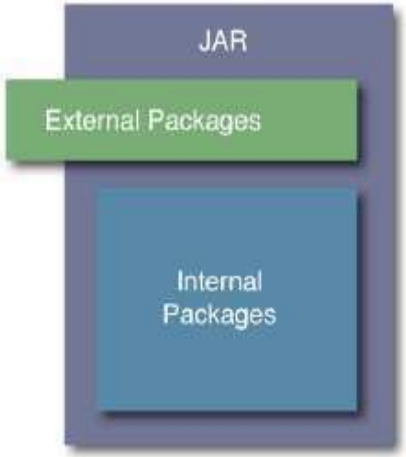
- Does Java supports building true modular systems?

Java for Modular Systems

- Java is one of the popular OOP languages for developing large enterprise applications.
- Java provides some level of modularity.
- Consider a Java class
  - The unit of information hiding in Java.
  - Public methods, expose a defined contract
- Yet Java alone fails to develop better modular systems. Why?



Java for Modular Systems

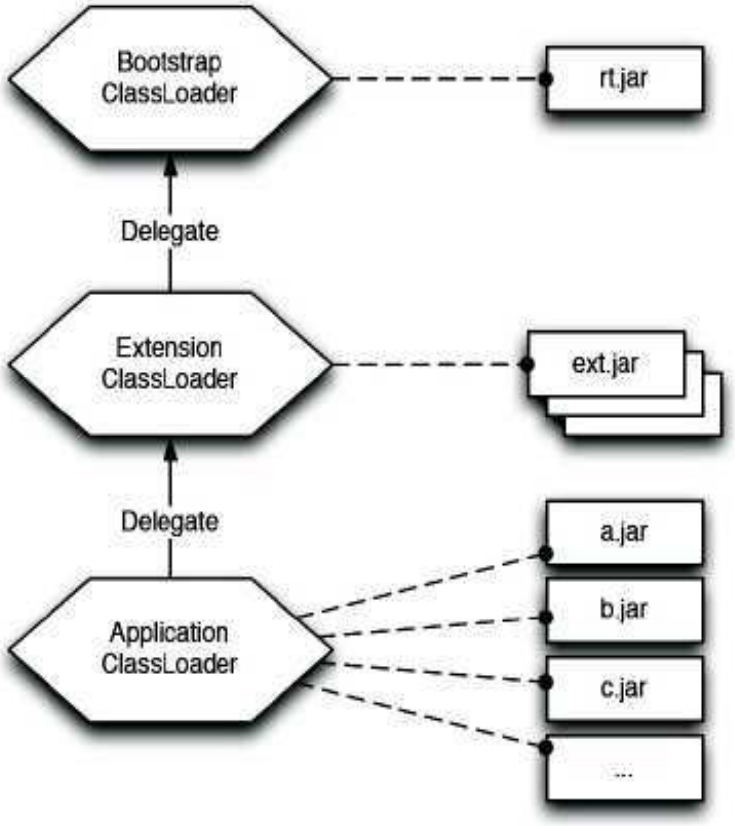


- What we need is something like this.
- A package should be the information hiding unit.
- It should be possible to,
  - Share a subset of packages from a Jar
  - Hide a subset of packages from a Jar

java classpath a.jar:b.jar:target/classes org.sample.HelloWorld

Class Loading

In standard **Java application**



Problem with JARs

- JAR is unit of deployment in Java.
- Typical Java application consists a set of JAR files.
- No runtime representation for a JAR.
- At runtime contents of all JAR files are treated as a single, ordered and global list which is called the class path
- Consider the following command.

```
java classpath
log4j.jar:statxapi.jar:woodstox.jar:axis2.jar:carbon.jar:utils.jar:target/classes
org.sample.HelloWorld
```

Problem with JARs

Problem with JARs

Problematic scenario

- HelloWorld class has a dependency on log4j version 2.0.
- What version of the Appender class is loaded?

Problem with JARs

- Multiple versions of JAR files cannot be loaded simultaneously
- A JAR cannot declare dependencies on other JARs.
- No mechanism for information hiding
- Hence, JARs cannot be considered as modules

Java for Modular Systems

- Can you update a part(can be a JAR file) of a running Java application?
- Can you add new functionality to a new Java application at runtime?
- The answer is NO.
- If you need to add new functionality or update existing functionality, JVM needed to be restarted.
- Java lacks dynamism

Java alone cannot be used to build true Modular Systems..

But Java has given a great flexibility which has allowed to build a powerful module system on top of it.

That is..

OSGi

The Dynamic Module **System for Java**

The OSGi technology

The Open Service Gateway initiative (OSGi) was formed in March 1999 by a consortium of leading technology companies.

Mission:- to define a *universal integration platform for the interoperability of applications and services*.

- Defines a way to create true modules and a way for those modules to interact at Runtime

- Modules(Bundles) in OSGi can be installed, updated and uninstalled without restarting the JVM.

A Case study.

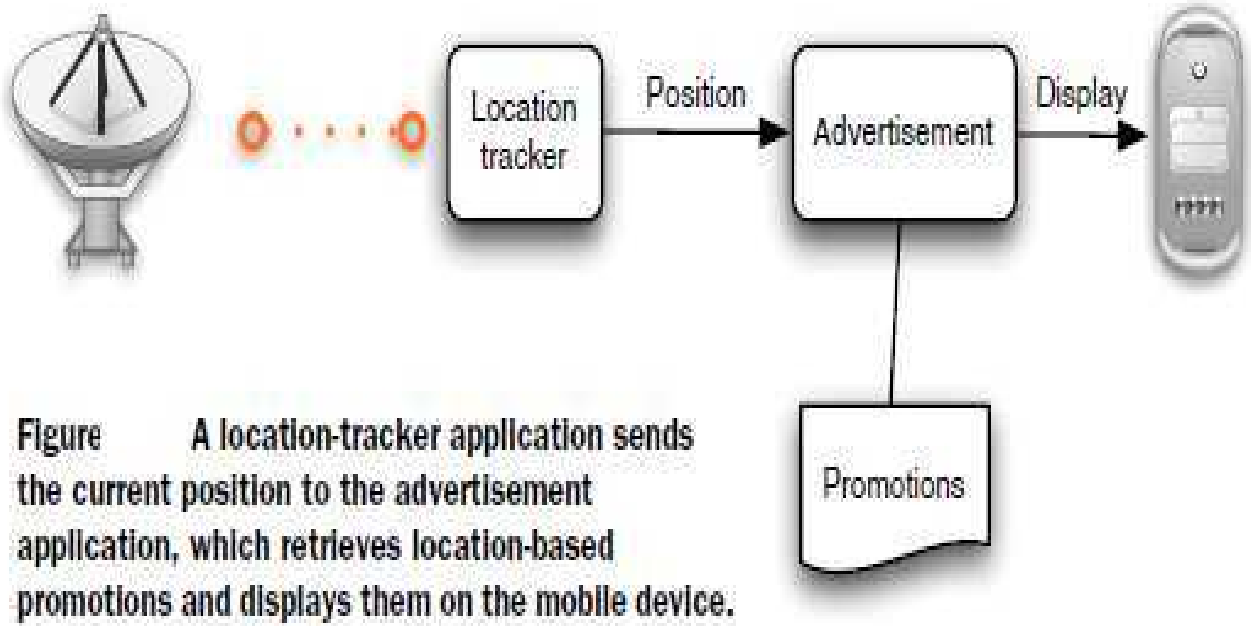


Figure A location-tracker application sends the current position to the advertisement application, which retrieves location-based promotions and displays them on the mobile device.

Issues with conventional Programming Model:

- ✓ COPING WITH DIVERSE PROGRAMMING APIS
- ✓ VARYING DEVICE CAPABILITY
- ✓ SUPPORTING DYNAMIC CHANGES
- ✓ PROVIDING A LIGHTWEIGHT SYSTEM

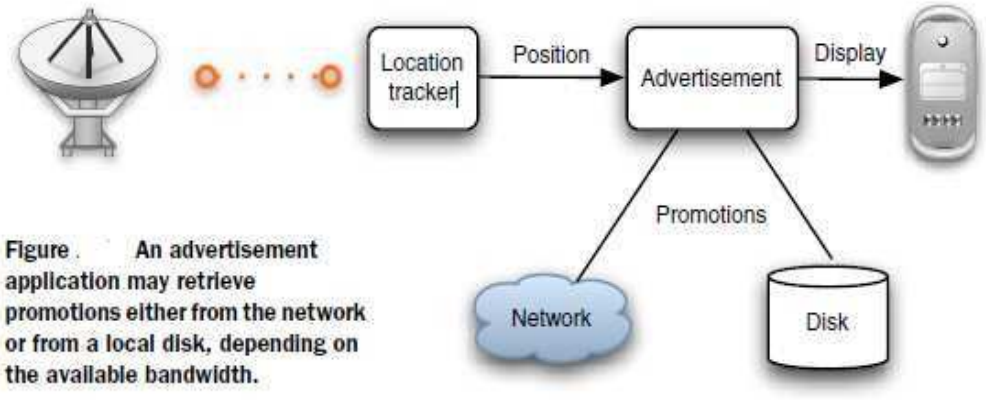


Figure . An advertisement application may retrieve promotions either from the network or from a local disk, depending on the available bandwidth.

The solution: a dynamic module system for Java. – OSGI

The OSGi Service Platform is a dynamic module system for Java.

In OSGi terminology,a Java module is called a *bundle*.

The OSGi Service Platform is composed of two main components, the OSGi framework and the OSGi services.

THE OSGI FRAMEWORK

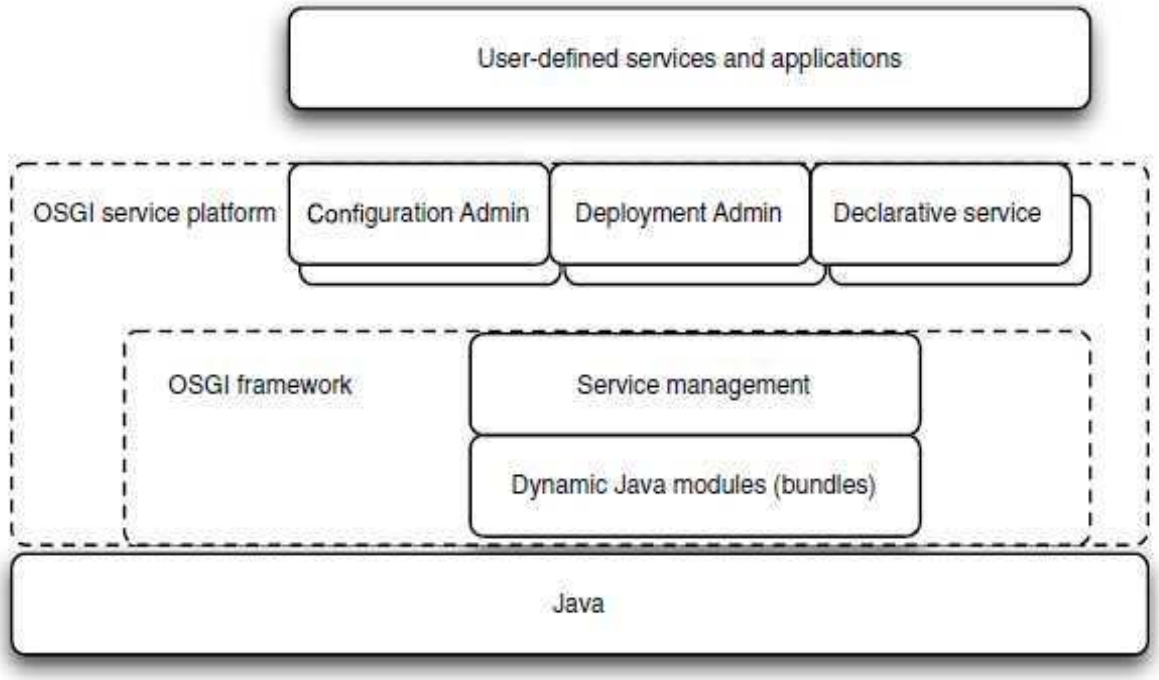


Figure **The OSGI Service Platform comprises the OSGI framework and the OSGI services.**

The OSGi framework provides:

- A portable and secure execution environment based on Java
- A service management system, which can be used to register and share services across bundles and decouple service providers from service consumers
- A dynamic module system, which can be used to dynamically install and uninstall Java modules, which OSGi calls bundles
- A lightweight and scalable solution

THE OSGI SERVICES

Alongside the OSGI framework, the OSGI Service Platform includes several general purpose Services.

- Logging service
- Configuration service
- HTTP service
- System services:
  - ✓ the bundle wiring:
    manages the dynamic module system itself
  - ✓ the start-level service:
    manages the bootstrap process of the framework.

The Enterprise OSGi:- enterprise

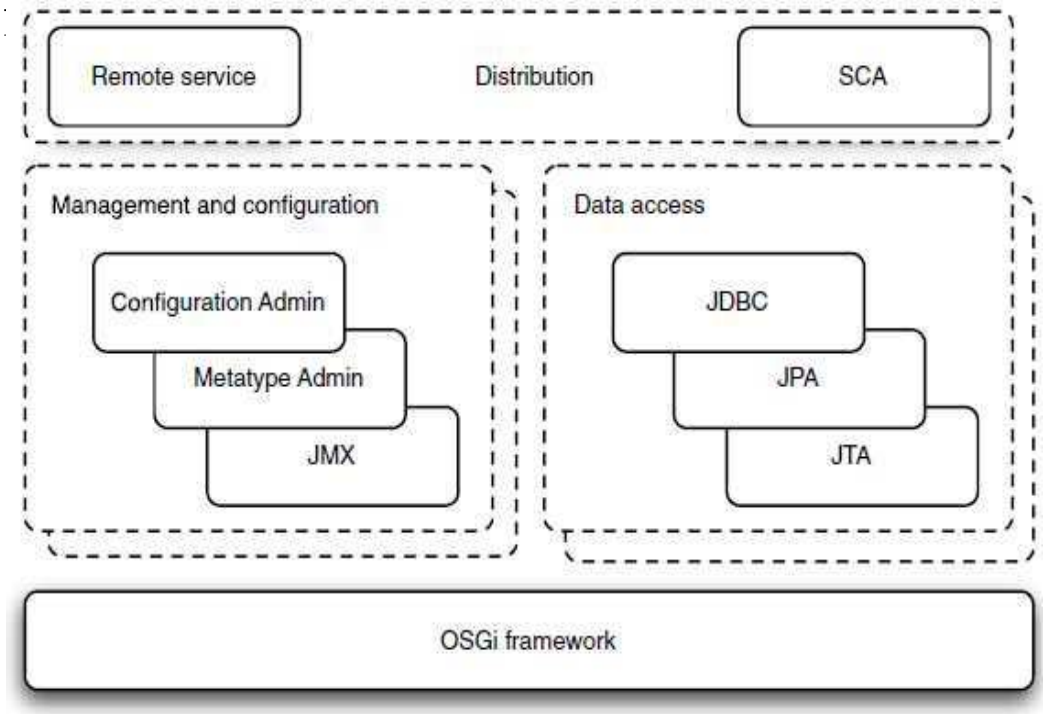


Figure Enterprise OSGi consists of the OSGi framework and several OSGi services; together they provide enterprise features.

Benefits of using the OSGi platform

OSGi manages the complexity of large systems :

OSGi decreases complexity by allowing you to efficiently modularize your code and thus deal with smaller problems one at a time.

OSGi provides extensibility without eroding the system

A service consists of an interface and an implementation. The service consumer only sees and uses the service interfaces, whereas the service provider supplies the service implementations and doesn't interact directly with the consumers.

OSGi is lightweight and customizable

The OSGi platform can be customized to be as lightweight or as complex as needed.

OSGi allows for portability

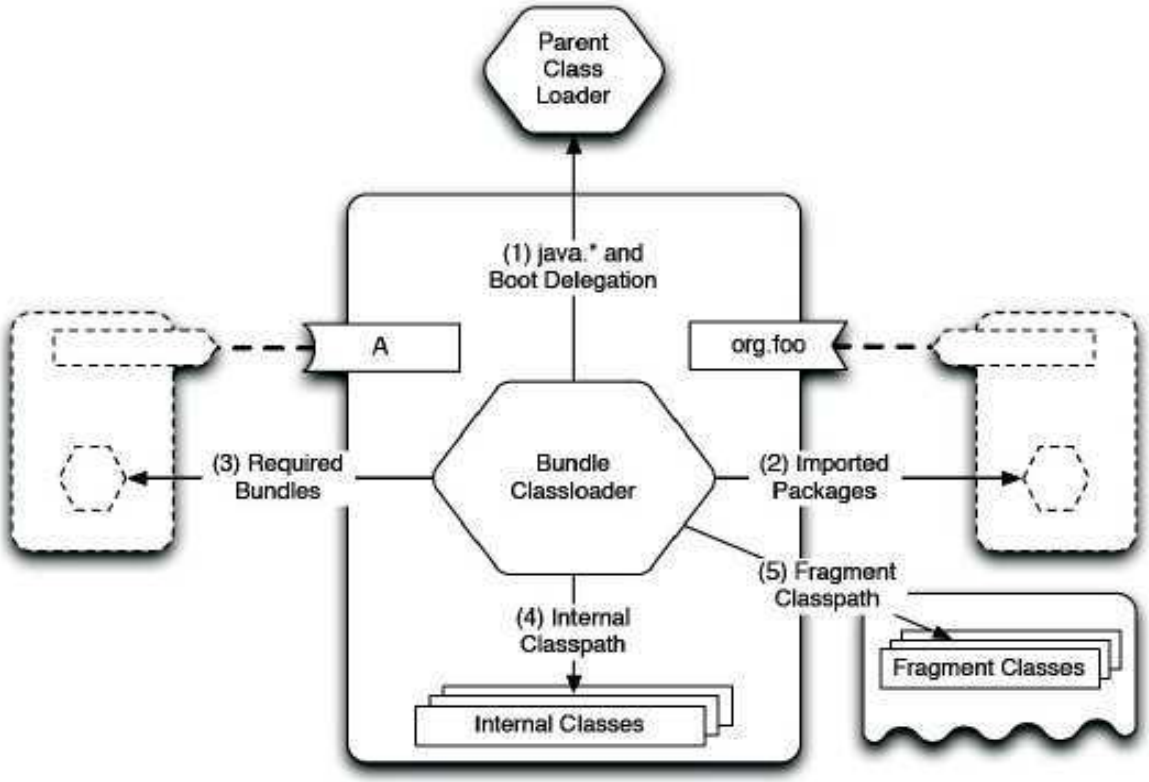
Implementations of OSGi frameworks

- Eclipse Equinox
- Apache Felix
- Knopflerfish

Examples of development platforms that are OSGi based:

- IBM WebSphere Application Server
- Oracle (formerly Sun) GlassFish Application Server
- Eclipse Virgo (SpringSource dm Server)
- JBoss Application Server
- Apache Camel
- Apache Sling
- Apache ServiceMix
- Apache Karaf

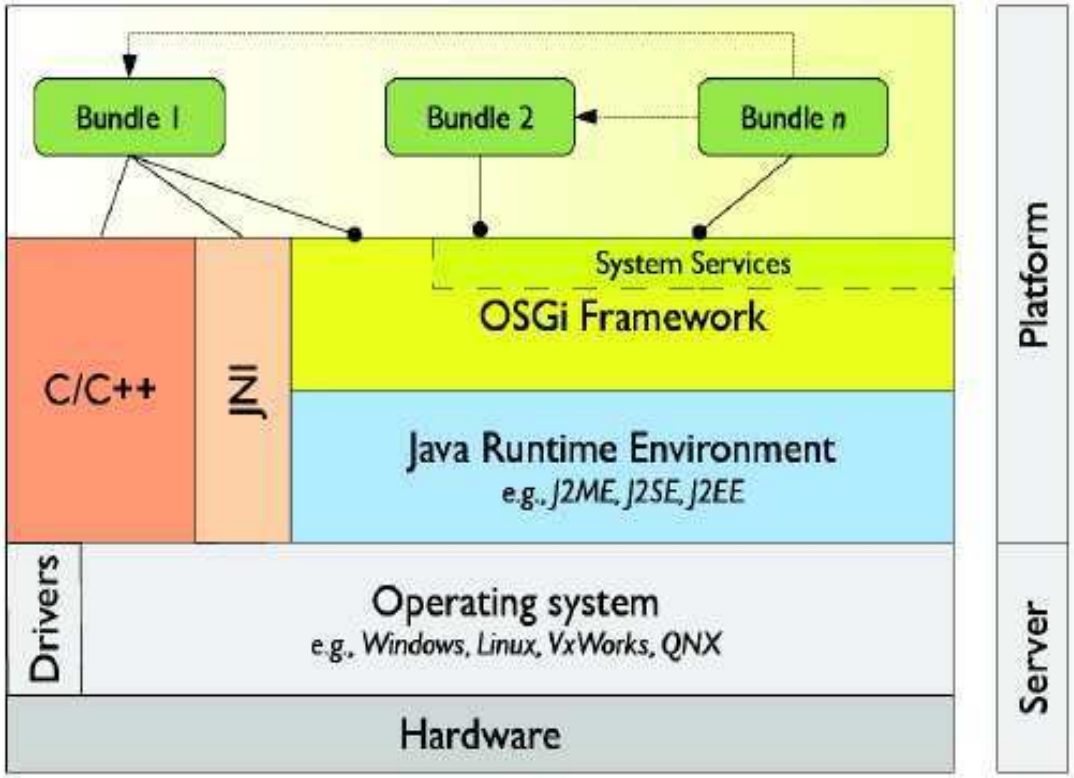
OSGi's class loading



OSGi Specifications

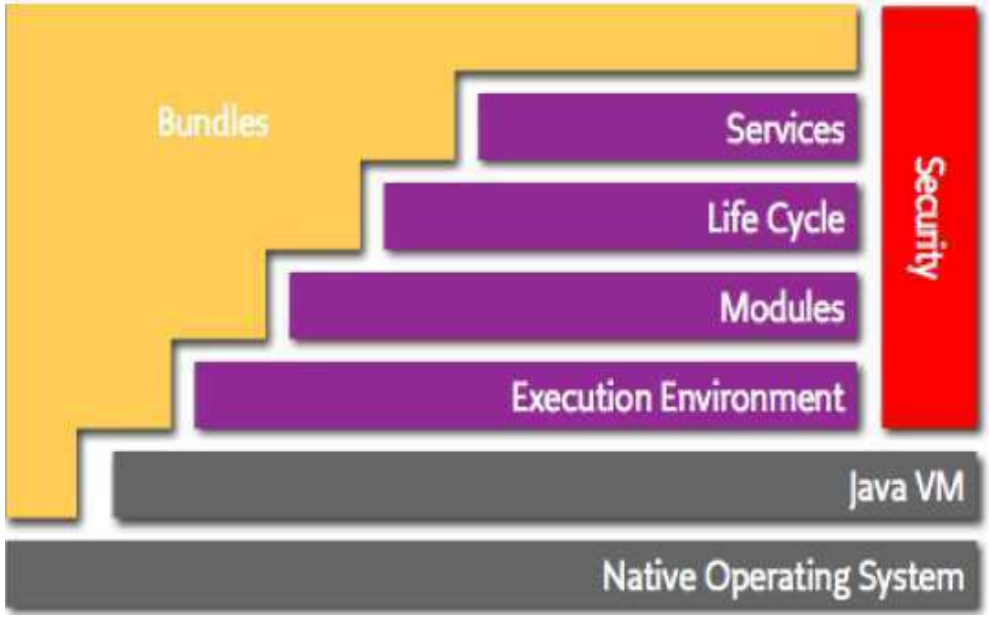
- Core specification
  - specifies the framework and the system services
- Service Compendium
  - specifies several OSGi services which are relevant for different markets such as vehicle and mobile.
- OSGi Alliance, a non profit organization.
  - develop OSGi specifications.
  - Latest released version of the OSGi platform is 4.1

Java & OSGi


Bundle 1
Bundle 2
Bundle n
System Services
OSGi Framework
C/C++
JNI
Java Runtime Environment
e.g., J2ME, J2SE, J2EE
Drivers
Operating system
e.g., Windows, Linux, VxWorks, QNX
Hardware
Platform
Server

Layering

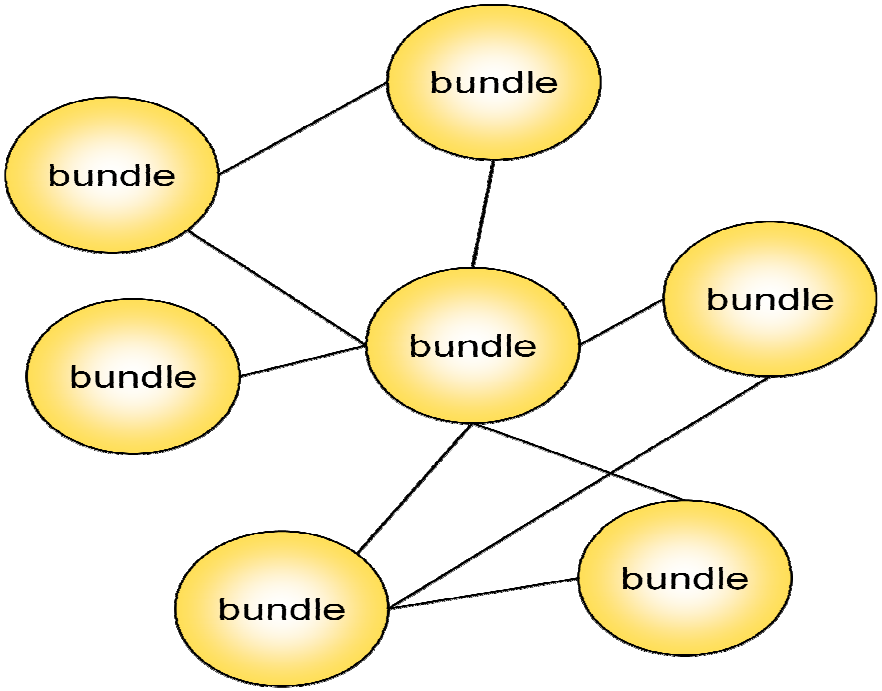
Functionality of the framework is divided up into several layers



Module Layer

- Packaging of applications and libraries in *Bundles*
  - Raw Java has significant deployment issues
- Class Loading modularization
  - Raw Java provides the Class Path as an ordered search list, which makes it hard to control multiple applications
- Protection
  - Raw Java cannot protect certain packages and classes
- Versioning
  - Raw Java cannot handle multiple versions of the same package

Module Layer



Modules and information hiding - Bundle

The unit of modularization in OSGi is called a bundle.

Bundles allow us to enforce the principles of information hiding.

A bundle is easily defined in OSGi by adhering to the following two simple steps:

- Package the module as a JAR file.
- Include a manifest file with the mandatory manifest header `Bundle-Symbolic-Name`. This header provides an identifier for the bundle.

Here's a sample manifest file: (META-INF/MANIFEST.MF)

```
Manifest-Version: 1.0
Bundle-SymbolicName: helloworld
```

Bundle (continue….)

Bundle (continue….)

A bundle encapsulates a collection of Java classes that are highly cohesive.

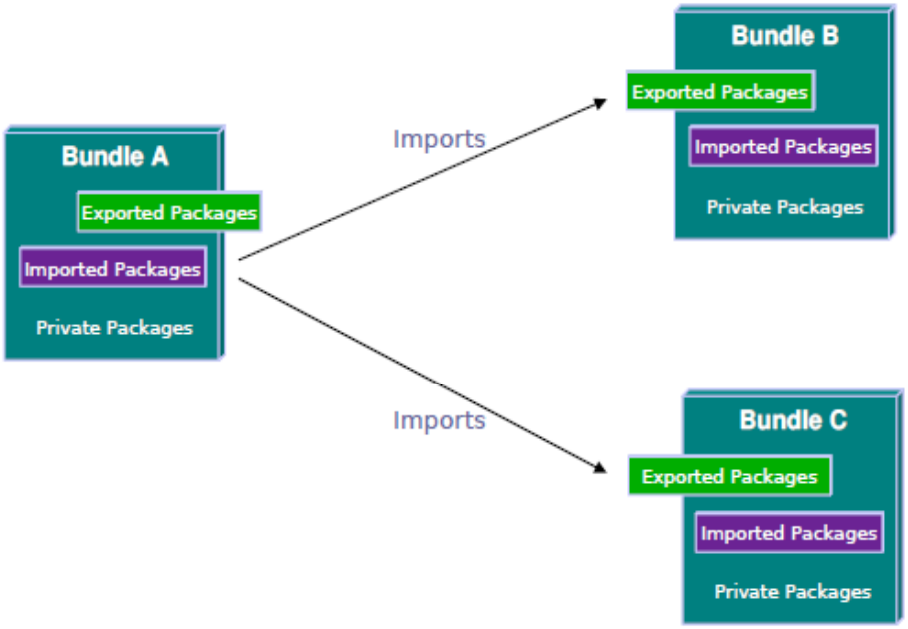
Bundles & Java packages

- By default packages in a Bundle are considered as private. Other bundles cannot see them.

- If a bundle needs to share packages, it needs to explicitly export packages using the manifest header Export-Package.

- The way to use classes/packages in other bundles is to import them explicitly using Import-Package manifest header.

Bundles & Java packages



How does OSGi achieve this level of information hiding...?

The System Bundle

- Represents the framework.
- OSGi Core framework implementation classes reside in the system bundle.
- Registers system services.
- Exports packages that are loaded from the system classpath.

Activating a bundle - Activator

In OSGi, an application is made of potentially several bundles, so you need a way of individually starting the bundles.

You can start a bundle by including a `Bundle-Activator` class in the bundle's JAR file. You have to follow these steps:

1) Create a class that implements the interface `org.osgi.framework.BundleActivator`.

2) Provide an empty constructor that can be instantiated with `Class.newInstance()`.

3)Add the manifest header `Bundle-Activator` to the bundle's manifest file.

Example:

```
package manning.osgi.helloworld.client;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;


public class PrinterClientActivator implements BundleActivator {


public void start(BundleContext context)
        throws Exception {
        new PrinterClient().printMyMessage();
    }
    public void stop(BundleContext context)  throws Exception {
    }

}

MANIFEST.mf
Manifest-Version: 1.0
Bundle-SymbolicName: helloworld.client
Import-Package: manning.osgi.helloworld
Bundle-Activator: manning.osgi.helloworld.client.PrinterClientActivator
```

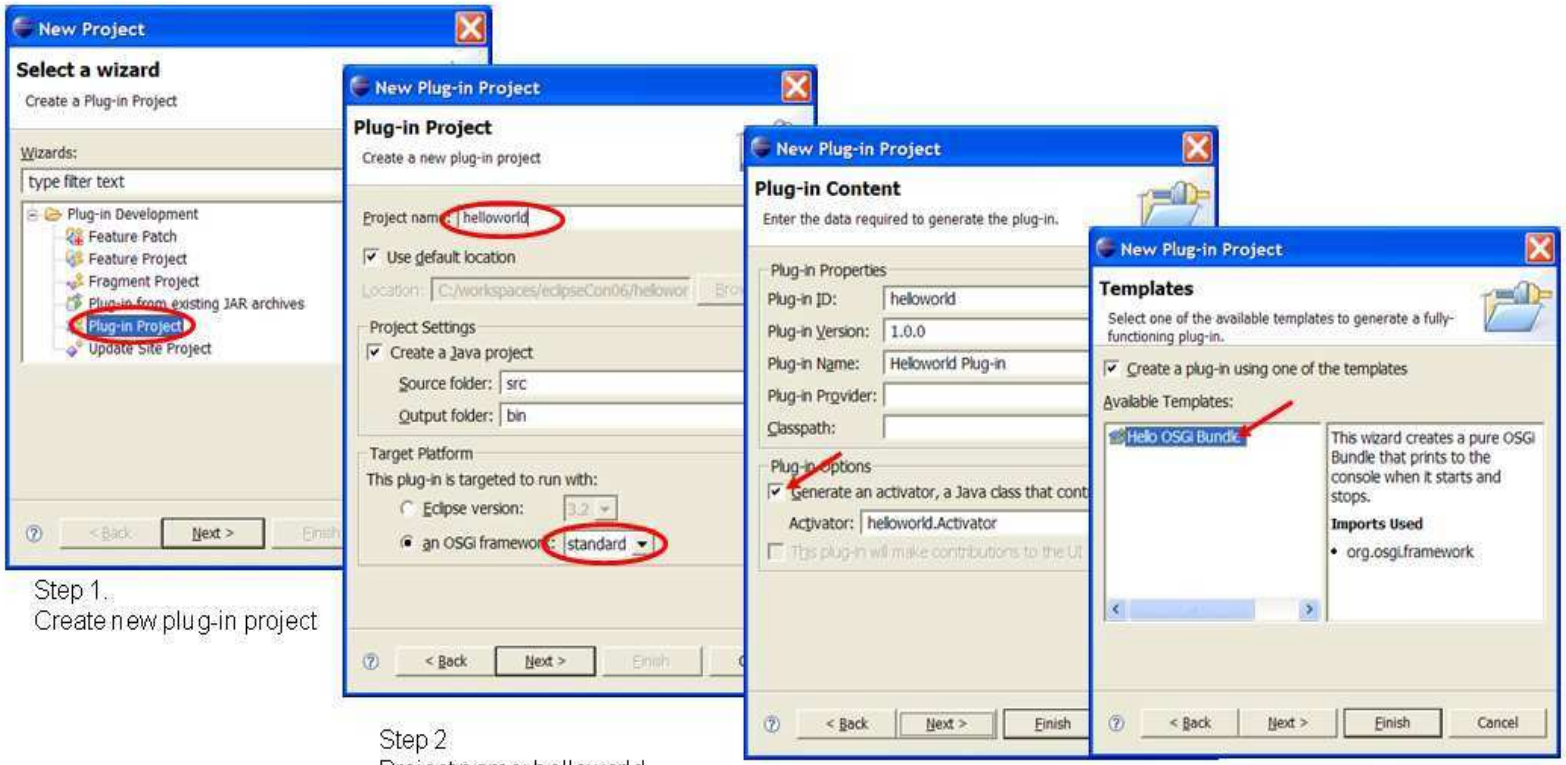
Running and testing OSGi

To run an OSGi bundle, you have to first run an OSGi framework and then install the bundle into the framework.

We'll use the Apache Felix / Equinox product.

Let us see how to create OSGI application with eclipse Plugin

Create the Hello World bundle - Eclipse



Step 1.
Create new plug-in project

Step 2
Project name: helloworld
an OSGi framework: standard

Step 3
Generate an activator

Step 4
Use the Hello OSGi Bundle template

Real code! Hello World (and Goodbye)

- The wizard has generated the code on the left
- This class implements the BundleActivator so that the Framework can start/stop the class
- The activator is referenced in the manifest
- package helloworld

```
public class HelloWorld  implements BundleActivator {
    public void start(
            BundleContext context)  throws Exception{
            System.out.println(
                    "Hello world!!");
    }

    public void stop(
            BundleContext context)  throws Exception {
                    System.out.println(  "Goodbye world!!");
    }
}
```

- The Manifest (in META-INF/MANIFEST.MF) is also generated by the wizard
- Eclipse provides convenient editors for the manifest
  - For the source: click on MANIEST.MF
- Notice:
  - Bundle-Activator (used to notify the bundle of lifecycle changes)
  - Import-Package (dependencies)

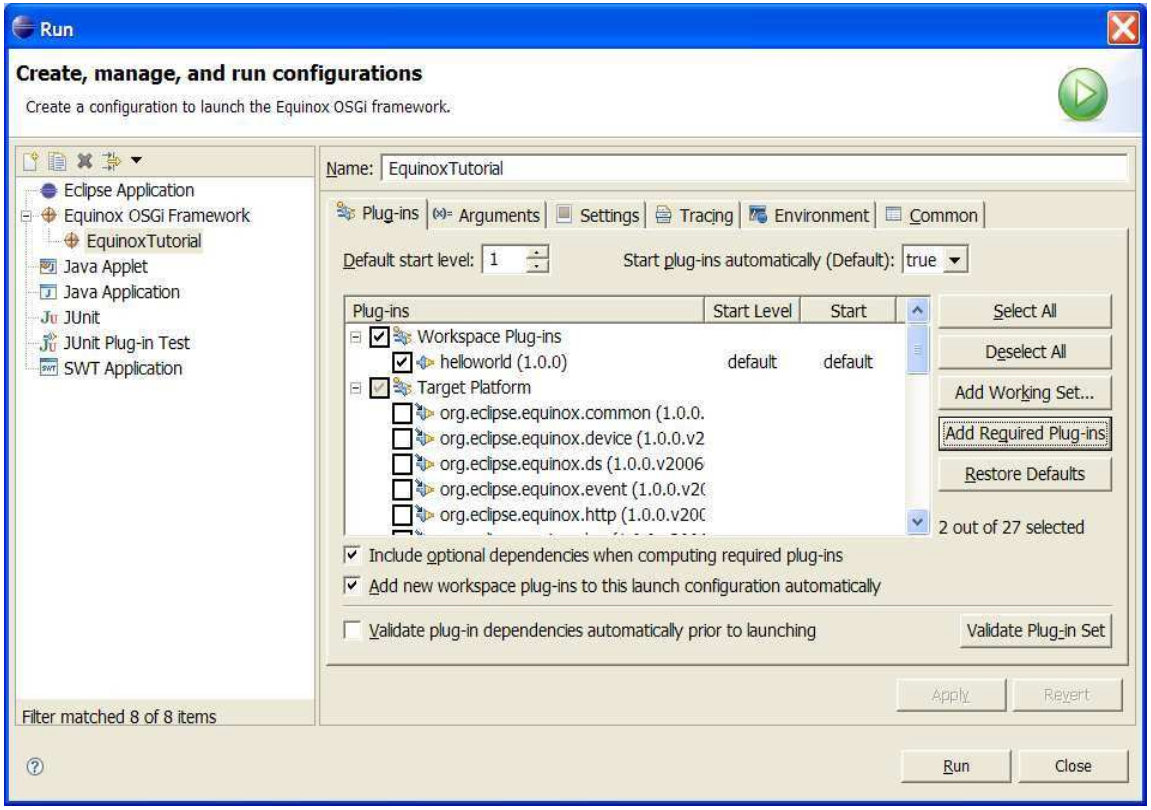
MANIFEST.Mf

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2  Bundle-Name:
Helloworld Plug-in
Bundle-SymbolicName: helloworld  Bundle-
Version: 1.0.0
Bundle-Localization: plugin
Bundle-Activator: helloworld.Activator
Import-Package:
 org.osgi.framework;version="1.3.0"
```

Eclipse Launch Configuration:

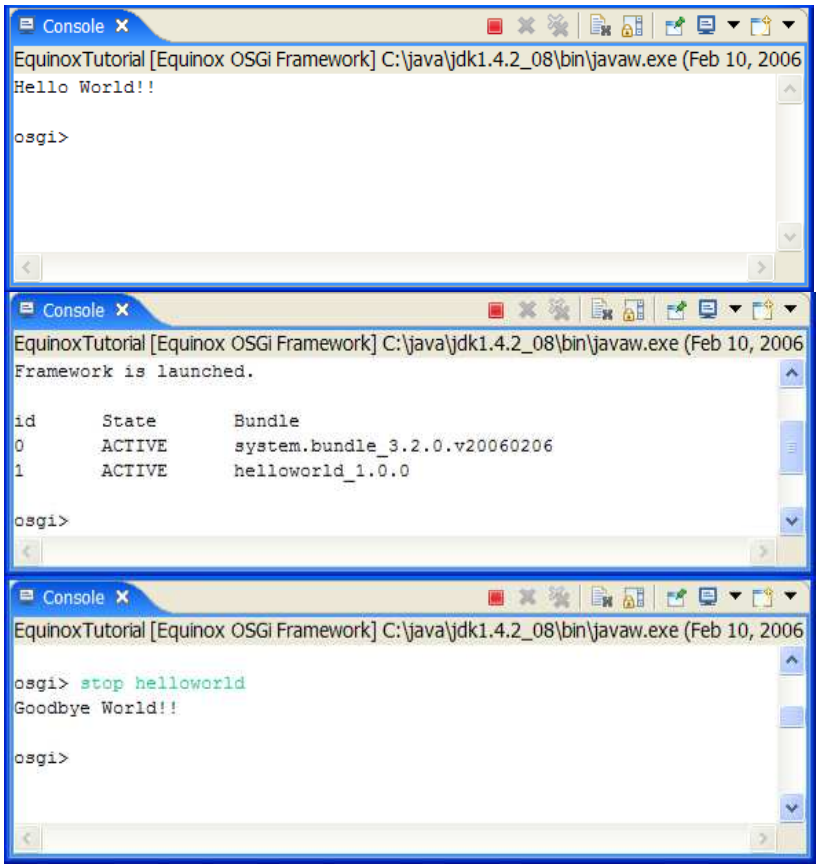
- The Launch Configuration is prepared for you
  - Run -> Run … -> EclipseTutorial
- Deselect “Target Platform” checkbox
  - This removes all possible bundles from the launch configuration
- Select “Add Required Plug-ins”
  - This calculates from the dependency information, which bundles are required to run our *helloworld* example
- Press Run
- The Framework is a console application

Equinox Launch Configuration



Run the Hello World bundle

- The Framework now runs the *helloworld* example
  - See the printed text
- It also runs a Framework console
  - Equinox specific
- Type “ss” (show status)
  - Look at the active bundles
  - Notice the number for the helloworld bundle. This is the bundle id.
- Type “stop <symbolic-name>”


Console
EquinoxTutorial [Equinox OSGi Framework] C:\java\jdk1.4.2_08\bin\javaw.exe (Feb 10, 2006
Hello World!!
osgi>
Console
EquinoxTutorial [Equinox OSGi Framework] C:\java\jdk1.4.2_08\bin\javaw.exe (Feb 10, 2006
Framework is launched.
id State Bundle
0 ACTIVE system.bundle_3.2.0.v20060206
1 ACTIVE helloworld_1.0.0
osgi>
Console
EquinoxTutorial [Equinox OSGi Framework] C:\java\jdk1.4.2_08\bin\javaw.exe (Feb 10, 2006
osgi> stop helloworld
Goodbye World!!
osgi>

Tutorial:- OSGI Application and run in eclipse only.
First OSGi bundle

OSGI Implementation:

- Felix
- Equinox
- Karaf

Apache Felix, the open source OSGi framework

Apache Felix is an open-source community effort to implement the OSGi Service Platform Release 4 Core specification under the Apache license.

The Felix project provides many services specified in the OSGi Service Compendium specification such as:

- Log Service specification implementation
- Http Service specification
- Configuration Admin Service
- Metatype service implementation
- Preferences service implementation
- Service Component Runtime
- Event Admin Service specification
- UPnP Device service implementation

Eclipse/Equinox

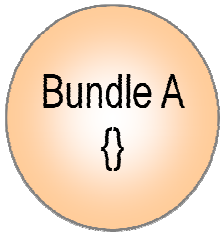
What is Equinox ?

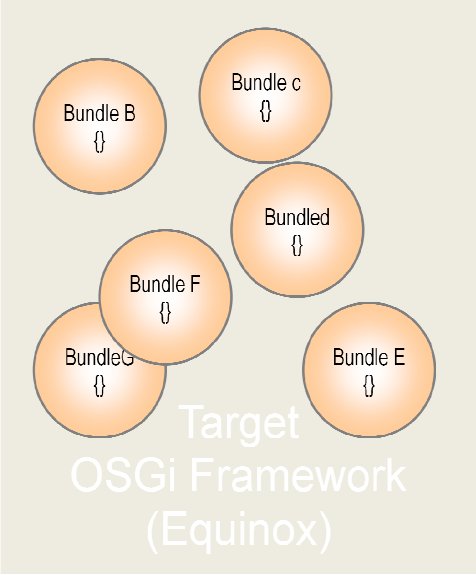
- An open source community focused on OSGi Technology
  - Develop OSGi specification implementations
  - Prototype ideas related to OSGi
- An OSGi Framework implementation
  - Core of the Eclipse runtime
  - Provides the base for Eclipse plug-in collaboration
  - Fully compatible with the OSGi R4 specification
- New for Eclipse 3.2 – Other specification implementations
  - Device Manager, Declarative Services, Event Admin, HTTP Service, Log Service, Metatype Service, Preferences Service, User Admin, Wire Admin – More on the way!!

The Equinox Target Environment

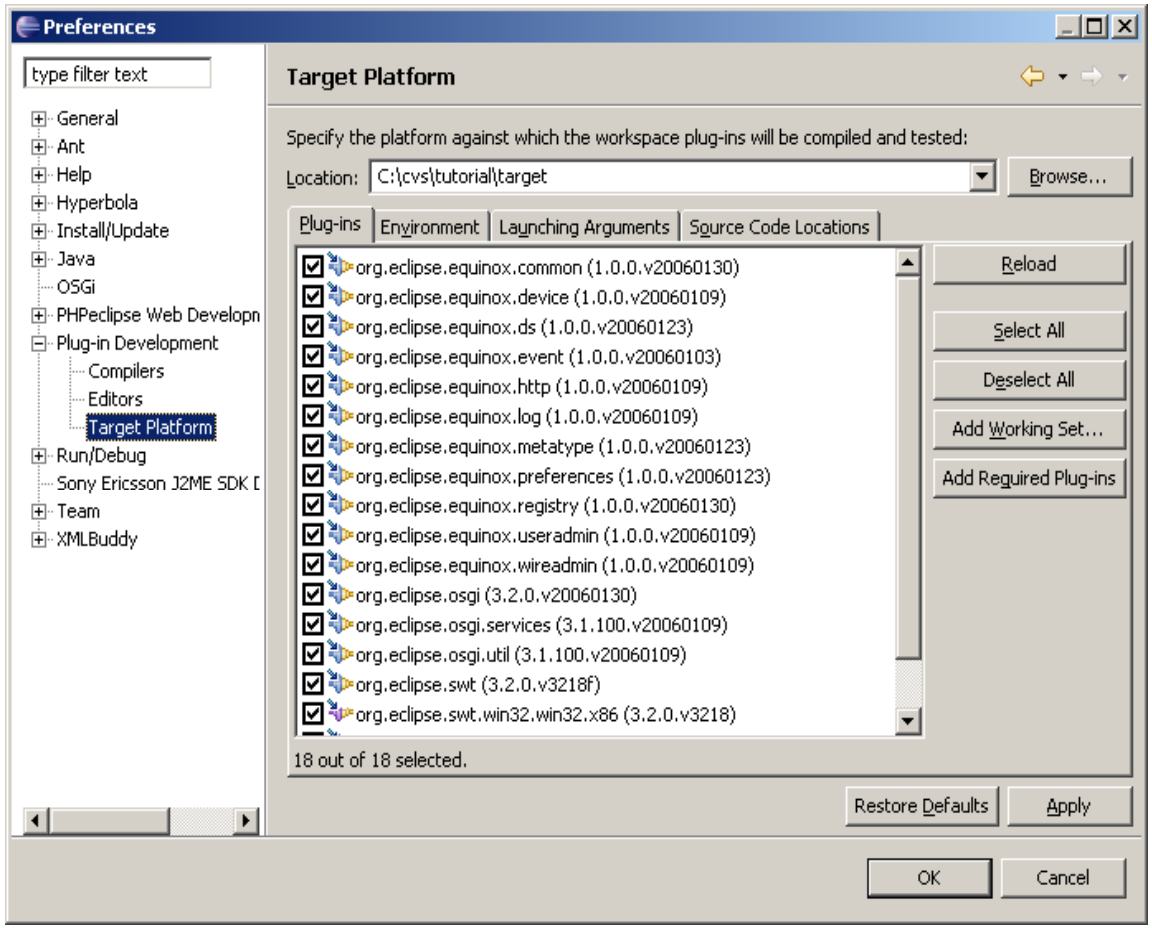
- Eclipse makes it *easy* to develop for OSGi Service Platforms
- A *target platform*
  - Contains a set of bundles
  - Defines runtime parameters
- To Define the Target Platform, goto:
  - Preferences ->Plug-in Development ->Target Platform
  - Select the target project in your workspace as location

The Equinox Target Environment





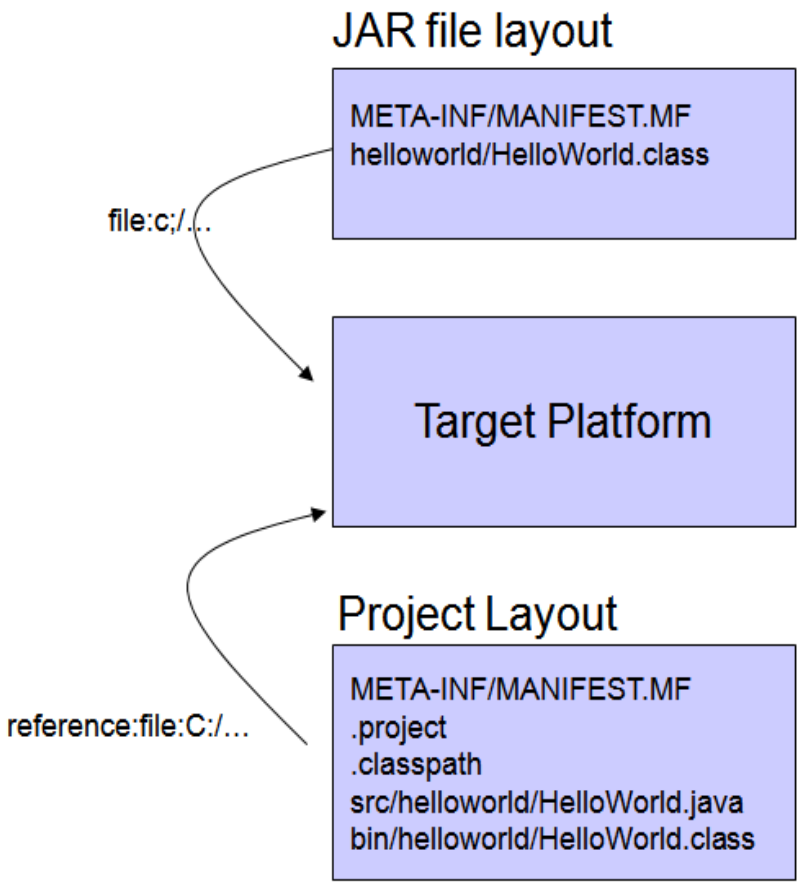
Setting up the Target Platform



Self-Hosting Bundle Projects

- Normally, a bundle is packaged in a *JAR* file
  - The traditional edit-compile-debug cycle.
- Self-Hosting Allows for quick debugging of bundle code
  - No packaging steps
  - No deployment steps
  - Just code/save/run
- Some changes require update of the bundle in the Framework
  - Console:

    update <symbolic-name>



Apache Karaf

HP

Apache Karaf

- Apache Karaf is a modern and polymorphic container.
- Karaf can be used standalone as a container, supporting a wide range of applications and technologies. It also supports the "run anywhere" (on any machine with Java, cloud, docker images, ...) using the embedded mode.
- It's a lightweight, powerful, and enterprise ready platform.
- With this flexibility, Karaf is the perfect solution for microservices, systems integration, big data, and much more.
- Apache Karaf is powered by OSGi

HP

Start the server

- Extract the apache karaf archive to a new folder
- bin\karaf.bat

```
        __ __                  ____
       / //_/____ _____ _____ / __/
      / ,<  / __ `/ ___/ __ `/ /_
     / /| |/ /_/ / /  / /_/ / __/
    /_/ |_|\__,_/_/   \__,_/_/

  Apache Karaf (4.0.0)

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown Karaf.

karaf@root()>
```

Shell console basics

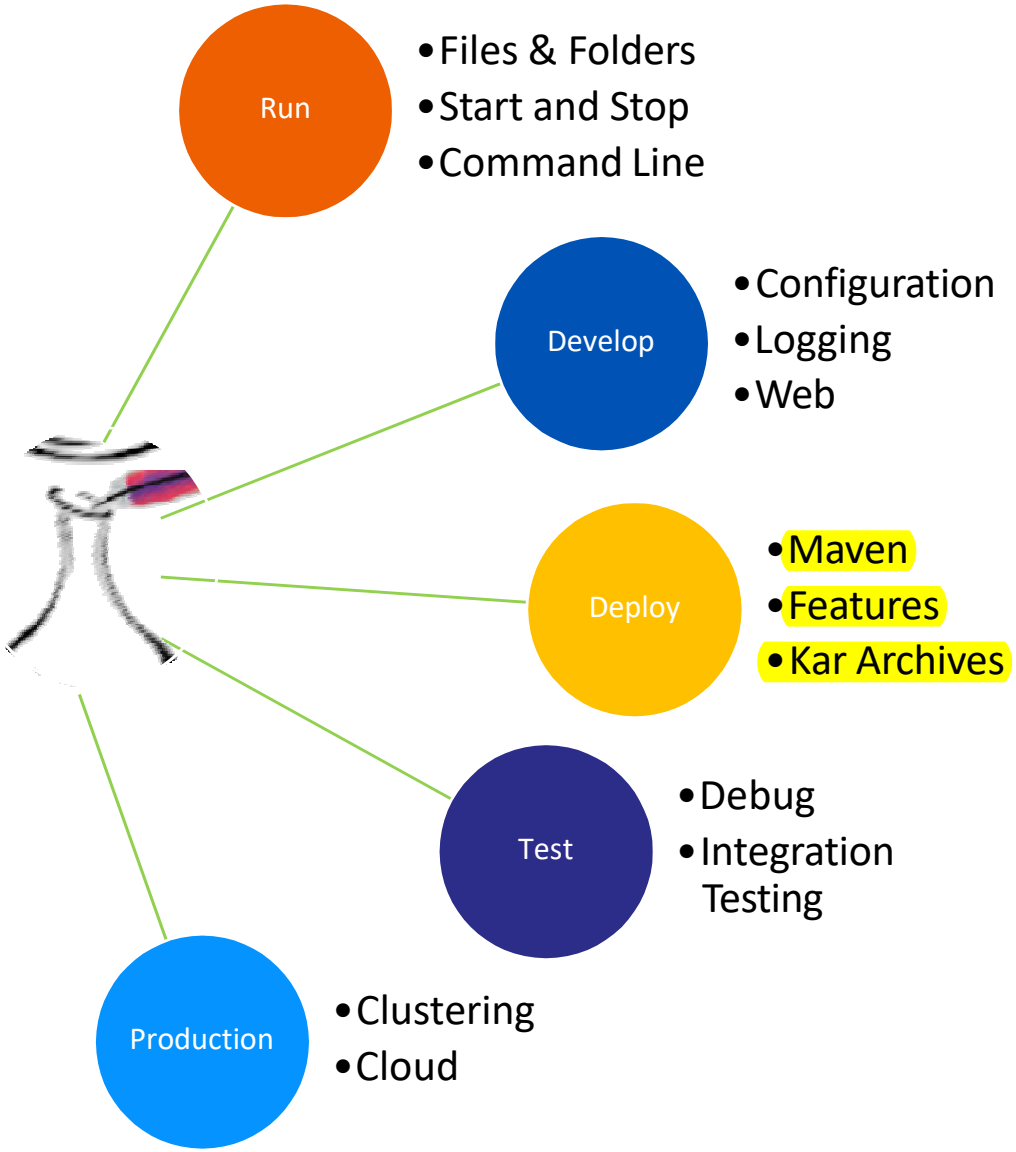
```
karaf@root()> bundle:list --help
DESCRIPTION
        bundle:list

        Lists all installed bundles.

SYNTAX
        bundle:list [options] [ids]
```

```
karaf@root()> feature:repo-add camel 2.15.2
Adding feature url mvn:org.apache.camel.karaf/apache-camel/2.15.2/xml/features
karaf@root()> feature:install camel-spring
karaf@root()> bundle:install -s mvn:org.apache.camel/camel-example-osgi/2.15.2
Bundle ID: 82
```

```
karaf@root()> bundle:stop camel-example-osgi
karaf@root()> bundle:uninstall camel-example-osgi
```

```
karaf@root()> system:shutdown
```

HP Features


Run
•Files & Folders
•Start and Stop
•Command Line
Develop
•Configuration
•Logging
•Web
Deploy
•Maven
•Features
•Kar Archives
Test
•Debug
•Integration Testing
Production
•Clustering
•Cloud

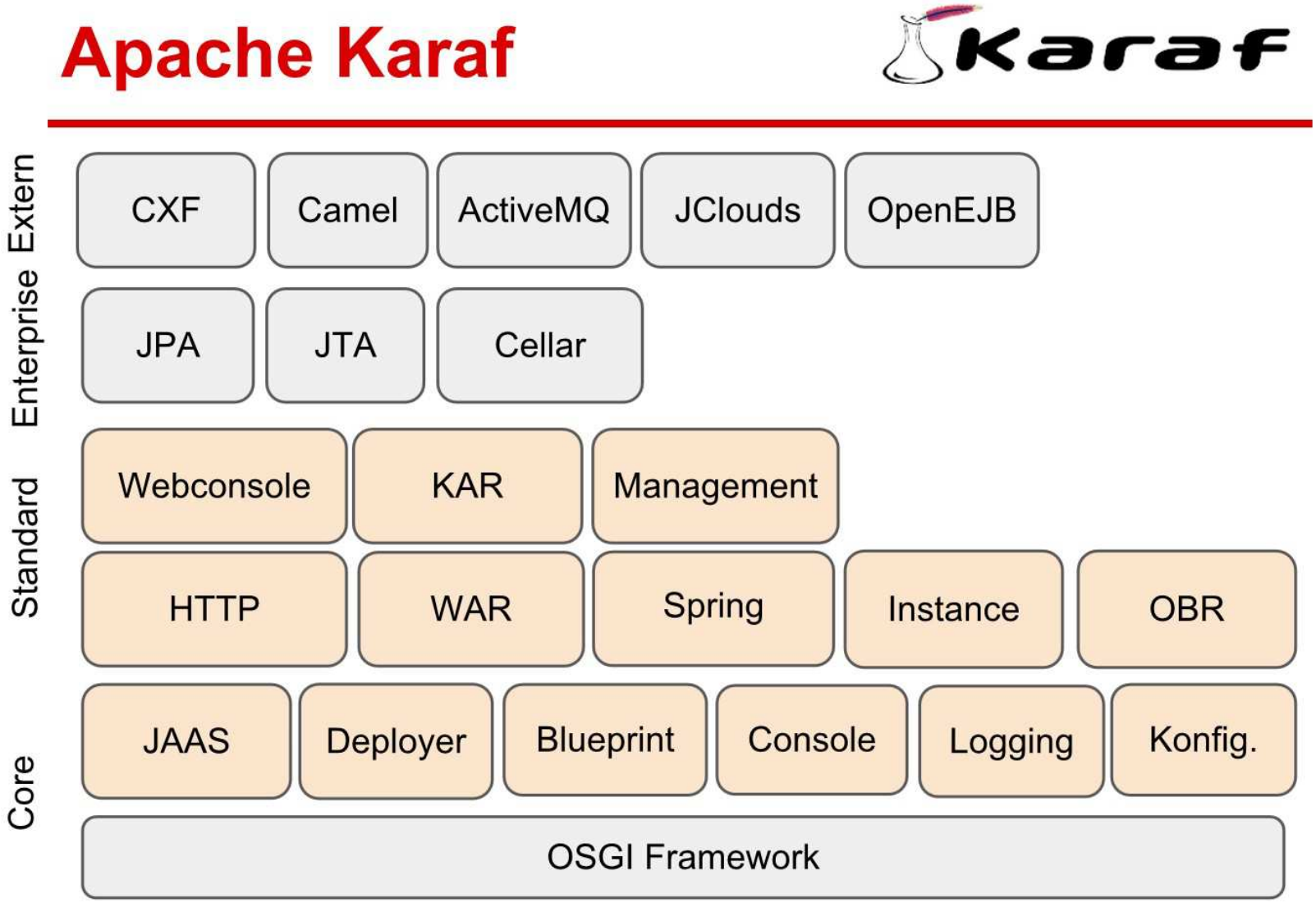
HP

Apache Karaf Highlights

- Flexible Provisioning and Deployment
- Application provisioning by „Features“
- Lightweight and modular
- Advanced Logging
- Dynamic Configuration
- Command Line
- Remote Management
- Supports different OSGi Frameworks
- Lots of available features

HP

Apache Karaf Overview



HP

Important Files and Folders

+---bin	: Startup scripts and commands
+---data	: Karaf work directory
| +---cache	: Installed bundles
| +---log	: Default log directory
| +---tmp	: Temporary files
+---deploy	: Directory for file system deployment
+---etc	: Configuration files
+---instances	: Instance management
+---lib	: Core libraries
+---system	: System bundle repository

HP

Karaf Specific Environment Variables

- KARAF_HOME: the location of your Apache Karaf installation (default is found depending where you launch the startup script).
- KARAF_BASE: the location of your Apache Karaf base (default is KARAF_HOME).
- KARAF_DATA: the location of your Apache Karaf data folder (default is KARAF_BASE/data).
- KARAF_ETC: the location of your Apache Karaf etc folder (default is KARAF_BASE/etc).
- KARAF_OPTS: extra arguments passed to the Java command line (default is null).
- KARAF_DEBUG: if 'true', enable the debug mode (default is null).

HP Commandline

```
        __ __                  ____
       / //_/____ __________ _/ __/
      / ,<  / __ `/ ___/ __ `/ /_
     / /| |/ /_/ / /  / /_/ / __/
    /_/ |_|\__,_/_/   \__,_/_/

  Apache Karaf (3.0.1)

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown Karaf.

karaf@root()> list
START LEVEL 100 , List Threshold: 50
 ID | State  | Lvl | Version         | Name
-------------------------------------------------------------------------------
111 | Active |  80 | 1.5.0.SNAPSHOT | hawtio :: hawtio-json-schema-mbean
112 | Active |  80 | 1.5.0.SNAPSHOT | hawtio :: hawtio-osgi-jmx
113 | Active |  80 | 1.5.0.SNAPSHOT | hawtio :: hawtio-web
114 | Active |  80 | 1.5.0.SNAPSHOT | hawtio :: Karaf terminal plugin
115 | Active |  80 | 1.5.0.SNAPSHOT | hawtio :: hawtio-maven-indexer
131 | Active |  60 | 0.9.0          | PaaS+ - Core - Util
134 | Active |  73 | 0.9.0          | PaaS+ - Services - Service API
147 | Active |  80 | 3.0.1          | Apache Karaf :: JNDI :: Command
157 | Active |  60 | 0.9.0          | PaaS+ - Core - Datamapper API
158 | Active |  60 | 0.9.0          | PaaS+ - Core - Datamapper - Orika
160 | Active |  60 | 0.9.0          | PaaS+ - Core - QueryDSL JPA, Lucene Processor
171 | Active |  74 | 0.9.0          | PaaS+ - Services - Persistence Service API
172 | Active |  74 | 0.9.0          | PaaS+ - Services - Persistence Service Provider
174 | Active |  75 | 0.9.0          | PaaS+ - Services - Persistence NoSql Service Provider
175 | Active |  75 | 0.9.0          | PaaS+ - Services - Report Service API
176 | Active |  75 | 0.9.0          | PaaS+ - Services - Report Provider
```

HP

Command Groups (Karaf 3.x)

- bundle:* - Install and control bundles
- dev:* - Development support
- feature:* - Manage features
- instance:* - Control of multiple instances
- jaas:* - JAAS role and rights management
- log:* - log display
- obr:* - Interaction with OSGi Bundle Repositories
- scr:* - Declarative Services management
- service:* - Manage OSGi Services
- shell:* - useful helper functions (i.e. grep, more, java info)
- ssh:* - SSH connections
- system:* - OSGi framework management
- web:* - WAR bundle management
- wrapper:* - OS service installation

HP

Remote Management

- Use command line via SSH:

```
ssh -p 38031 karaf@karaf-mydomain.paasplus.com
```

- Configuration via: org.apache.karaf.shell.cfg
- User and Role management via JAAS
  - Configure via users.properties
  - org.apache.karaf.command.acl.* for command specific permissions
  - Default roles: viewer, manager, admin
- Connect to other Karaf using ssh command
- Connect to local running Karaf using the client script

HP Webconsole

Apache Karaf Web Console Bundles

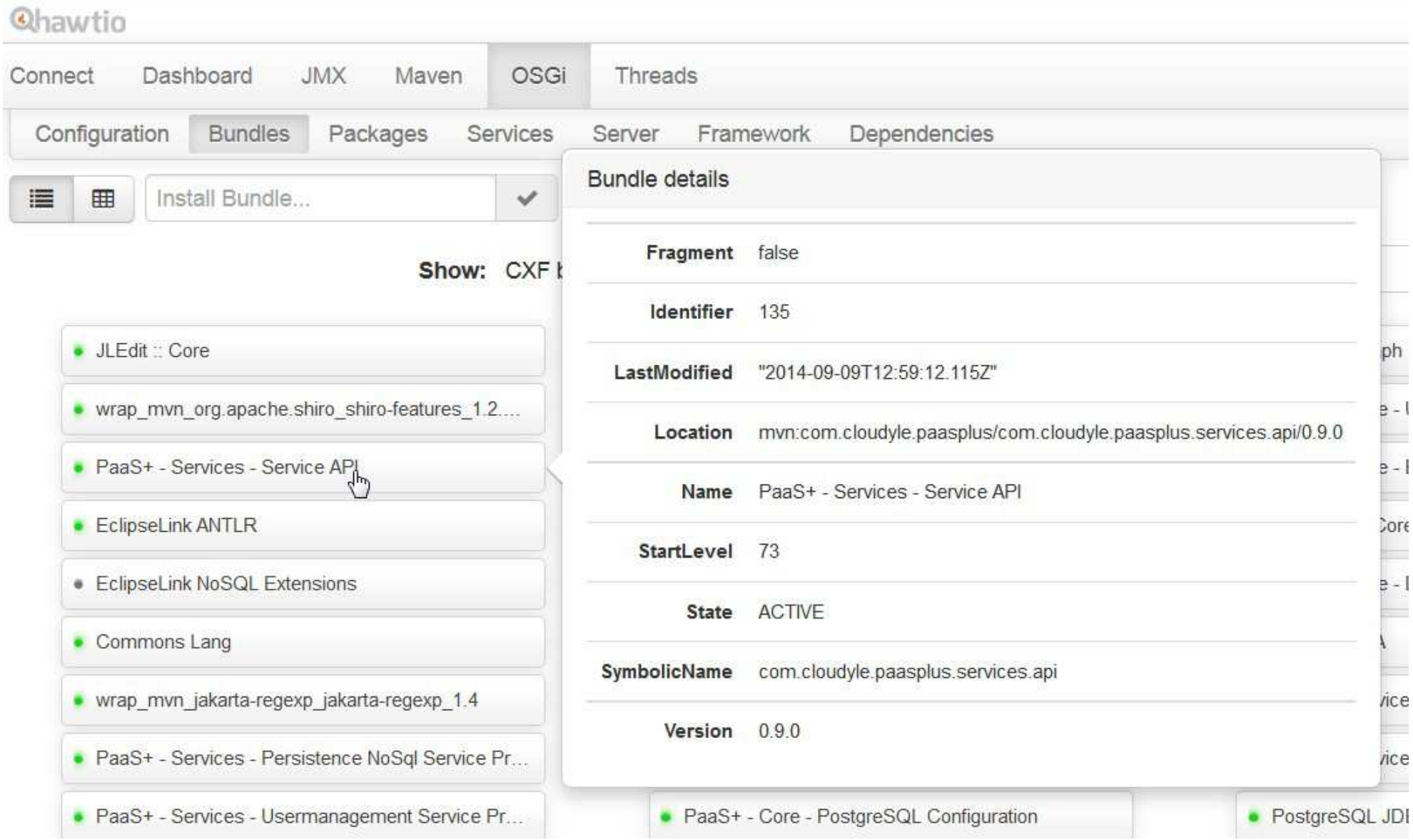
Main
OSGi
Web Console

Bundle information: 298 bundles in total - all 298 bundles active

PaaS | Apply Filter | Filter All | Reload | Install/Update... | Refresh Packages

Id	Name	Version	Category	Status	Actions
258	PaaS+ - Core - Camunda BPM OSGi *(com.cloudyle.paasplus.core.camunda)*	0.9.0		Active	
158	PaaS+ - Core - Datamapper - Orika *(com.cloudyle.paasplus.core.datamapper.orika)*	0.9.0		Active	
157	PaaS+ - Core - Datamapper API *(com.cloudyle.paasplus.core.datamapper.api)*	0.9.0		Active	
148	PaaS+ - Core - Eclipselink JPA *(com.cloudyle.paasplus.core.eclipselink)*	0.9.0		Active	
256	PaaS+ - Core - PostgreSQL Configuration *(com.cloudyle.paasplus.core.postgrescfg)*	0.9.0		Active	
160	PaaS+ - Core - QueryDSL JPA, Lucene Processor *(com.cloudyle.paasplus.core.querydsl)*	0.9.0		Active	
131	PaaS+ - Core - Util *(com.cloudyle.paasplus.core.util)*	0.9.0		Active	

HP Webconsole



HP Scripting

- Scripting allows customizing console
  - Variable assignment:
    msg = "Welcome to PaaS+"
  - Lists, Maps
    map = [Jan=1 Feb=2 Mar=3]
  - Pipe commands
    ($.context bundles) | grep -i PaaS+
  - Flow control
    shell:if, shell:each
- Startup scripts configured in shell.init.script

```
paasplus:refresh={
feature:repo-remove -u mvn:com.cloudyle.paasplus/$args/LATEST/xml/features;
feature:repo-add mvn:com.cloudyle.paasplus/$args/LATEST/xml/features;
feature:install $args } ;
```

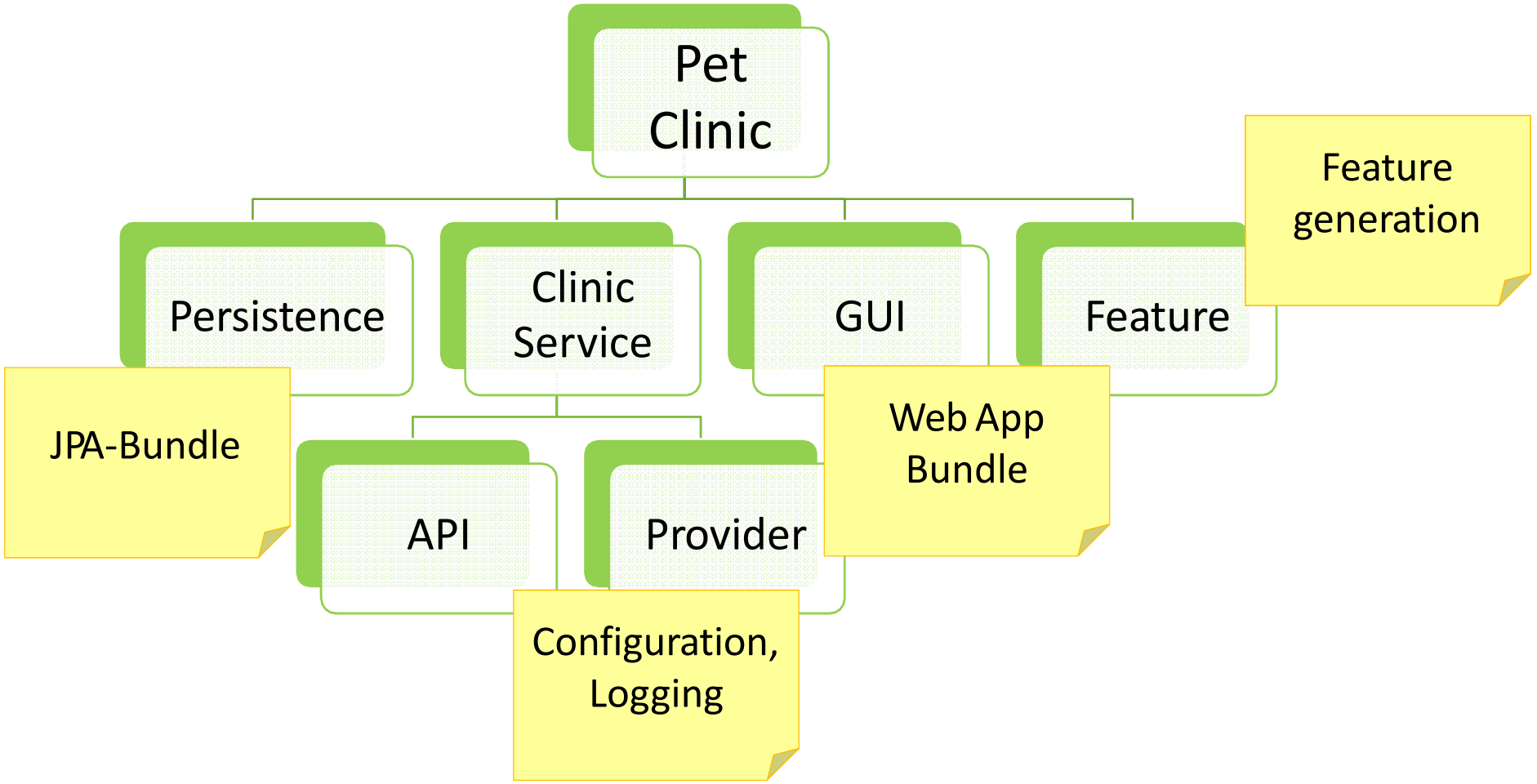
HP

Developer Commands

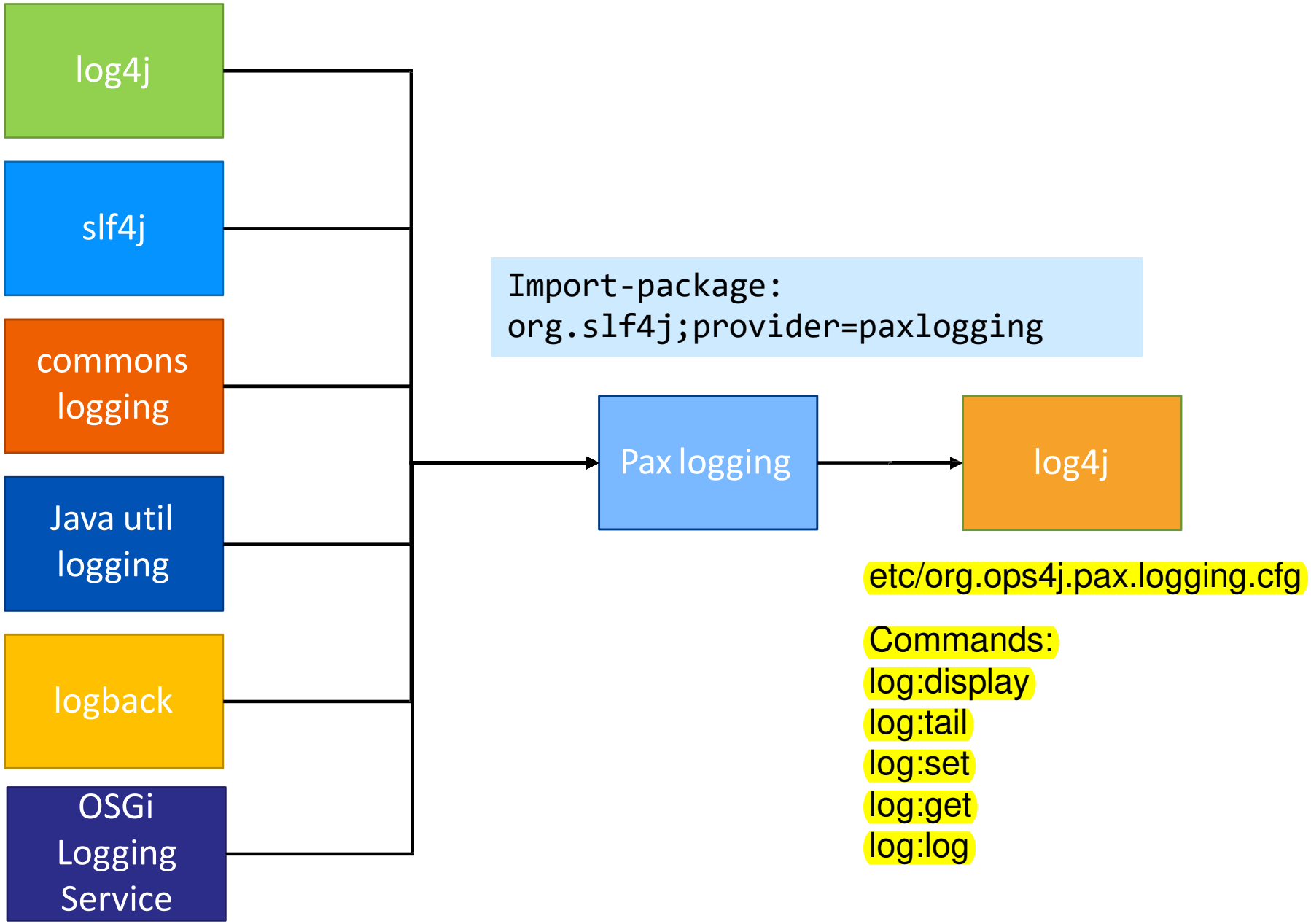
- dev:dump-create
  - Creates a system dump of Karaf
- bundle:diag
  - Gives details why bundle did not start
- bundle:dynamic-import
  - enable or disable the dynamic import of a given bundle
- log:exception-display
  - Displays the last occurred exception from the log
- log:tail
  - Continuously display log entries
- bundle:tree-show
  - Show bundle dependency tree
- bundle:watch
  - Automatic update from maven repo

HP Pet Clinic

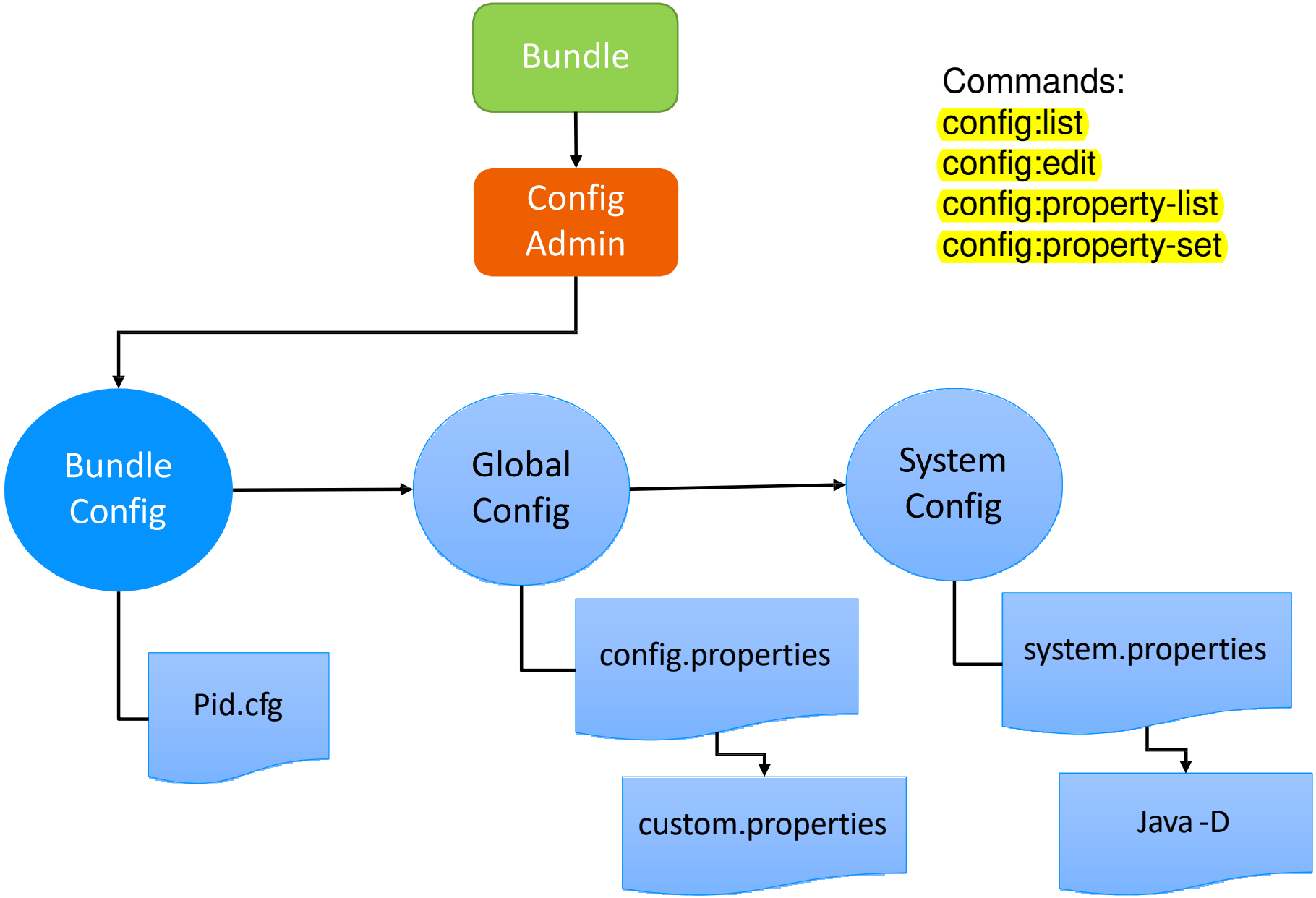
https://github.com/Cloudyle/petclinic



HP Logging


log4j
slf4j
commons logging
Java util logging
logback
OSGi Logging Service
Import-package:
org.slf4j;provider=paxlogging
Pax logging
log4j
etc/org.ops4j.pax.logging.cfg
Commands:
log:display
log:tail
log:set
log:get
log:log

HP Configuration

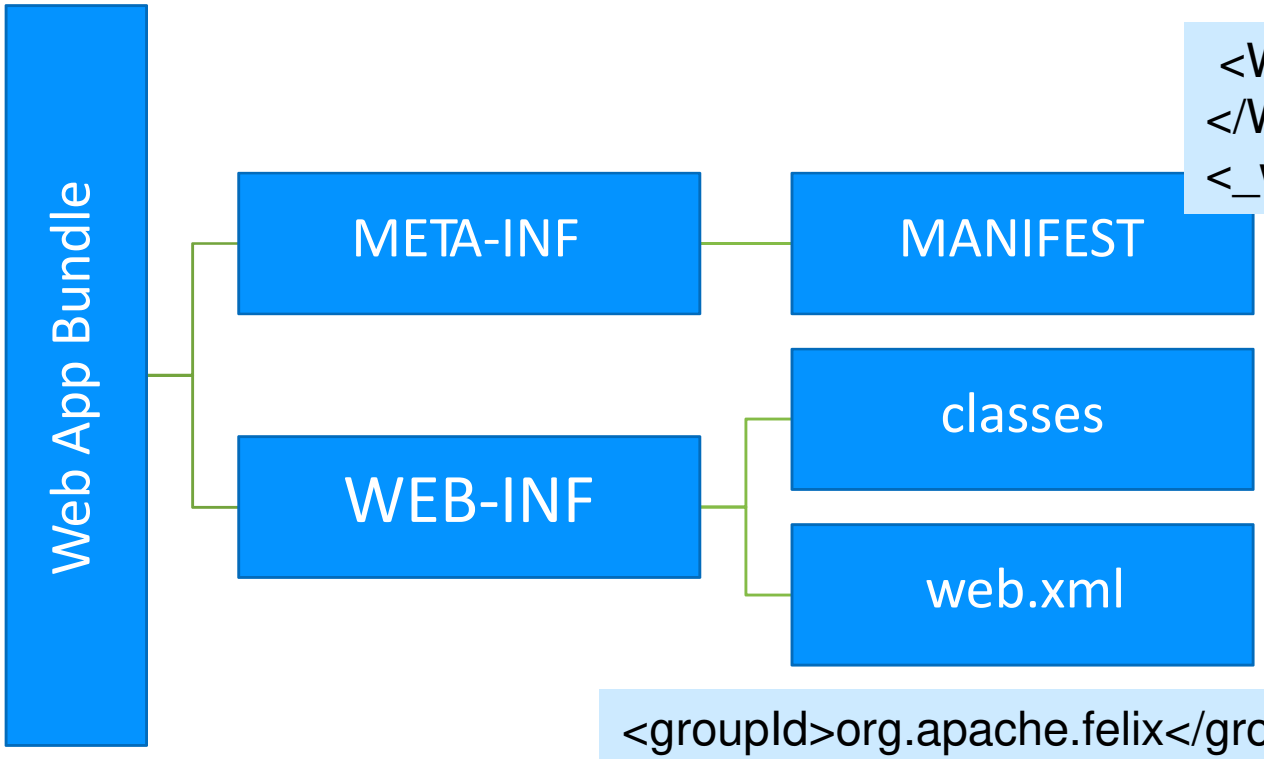


HP

Web Applications

- Webapp support via Pax Web feature
- Integrated Jetty
  - etc/jetty.xml
- OSGi http Service (Whiteboard pattern)
  - Register Web application resources as OSGi services
- Full Webapp
  - Deploy war or wab
  - CDI support
- Commands
  - web:list, stop, start (Webapps)
  - http:list (Servlets)

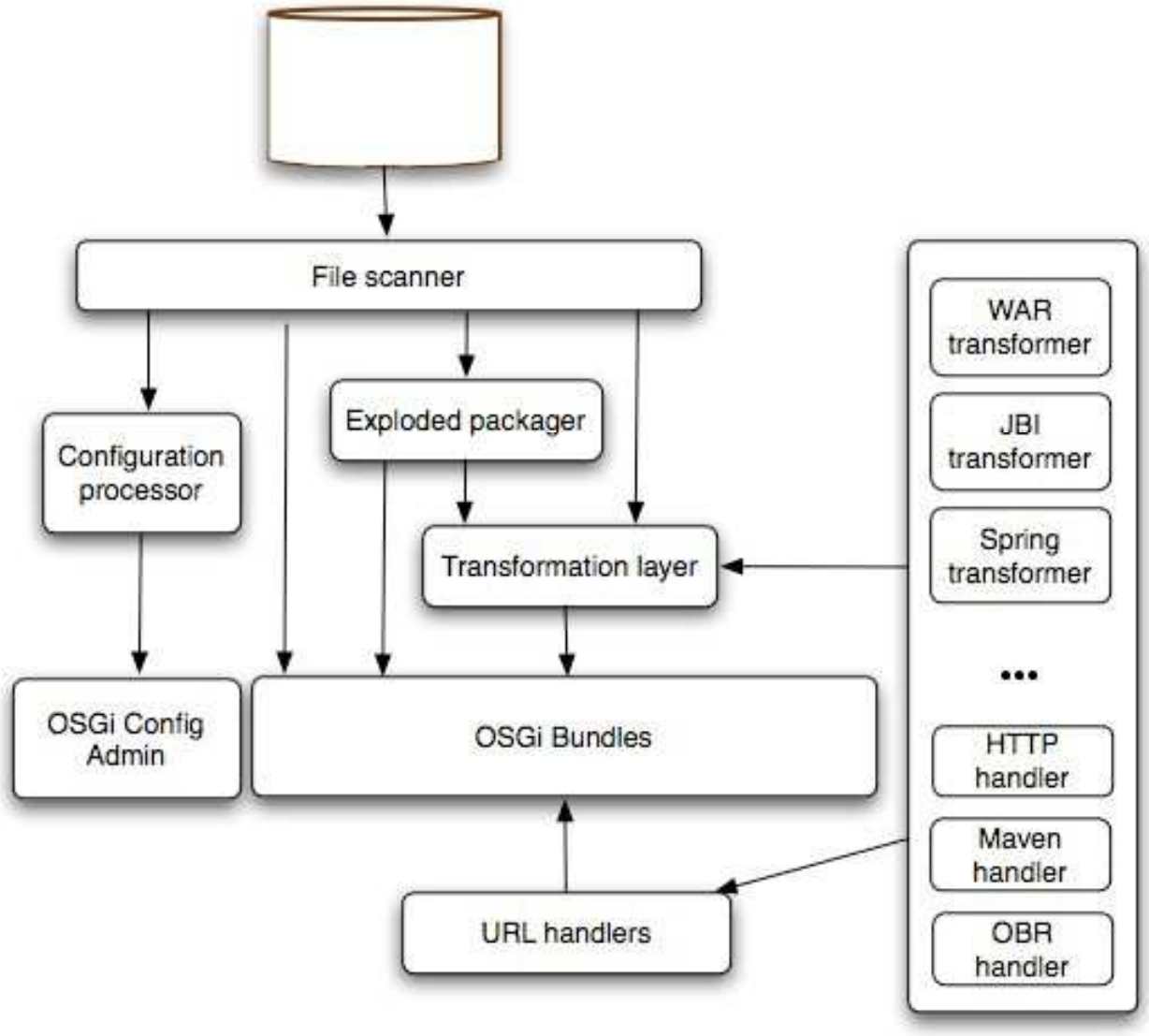
HP Web Application Bundle




```
<Web-ContextPath>/petclinic
</Web-ContextPath>
<_wab>src/main/resources</_wab>
```

```
<groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
    <extensions>true</extensions>
     <configuration>
       <supportedProjectTypes>
         <supportedProjectType>jar</supportedProjectType>
         <supportedProjectType>bundle</supportedProjectType>
         <supportedProjectType>war</supportedProjectType>
       </supportedProjectTypes>
```

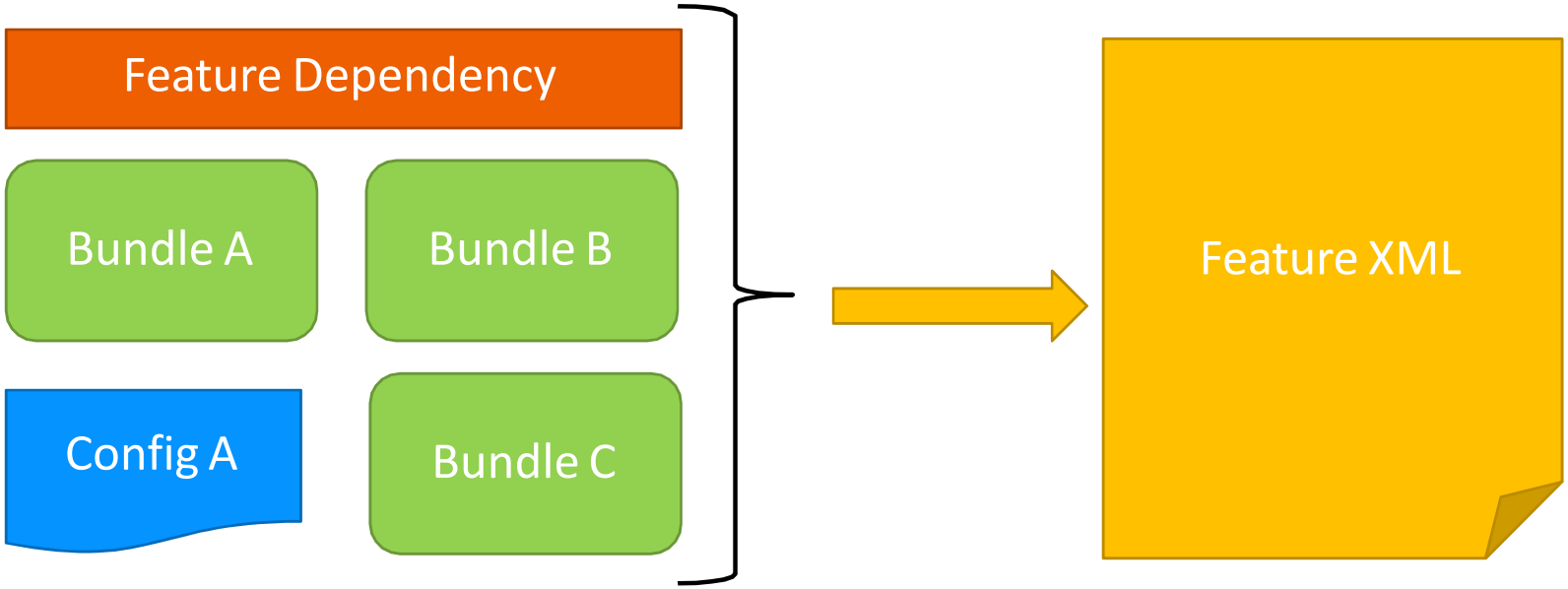
HP Deployment



Deployment of:

- Bundles
- Blueprint files (xml)
- Features
- Kar Archives
- „plain“ jar files

HP Features




```
<feature name="com.cloudyle.paasplus.samples.petclinic.feature" version="0.9.0"
description="PaaS+ Sample - PetClinic - Deployment feature" resolver="(obr)"
install="auto">
 <details>Pet Clinic Feature.</details>
 <configfile finalname="/etc/com.cloudyle.paasplus.petclinic.cfg">
mvn:com.cloudyle.paasplus.samples/com.cloudyle.paasplus.samples.petclinic.provider/0.
9.0/cfg</configfile>
<feature>paasplus-persistence-service</feature>
<bundle start-level="75">mvn:com.cloudyle.paasplus.samples/
com.cloudyle.paasplus.samples.petclinic.persistence/0.9.0</bundle>
<bundle start-level="75">mvn:com.cloudyle.paasplus.samples/
com.cloudyle.paasplus.samples.petclinic.api/0.9.0</bundle>
</feature>
```

HP Maven Deployment

- Deploy via any Maven repository
  - Bundles
  - Features
  - Configurations
- Add additional repos to org.ops4j.pax.url.mvn.cfg

```
org.ops4j.pax.url.mvn.repositories= \
    http://nexus.paasplus.com:8081/nexus/content/groups/public/, \
    http://repo1.maven.org/maven2@id=central, \
```

- Install features or bundles

```
repo-add mvn:com.cloudyle.paasplus/com.cloudyle.paasplus.karaf.deployment.services/
    LATEST/xml/features;
feature:install paasplus-persistence-service;
bundle:install mvn:com.cloudyle.paasplus.samples/
    com.cloudyle.paasplus.samples.petclinic.api;
```

- Useful Commands
  - bundle:install, update, list
  - feature:install, list

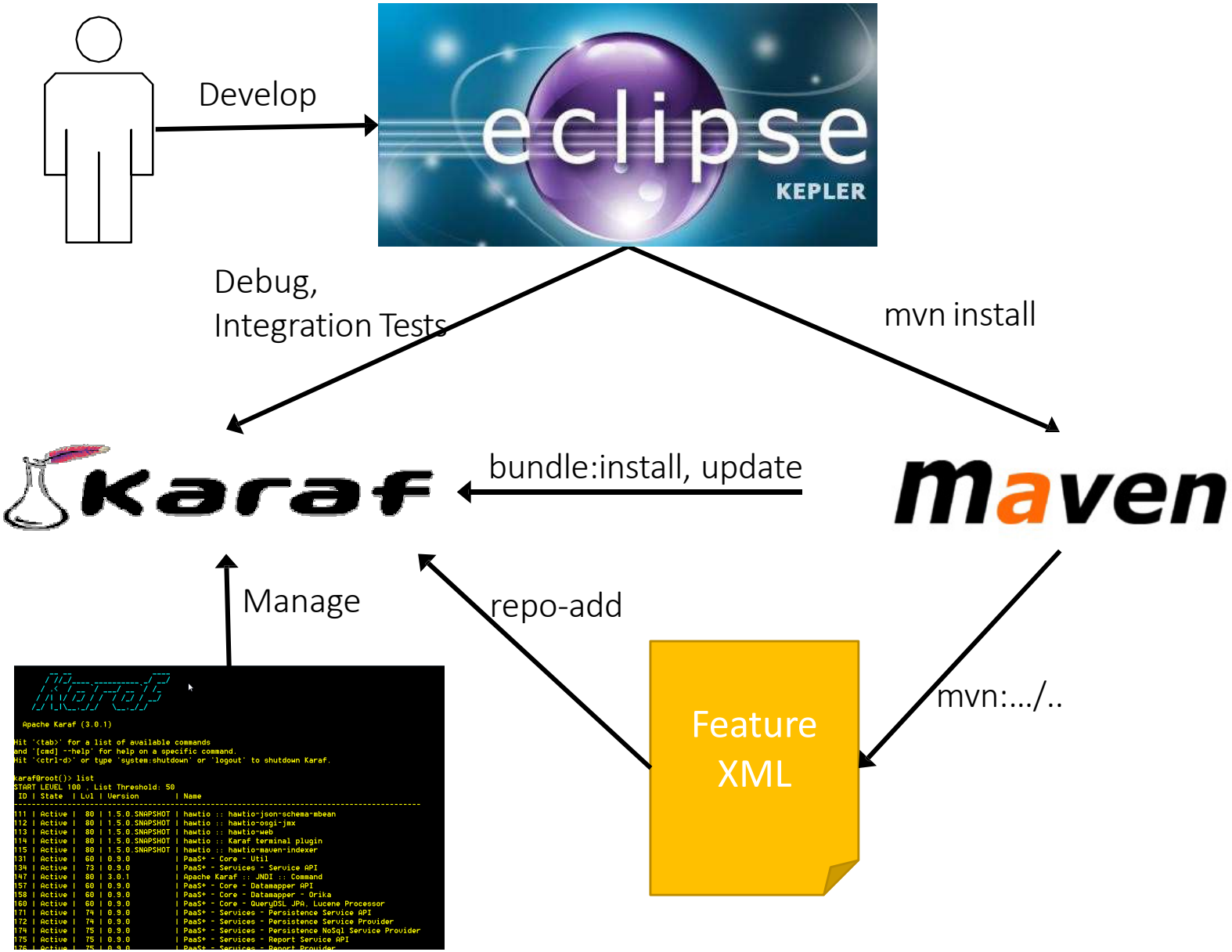
HP

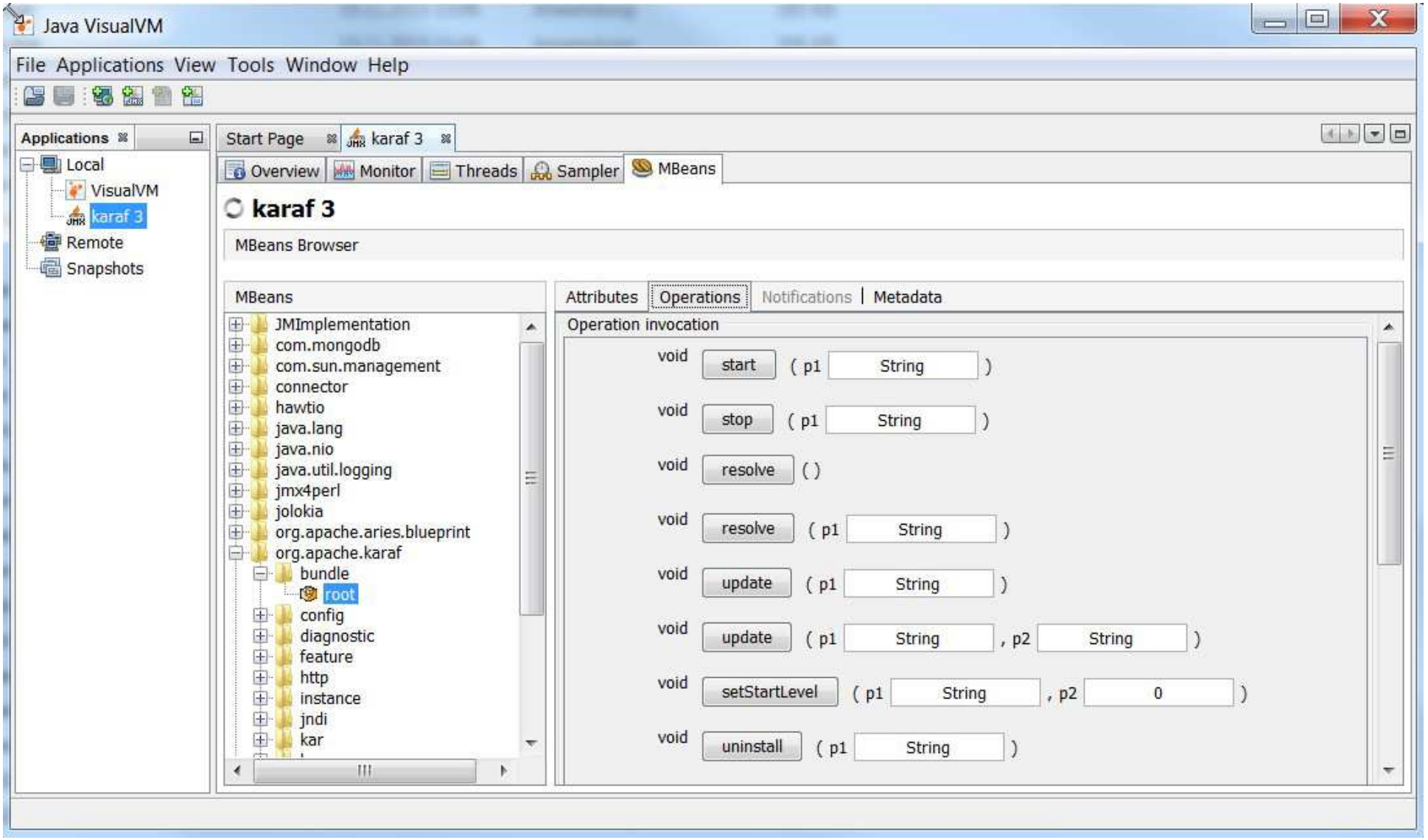
Karaf Maven Plugin

- Generate feature.xml with project dependencies
- Generate command documentation
- Kar packaging
- Custom Karaf distribution
- Kar and feature packaging

```
<plugin>
  <groupId>org.apache.karaf.tooling</groupId>
  <artifactId>karaf-maven-plugin</artifactId>
  <version>3.0.2</version>
  <extensions>true</extensions>
</plugin>
```

HP Integrated Development Process




Java VisualVM
File Applications View Tools Window Help
Applications
Local
VisualVM
karaf 3
Remote
Snapshots
Start Page
karaf 3
Overview
Monitor
Threads
Sampler
MBeans
karaf 3
MBeans Browser
MBeans
JMImplementation
com.mongodb
com.sun.management
connector
hawtio
java.lang
java.nio
java.util.logging
jmx4perl
jolokia
org.apache.aries.blueprint
org.apache.karaf
bundle
root
config
diagnostic
feature
http
instance
jndi
kar
Attributes
Operations
Notifications
Metadata
Operation invocation
void start (p1 String)
void stop (p1 String)
void resolve ( )
void resolve (p1 String)
void update (p1 String)
void update (p1 String , p2 String)
void setStartLevel (p1 String , p2 0)
void uninstall (p1 String)

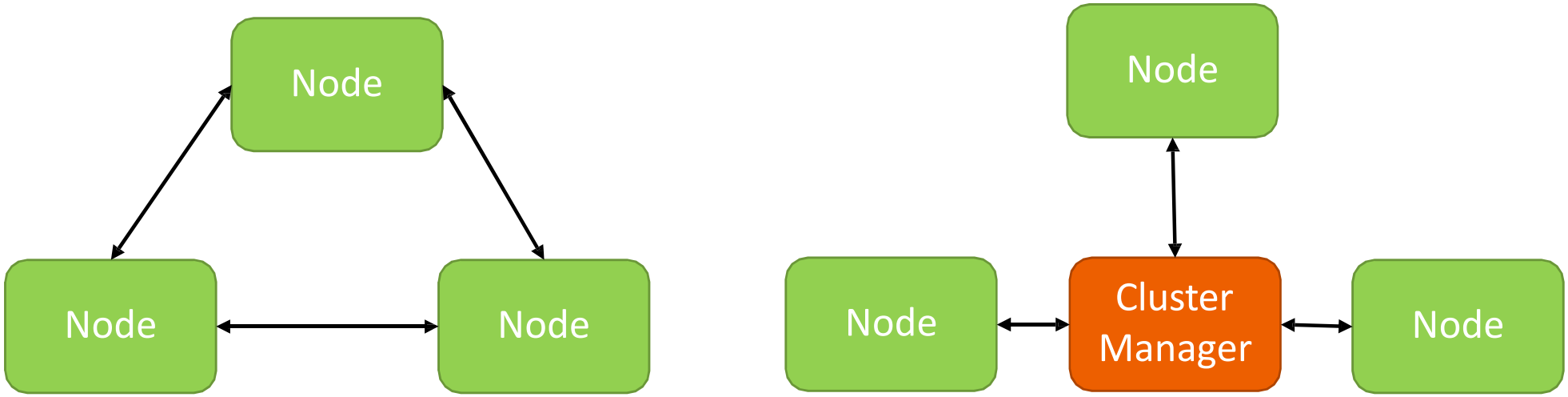
HP Integration Testing

- Automated Testing via PaxExam
- JUnit tests running in Apache Karaf instance
- Configure and manage Karaf via JUnit test

```
@RunWith(PaxExam.class)
public class ServiceProviderIT extends AbstractBaseTest{
  @Configuration
  public Option[] config(){
   return combine(super.config(), new Option[] {
      editConfigurationFileExtend("etc/org.apache.karaf.features.cfg", "featuresRepositories",
",mvn:com.cloudyle.paasplus:com.cloudyle.paasplus.karaf.deployment.services:xml:features"),
 new KarafDistributionConfigurationFileReplacementOption("etc/org.elasticsearch.connection.cfg",
new File("src/test/resources/etc/org.elasticsearch.connection.cfg")),
features("mvn:com.cloudyle.paasplus/com.cloudyle.paasplus.karaf.deployment.services/LATEST/xml/features", "paasplus-report-service"),
vmOption("-Djava.net.preferIPv4Stack=true"), vmOption("-XX:MaxPermSize=256M") });}

@Test
public void doTest(){
…
```

HP Cellar Clustering

- Cluster solution for Apache Karaf
- Based on Hazelcast
- Support for different topologies
- Synchronization of deployments and configurations
- Cross-Node Event publishing



Lab: Karaf Installation , Installed Bundle in equinox/Felix/Karaf and Karaf – Server Mode

DEPLOY THE HELLO BUNDLE TO FELIX/EQUINOX – APACHE KARAF

Developing Tools for OSGI:

- Eclipse
- Maven
- ANT
- Pax tools

MAVEN

- Maven 2 is a popular and powerful build tool
- Spans the whole build lifecycle of a software project: compilation, automated tests, packaging, and deployment.
- Maven 2 also offers good support for OSGi-based development.

Basics of Maven 2:

- Project creation
- the POM file
- build lifecycle
- dependency management
- multimodule projects

Installing Maven 2

- Maven 2 can be downloaded from its official website: http://maven.apache.org.

- create an `M2_HOME` environment variable

Creating a simple project – using Maven `mvn archetype:generate`

The Maven 2 project structure and the POM file

```
maven101
  pom.xml
  src/
    main/
      java/
        com/manning/sdmia/appb/App.java
    test
      java/
        com/manning/sdmia/appb/AppTest.java
```

Maven 2 standard directory layout

Directory	Description
`src/main/java`	Application classes
`src/main/resources`	Application resources (non-Java files)
`src/main/webapp`	Web application files
`src/test/java`	Test classes
`src/test/resources`	Test resources (non-Java files)
`target`	Build output

POM.xml

POM file created with the quickstart archetype

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.manning.sdmia.appb</groupId>
  <artifactId>maven101</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>maven101</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Identifies project

Sets JUnit as dependency

Compiling and packaging: the build lifecycle

Main phases of the Maven 2 default build

Phase	Description
`compile`	Compiles the source code of the project
`test-compile`	Compiles the test source code of the project
`test`	Runs the tests of the project
`package`	Packages the compiled code and resource in the appropriate archive format (JAR, WAR, and so on)
`install`	Copies the generated package in the local repository
`deploy`	Copies the generated package in a remote repository for sharing with other developers

DECLARING DEPENDENCIES

```
<dependencies>  (...)
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>  (...)
</dependencies>
```

REPOSITORIES

The default remote repository is called the “central Maven 2 repository,” and it’s located at http://repo1.maven.org/maven2/.

Repository entries can be added at the end of the POM, in a `repositories` element.

Adding the SpringSource EBR

```
<repositories>

  <repository>
    <id>com.springsource.repository.bundles.release</id>
    <name>SpringSource EBR - SpringSource Bundle Releases</name>
    <url>
    http://repository.springsource.com/maven/bundles/release
    </url>
  </repository>

  <repository>
    <id>com.springsource.repository.bundles.external</id>
    <name>SpringSource EBR - External Bundle Releases</name>
    <url>
    http://repository.springsource.com/maven/bundles/external
    </url>
  </repository>

</repositories>
```

Apache Felix Bundle Plugin:

The Apache Felix Bundle Plugin for Maven 2 builds on top of Bnd to package Maven projects as OSGi bundles.

[redacted] Basic setup for the Apache Felix Bundle Plugin

```
<project (...)>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <artifactId>parent</artifactId>
    <groupId>com.manning.sdmia.appb.directory</groupId>
    <version>1.0.0</version>
  </parent>
  <artifactId>directory.domain</artifactId>
  <name>Directory Domain</name>
  <packaging>bundle</packaging>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <version>2.0.1</version>
        <extensions>true</extensions>
      </plugin>
    </plugins>
  </build>

</project>
```

Sets packaging type as bundle

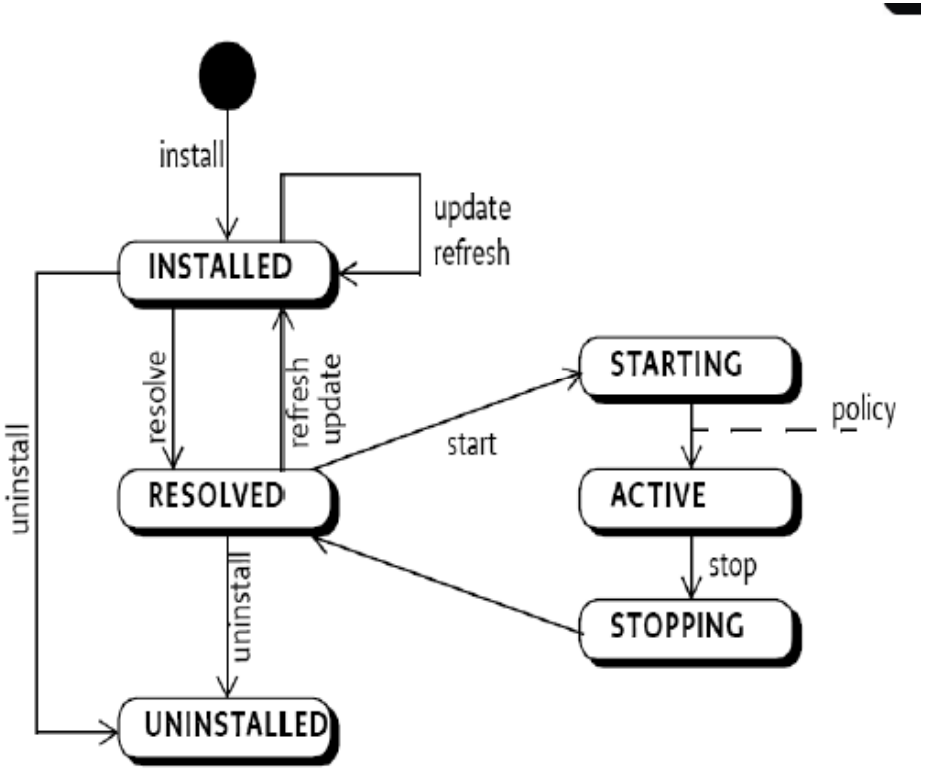
Declares Apache Felix Bundle Plugin

Tutorial : Create a Bundle using maven -
OSGI Bundle Using Maven - Karaf

Life Cycle Layer

- Provides an API to manage bundles at runtime.
- This API can be used to install, uninstall, update, start, stop bundles.
- Provides a runtime model for bundles.

Bundle States



Bundle States

- A bundle can be in one of the following states:
- INSTALLED The bundle has been – successfully installed.
- RESOLVED – All Java classes that the bundle needs are available. This state indicates that the bundle is either ready to be started or has stopped.
- STARTING – The bundle is being started, the BundleActivator.start method will be called, and this method has not yet returned.
- ACTIVE – The bundle has been successfully activated and is running; its Bundle Activator start method has been called and returned.
- STOPPING – The bundle is being stopped. The BundleActivator.stop method has been called but the stop method has not yet returned.
- UNINSTALLED – The bundle has been uninstalled. It cannot move into another state.

Tutorial: Lifecycle. - Bundle Lifecycle - Karaf

Service Layer

Specifies a mechanism for bundles to collaborate at runtime by sharing objects.

OSGi provides a in-VM publish-find-bind model for plain old Java objects(POJO).

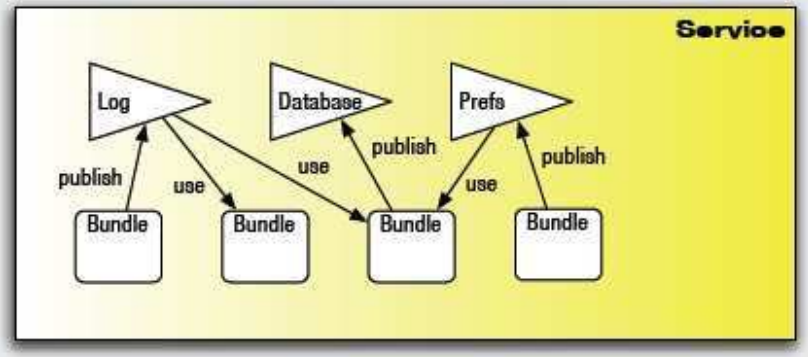
Service Layer

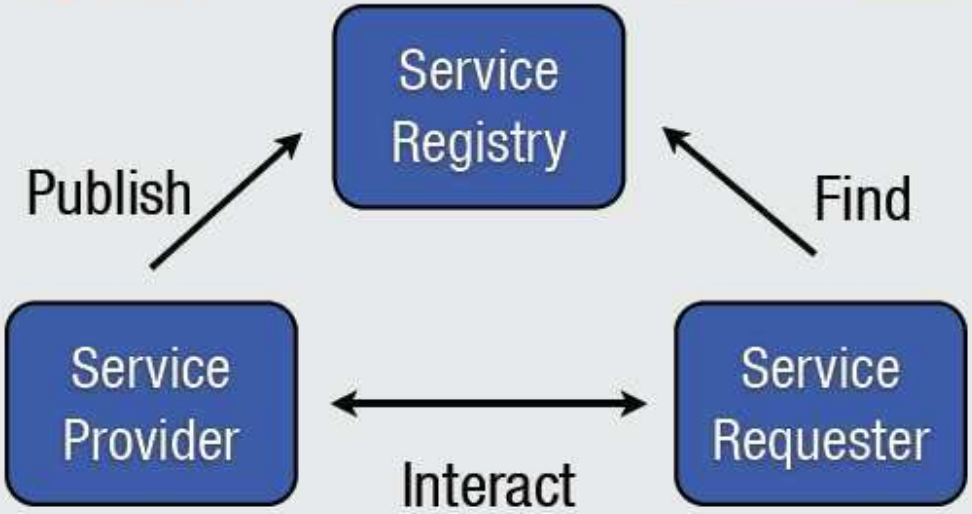
- Provides an in-VM service model
  - Discover (and get notified about) services based on their interface or properties
  - Bind to one or more services by
    - program control,
    - default rules, or
    - deployment configuration
- SOA Confusion
  - Web services bind and discover over the net

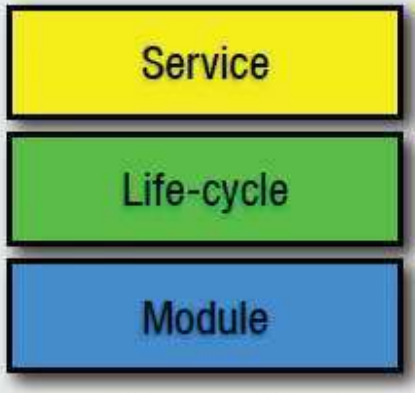
- The OSGi Service Platform binds and discovers inside a Java VM
- The OSGi Alliance provides many standardized services

Service Layer

- OSGi framework promotes service oriented interaction pattern among bundles

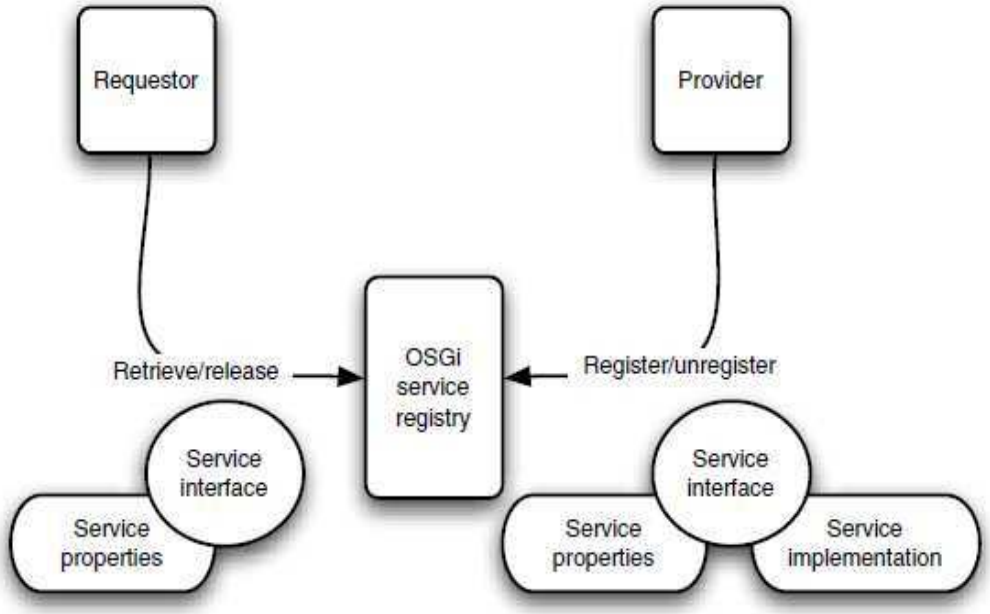






luminis

The OSGi service registry

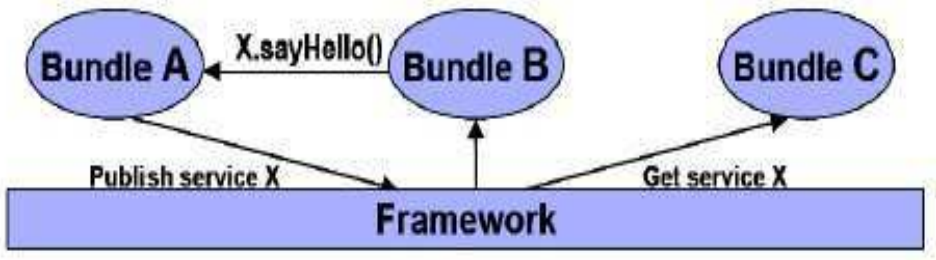


The service provider bundle registers the service interface, service properties, and service implementation into the OSGi service registry. The service requestor bundle retrieves the service interface and the service properties from the OSGi service registry.

The OSGi service registry acts as the mediator, isolating the provider from the requestor. The service provider bundle owns the service interface, service properties, and service implementation. The service requestor bundle uses the service interface and service properties, but it can't access the service implementation

Service Layer

- Introduces the OSGi service registry.
- A service is Java object published in the framework service registry.
- Bundles can register Java objects(services) with this service registry under one or more interfaces.
- A Java interface as the type of the service is strongly recommended.
- All these operations are dynamic.



Services

In OSGi, a service has three parts:

■ *An interface*—An OSGi service interface is a conjunction of Java class (`java.lang.Class`) names. These Java classes don't have to extend or implement any particular OSGi API, or follow any particular conventions, such as that of JavaBeans.

TIP Remember that `java.lang.Class` represents classes, interfaces, enums, and annotations.

■ *An implementation*—An OSGi service implementation is a Java object. This Java object must be an instance of all the classes whose names were used to define the service interface.

■ *Service properties*—A map of key-value pairs that can be used to provide metadata to the service is optional.

The OSGi service registry:

The OSGi service registry manages the OSGi services. The OSGi service registry has two main roles:

- It allows services to be registered and deregistered by the service provider bundle.

- It allows services to be retrieved and released by the service requestor bundle.

Registering a Service

A bundle publishing a service in the framework registry supplies.

- A string or string array, with fully qualified class name(s) that the service implements.
- Actual Java object (i.e service)
- A dictionary with additional service properties

Registering a Service

```
public class Activator implements BundleActivator {


 public void start(BundleContext bc) {
        Hashtable props = new Hashtable();  props.put("language", "en");
        //Registering the HelloWorld service
        bc.registerService(HelloService.class.getName(),  new   HelloServiceImpl(), props);
}
public void stop(BundleContext bc) {
}
}
```

Using a Service

1)Use framework to find a ServiceReference for the actual service. The ServiceReference,

- avoid unnecessary dynamic service dependencies between bundles.
- encapsulate the properties and other meta-date about the service object it represents.

2) Use ServiceReference to get the service object.

3)Cast the service object to appropriate Java type.

4)Use the service.

– If you do not need the service anymore, use ServiceReference to unget the service object.

Using a Service

```
public void start(BundleContext bc) {
//Get the service reference for HelloService
        serviceRef =
        bc.getServiceReference(HelloService.class.getName());
        //service reference can be null, if the service is not registered.
        if(serviceRef != null) {
        helloService = (HelloService)bc.getService(serviceRef);
} else {
        System.err.println("service reference not found.");
}
//service can be null..
if (helloService!=null) {
        helloService.sayHello
        ();
} else {
        System.err.println("No HelloService found!");
}
}
```

Using a Service

Once the bundle finished utilizing the service, It should release service using the following mechanism.

```
public void stop(BundleContext bc)
{  if (helloService!=null) {
        bc.ungetService(serviceRef);
        helloService =
        null;  serviceRef
        = null;
} else {
        System.err.println("HelloService is
        null!");
}
}
```

Using a Service

Once the bundle finished utilizing the service, It should release service using the following mechanism.

```
public void stop(BundleContext bc) {
  if (helloService!=null) {
                bc.ungetService(serviceRef);
                helloService = null;
                serviceRef = null;
} else {
                System.err.println("HelloService is null!");
}
}
```

Loading Native Code Libraries

Loading Native Code Libraries

- Dependency on native code is expressed in the Bundle-NativeCode header.
- The framework must verify this header and satisfy its dependencies before it attempts to resolve the bundle.

```
Bundle-NativeCode: lib/win32/x86/http.dll; lib/win/zlib.dll;
    processor=x86; osname=win32,
  lib/macosx/libhttp.dylib; lib/macosx/libzlib.jnilib;
    osname=macosx; processor=x86; processor=x86_64,
```

Native code declaration in a bundle's manifest:

- The following attributes are architected:
  1. osname
  2. osversion
  3. processor
  4. language
  5. selection-filter

Notes:
The following attributes are architected:

- osname – Name of the operating system. The value of this attribute must be the name of the operating system upon which the native libraries run.
- osversion – The operating system version. The value of this attribute must be a version range as defined in *Version Ranges*
- processor – The processor architecture. The value of this attribute must be the name of the processor architecture upon which the native libraries run.
- language – The ISO code for a language. The value of this attribute must be the name of the language for which the native libraries have been localized.
- selection-filter – A selection filter. The value of this attribute must be a filter expression that indicates if the native code clause should be selected or not.

Tutorial: Create a service example. - Services Using Karaf