Introduction to OSGi

The Dynamic Module System for Java

18th July, 2009

Sameera Jayasoma
Senior Software Engineer

WSO2
The open source SOA company

WSO2

The open source SOA company

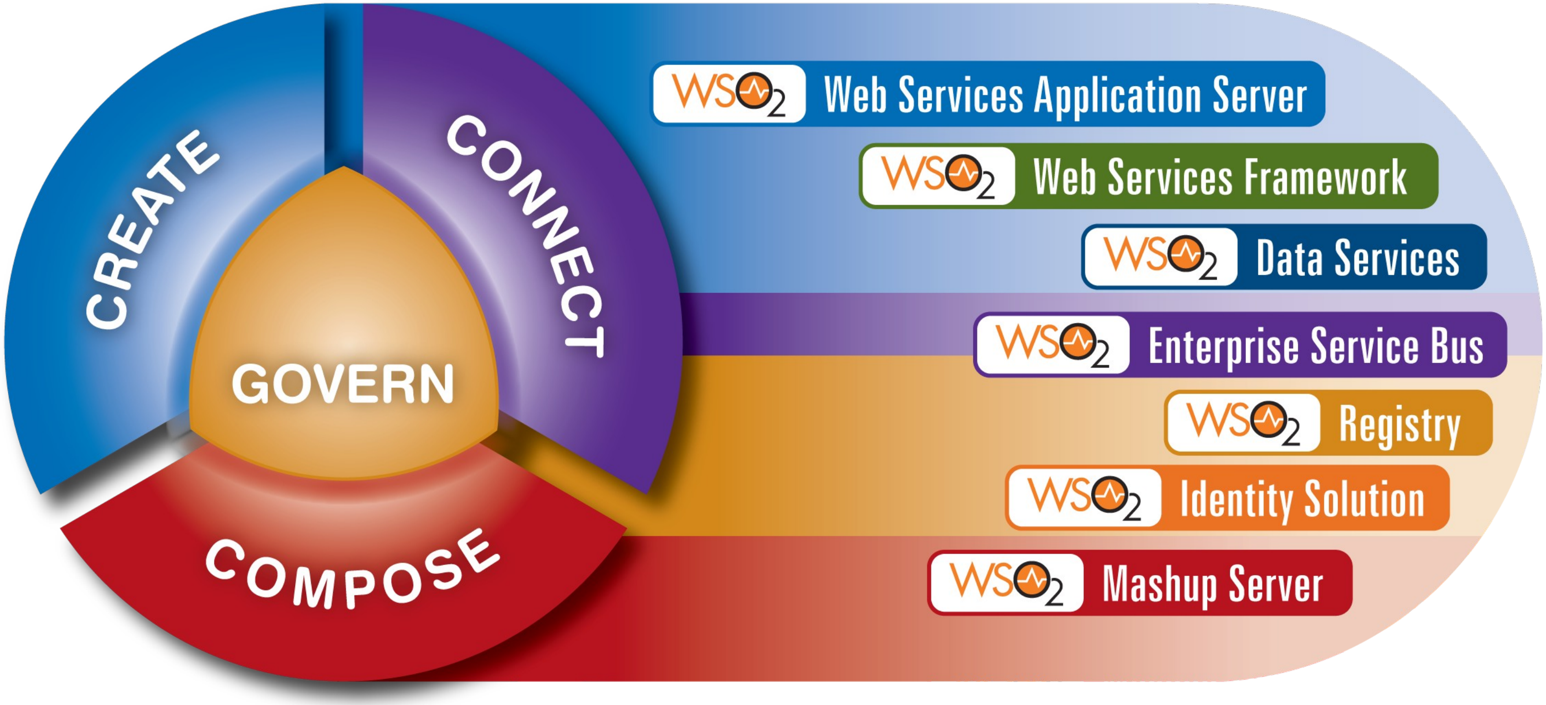
- Founded in 2005 by pioneers in XML and Web services technologies & standards as well as open source.
- Founders & leading contributors to all key Apache Web services projects.
- Offering complete SOA platform, 100% free and open source.
- Business model based on providing training, consultancy and support for the software.
- Global corporation with R&D center in Sri Lanka and offices in US & UK.

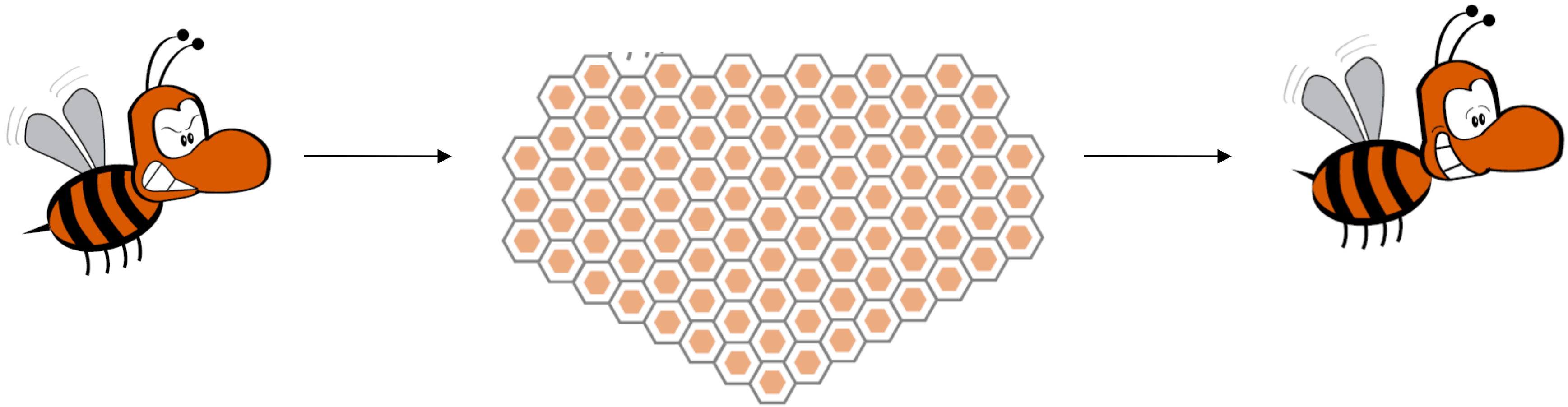
WSO2 Carbon

Middleware á la Carte

- Industry's only fully componetized SOA platform based on OSGi.
- A well defined component model for Enterprise SOA middleware
- The base platform for all WSO2' Java products
  - Web Services Application Server(WSAS)
  - Enterprise Service Bus(ESB)
  - Identity Server(IS)
  - Governance Registry(GReg)
- Offers unprecedented flexibility for developers to create customized SOA products with P2 based provisioning supporte
- Adapt middleware to your enterprise architecture, rather than adapt your enterprise architecture to middleware

WSO2 SOA Platform



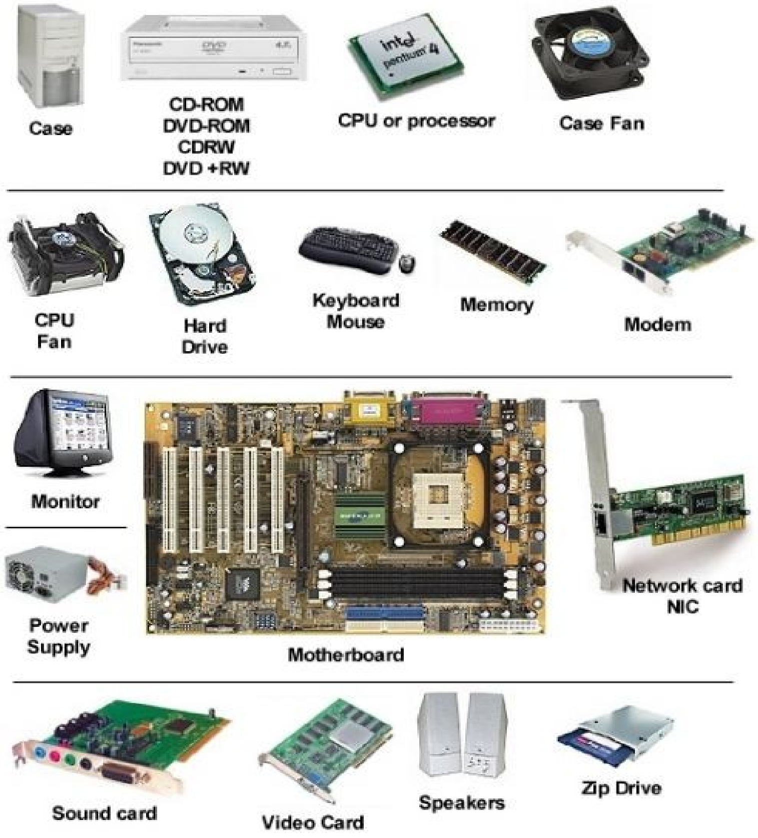


Modular Systems..

Modular Systems

- No doubt, a computer is complex system.
- How do yo 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(loose coupling)

Modular Systems

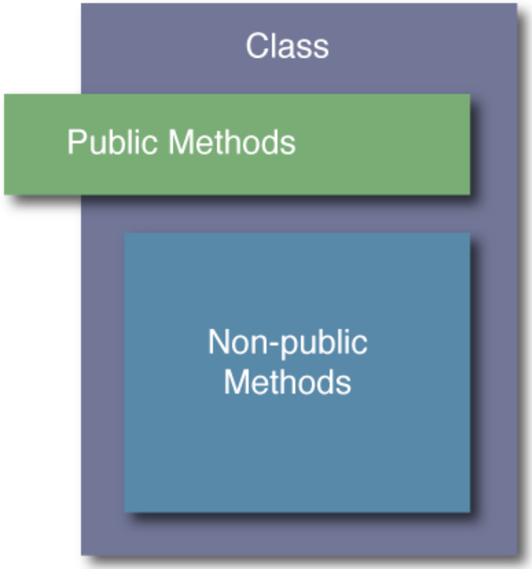
In the software world

- 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 building modular systems..

Java for Modular Systems

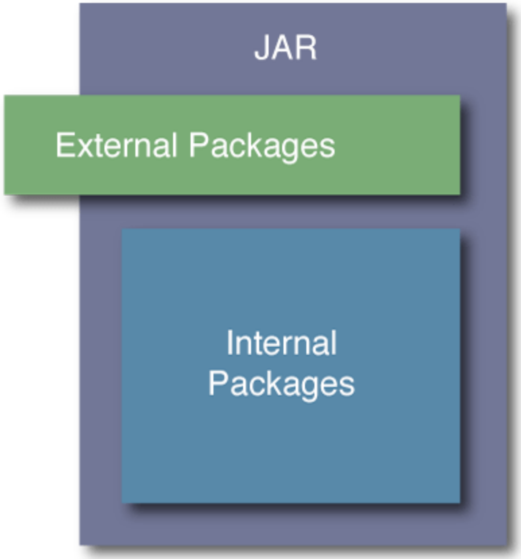
- Java is one of the popular OOP languages for developing large enterprise applications.
- Java provides some level of modularity.
- Consider a Java class
  - The unit of information hiding in Java.
  - Public methods, expose a defined contract
- Yet Java alone fails to develop better modular systems.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



Java for Modular Systems

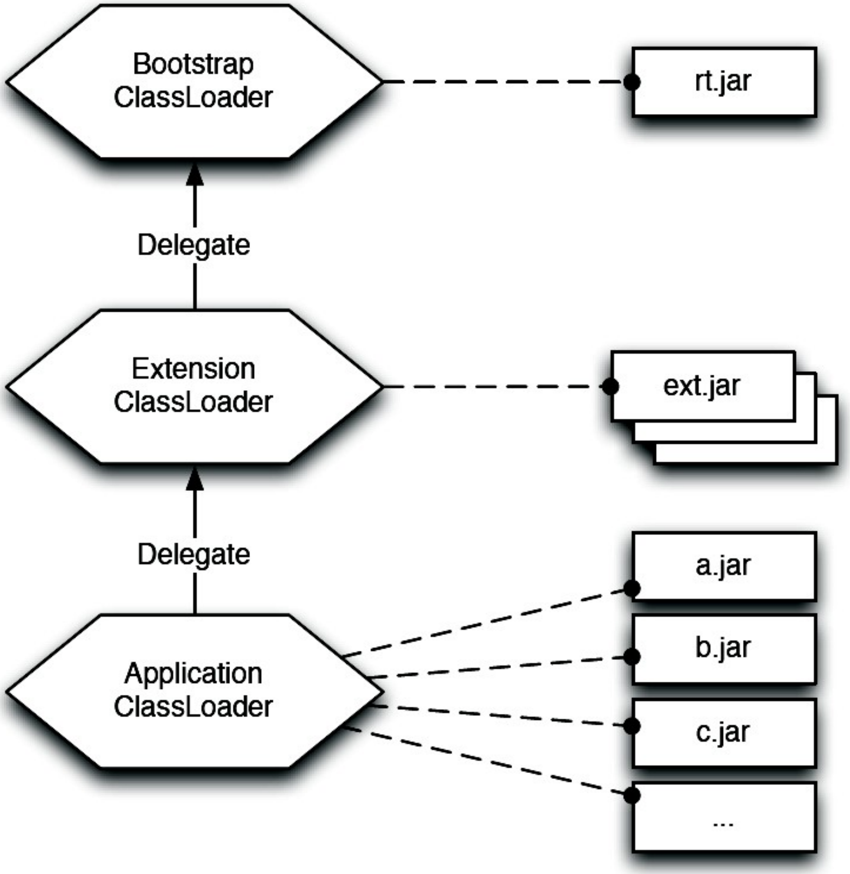
- What we need is something like this.
- A package should be the information hiding unit.
- It should be possible to,
  - Share a subset of packages from a Jar
  - Hide a subset of packages from a Jar

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



Class Loading

In standard Java application

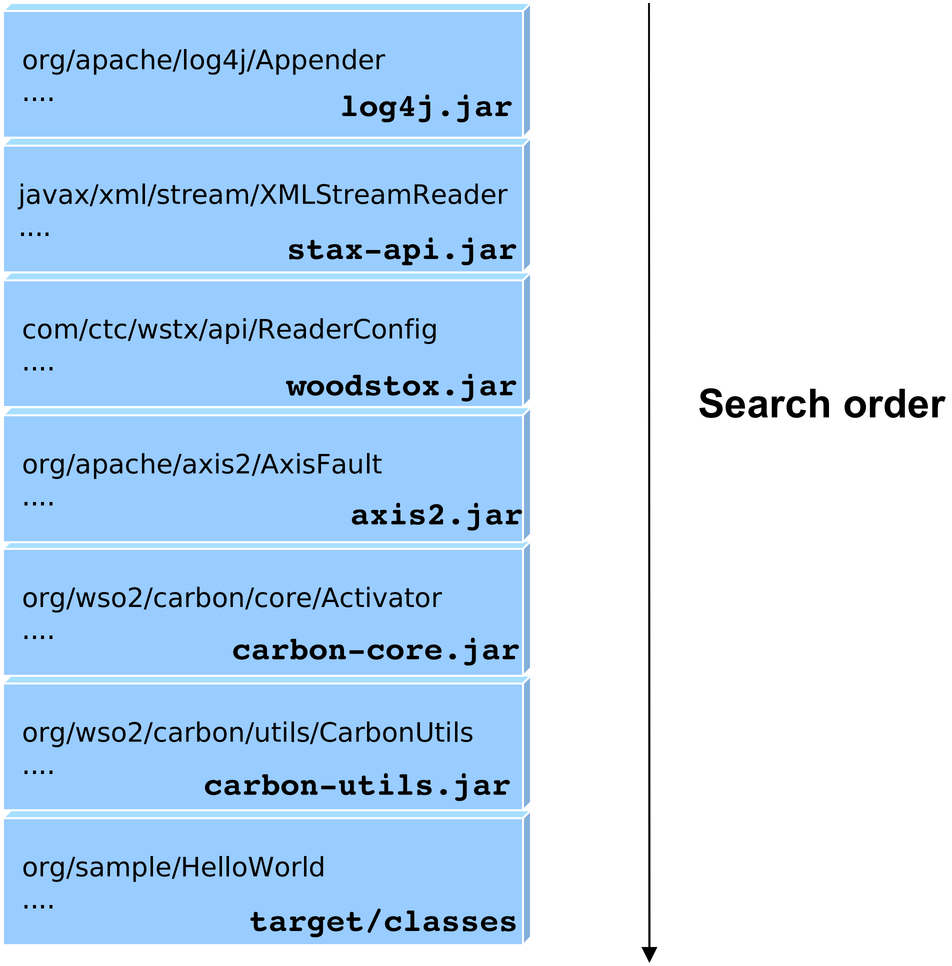


Problem with JARs

- 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:statx-api.jar:woodstox.jar:axis2.jar:
        carbon.jar:utils.jar:target/classes org.sample.HelloWorld
```

Problem with JARs



org/apache/log4j/Appender
....
log4j.jar

javax/xml/stream/XMLStreamReader
....
stax-api.jar

com/ctc/wstx/api/ReaderConfig
....
woodstox.jar

org/apache/axis2/AxisFault
....
axis2.jar

org/wso2/carbon/core/Activator
....
carbon-core.jar

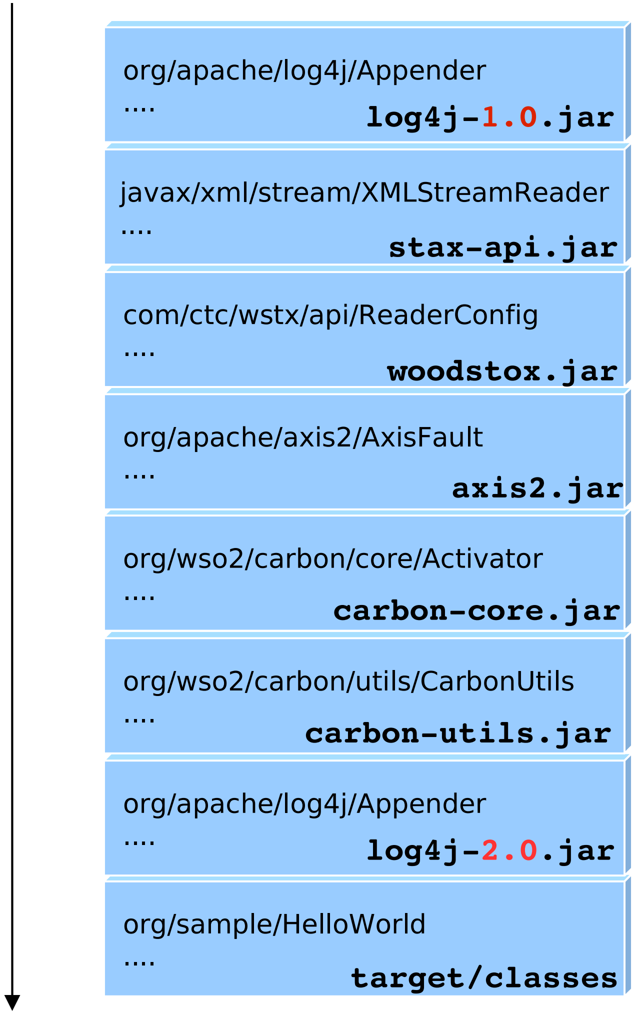
org/wso2/carbon/utils/CarbonUtils
....
carbon-utils.jar

org/sample/HelloWorld
....
target/classes

Search order

Problem with JARs

Problematic scenario



org/apache/log4j/Appender
....
log4j-1.0.jar

javax/xml/stream/XMLStreamReader
....
stax-api.jar

com/ctc/wstx/api/ReaderConfig
....
woodstox.jar

org/apache/axis2/AxisFault
....
axis2.jar

org/wso2/carbon/core/Activator
....
carbon-core.jar

org/wso2/carbon/utils/CarbonUtils
....
carbon-utils.jar

org/apache/log4j/Appender
....
log4j-2.0.jar

org/sample/HelloWorld
....
target/classes

Depends on log4j
2.0 version

- 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 simultaneosly
- 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

OSGi

The Dynamic Module System for Java

- 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.

Bundle

- The unit of modularization in OSGi.
- Standard Java application is a collection of Jars. In the same way OSGi based application can be considered as a collection of Bundle.
- A Java package is the unit of Information hiding.
- Bundles can share packages with other bundles and hide packages from other bundles
- Bundle is just a Jar file with some additional metadata(manifest headers) in the MANIFEST.MF file.

Bundle

- Sample MANIFEST.MF file of a bundle.

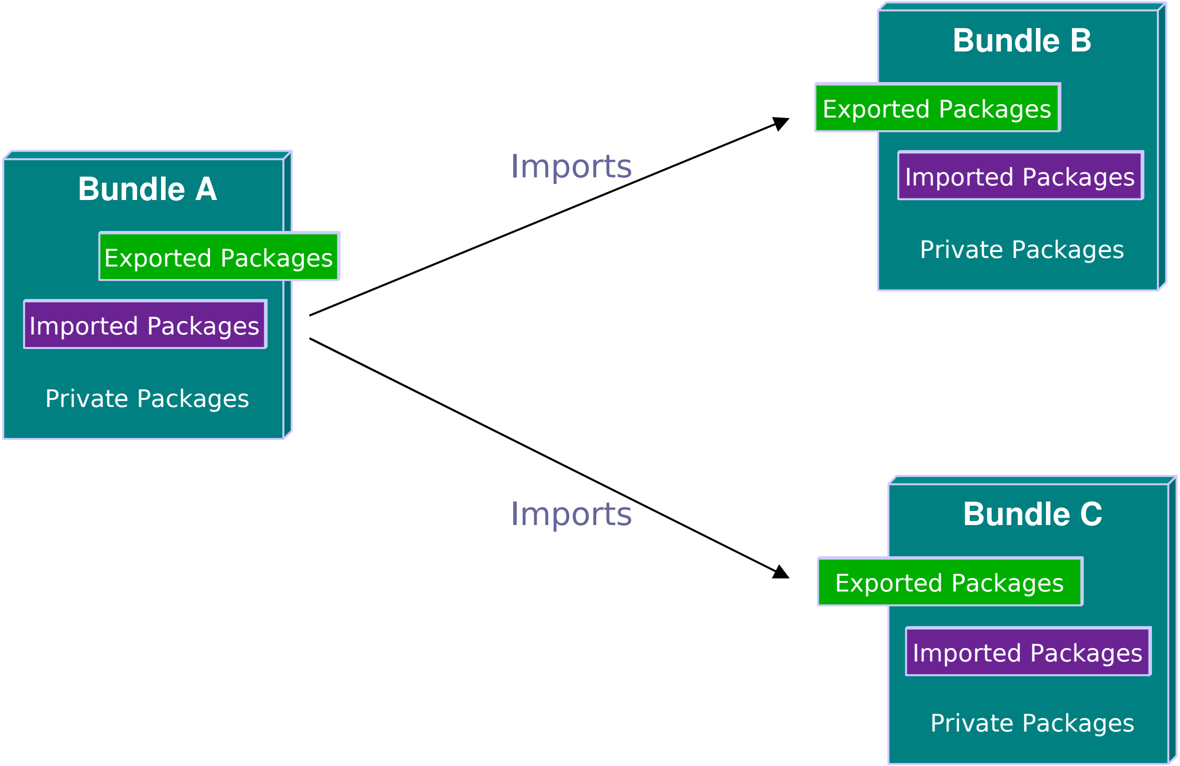
```
Bundle-ManifestVersion : 2
Bundle-Name : My First OSGi Bundle
Bundle-SymbolicName: HelloWorldBundle
Bundle-Version: 1.0.0
Export-Package:org.helloworld
Import-package:org.osgi.framework
```

- Bundle-SymbolicName and Bundle-Version is used to uniquely identify a bundle.
- Bundle-Version header is optional.

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

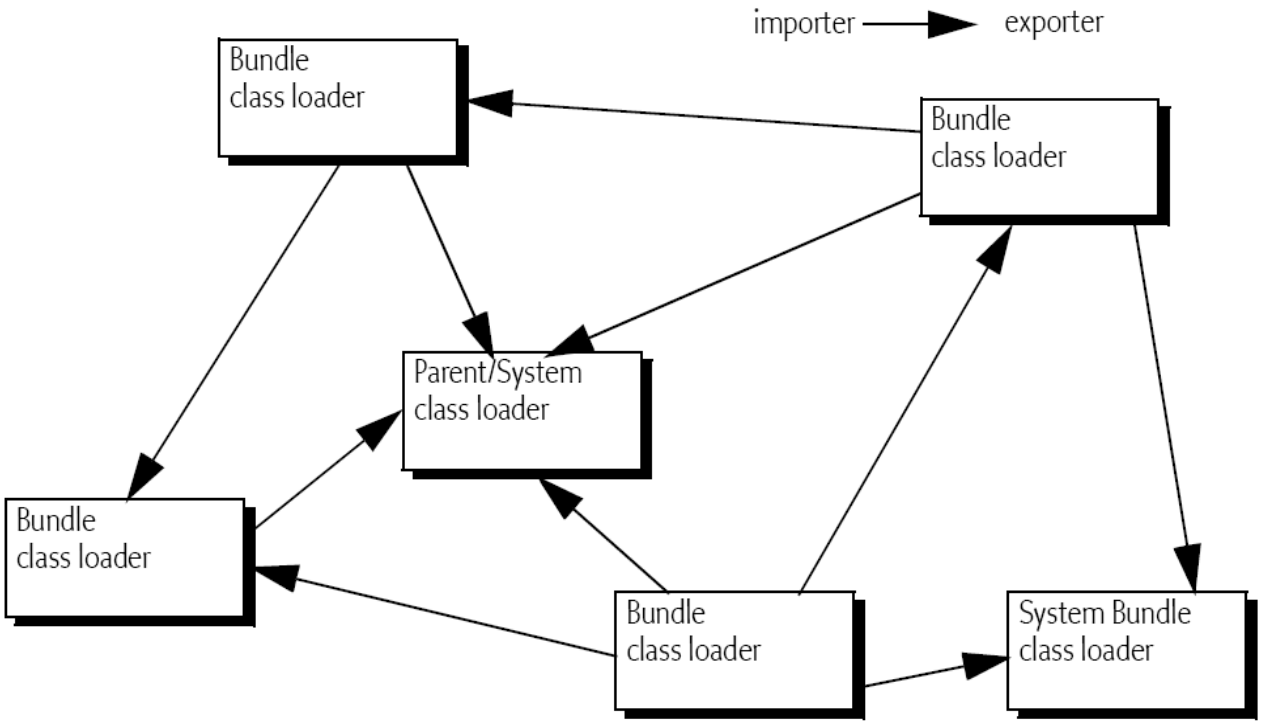


- How does OSGi achieve this level of information hiding...?

Bundles & Class Loaders

- Hierarchical class loading architecture in Java results in a global, flat and ordered class path.
- Root cause for most of the issues in Java
- OSGi eliminates these issues by introducing a separate class path for bundles. i.e a separate class loaders per bundle
- Bundle class loader can load classes from
  - system class loader
  - other bundle class loaders.(imported packages)
  - The bundle's jar file.
- This delegation forms a class loader delegation network.

Bundles & Class Loaders



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.

Demo

Require Bundles

- Mechanism where bundles can be directly wired to other bundles.

```
Require-Bundle : sample-api
```

- This header allows a bundle to import all exported packages from another bundle.
- Consider the following set of headers of the bundle sample-impl.

```
Bundle-SymbolicName: sample-impl
Require-Bundle: sample-api;visibility=reexport
```

- bundles that require this bundle(sample-impl) will transitively have access to the required bundle’s(sample-api) exported packages.
- The use of Require-Bundle is strongly discouraged. why?

Require Bundles

Issues with Require Bundles

- **Split Packages** – Classes from the same package can come from different bundles with Require bundle, such a package is called a split package.
- Say bundle A requires bundle B. What if bundle B changes over time?
    - Bundle B stops exporting certain packages on which bundle A depends on.
    - Bundle B is spited into several bundles.
- Require bundles chain can occur.
    - Bundle A requires bundle B.
    - Bundle B requires bundle C.
    - Bundle C requires bundle D and so on.
- Bundle A may depends on a small portion of bundle B. Yet Bundle A has to bring all the bundles in the chain.
- This can result in us bringing in a large amount of functionality when only a small amount is really required.

Fragment Bundles

- Fragments are bundles that are attached to a host bundle by the framework.
- Fragments are treated as part of the host, including any permitted headers.
- All class or resource loading of a fragment is handled through the host’s class loader, a fragment must never have its own class loader.
- Fragment-Host manifest header is used to specify the host bundle of the fragment bundle

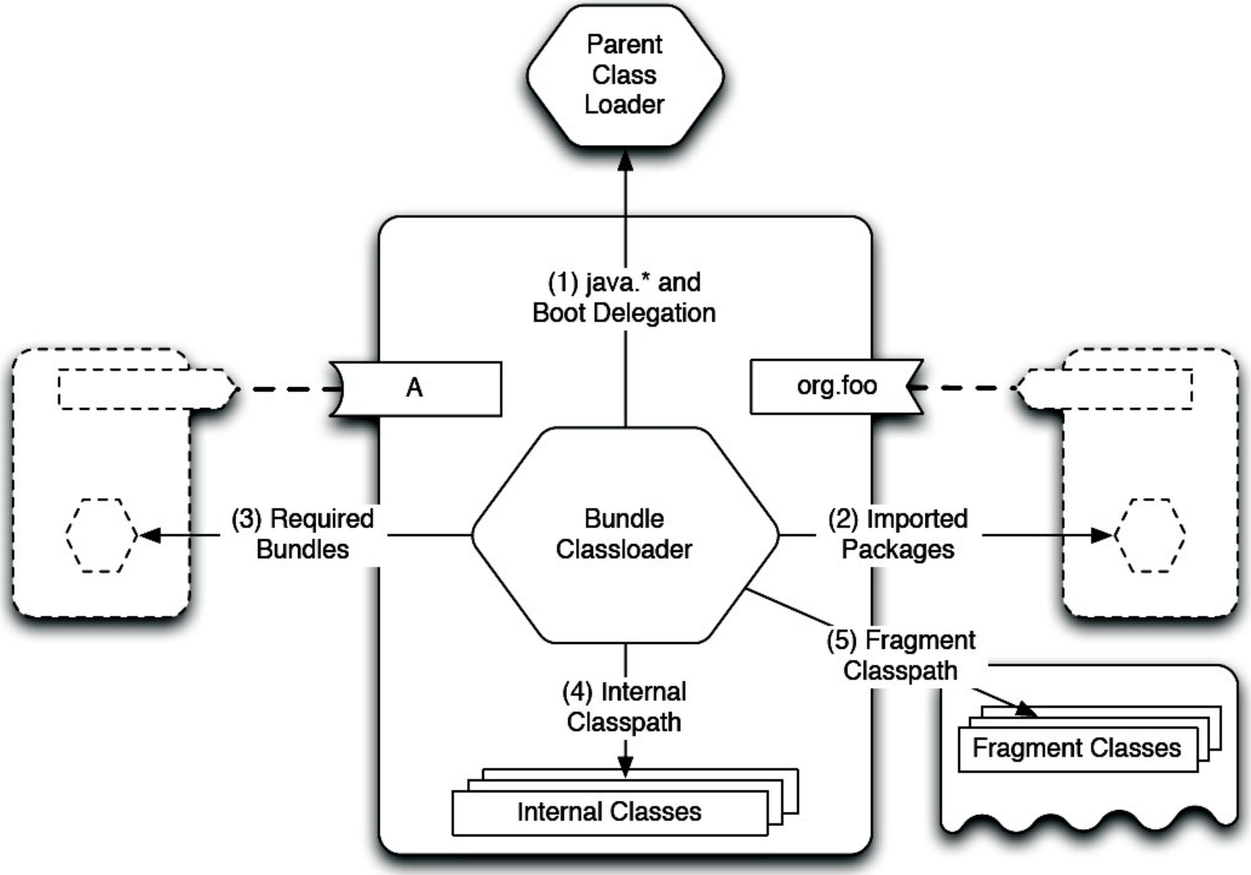
```
Bundle-SymbolicName: child-bundle
Fragment-Host: parent-bundle
```

Usage:

1) to provide translation files for different locales

2) to provide some platform specify code.

Runtime Class Loading

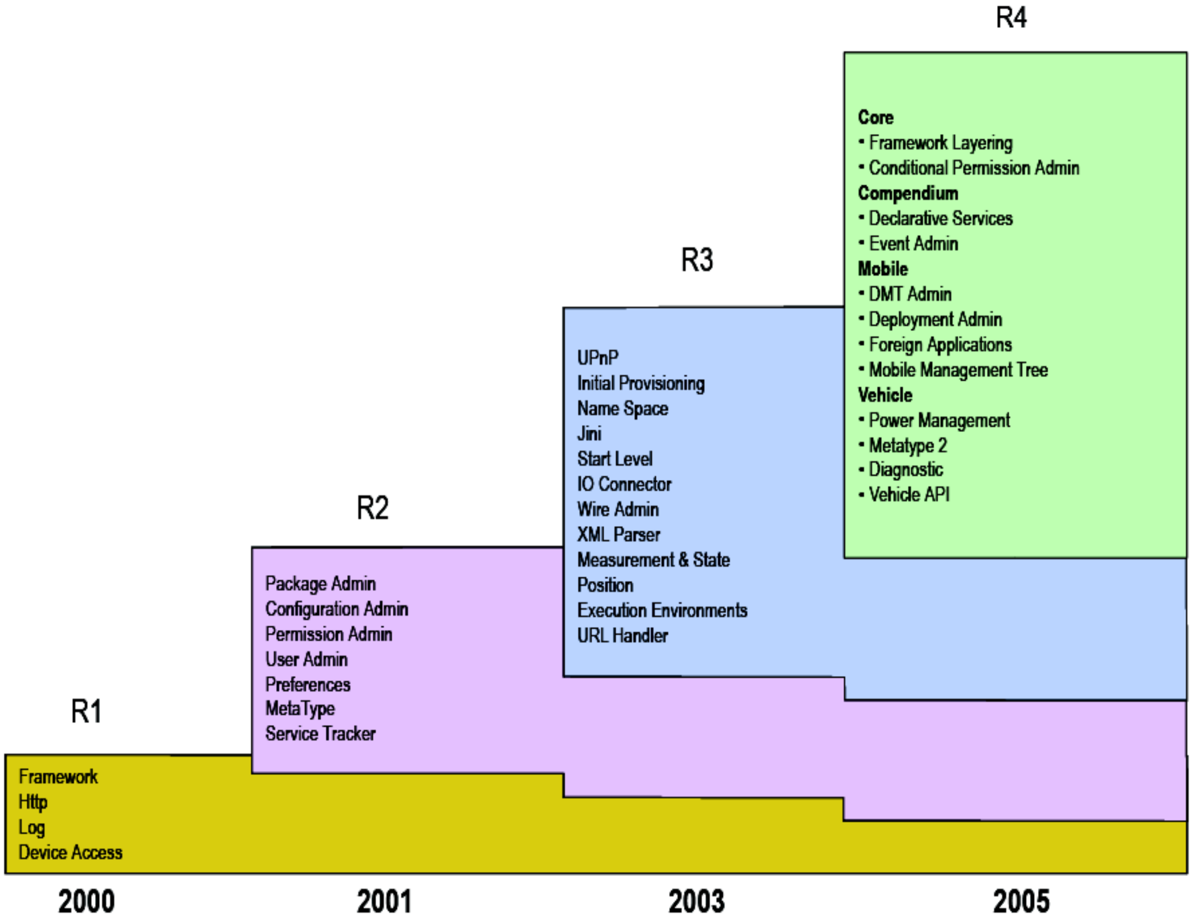


OSGi Specifications

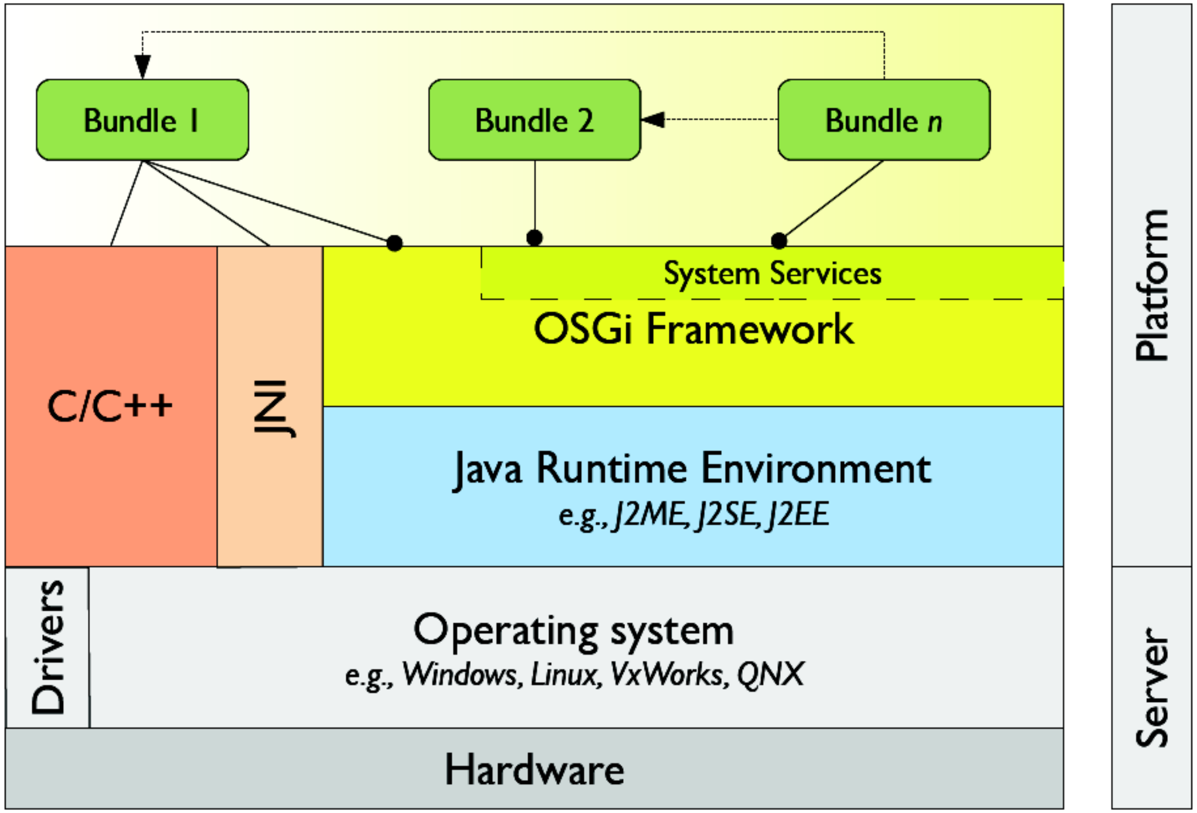
- Core specification
  - specifies the framework and the system services
- Service Compendium
  - specifies several OSGi services which are relevant for different markets such as vehicle and mobile.
- OSGi Alliance, a non profit organization.
  - develop OSGi specifications.
- Latest released version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

OSGi Specifications

Evolution and Contents

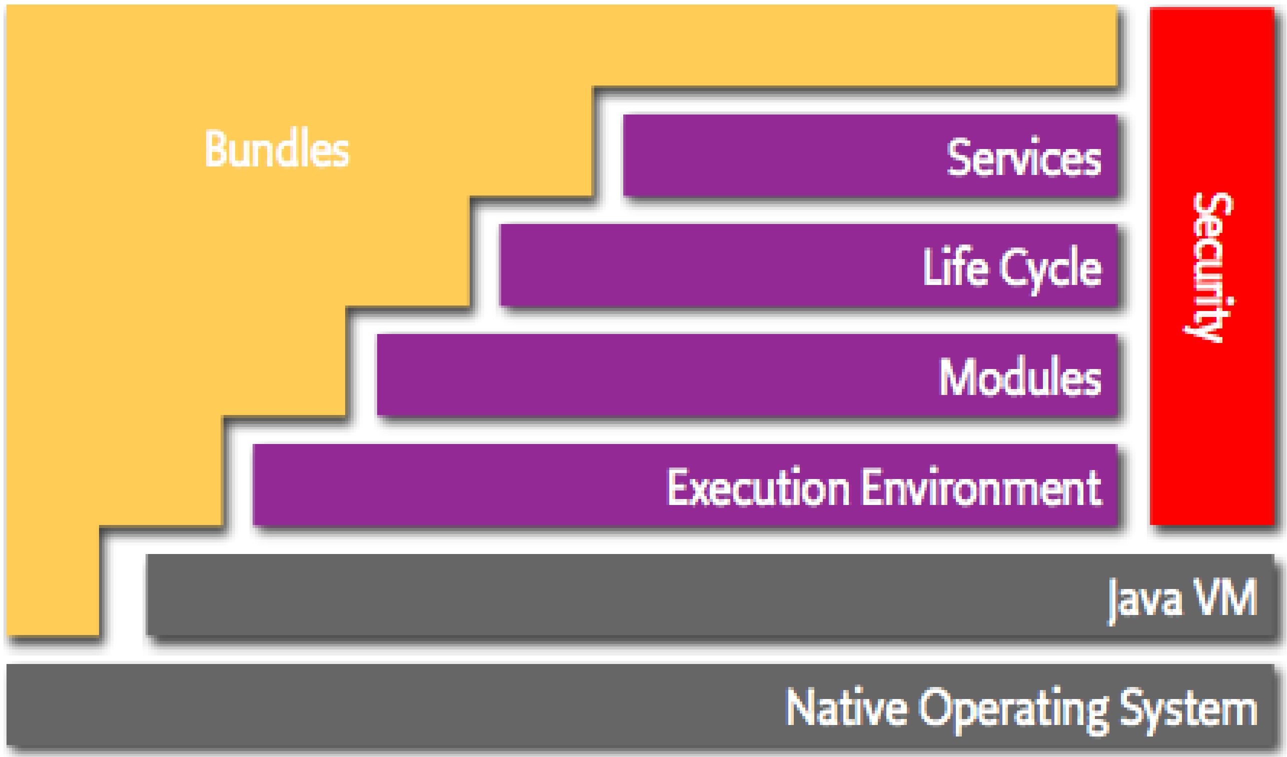


Java & OSGi


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



Life Cycle Layer

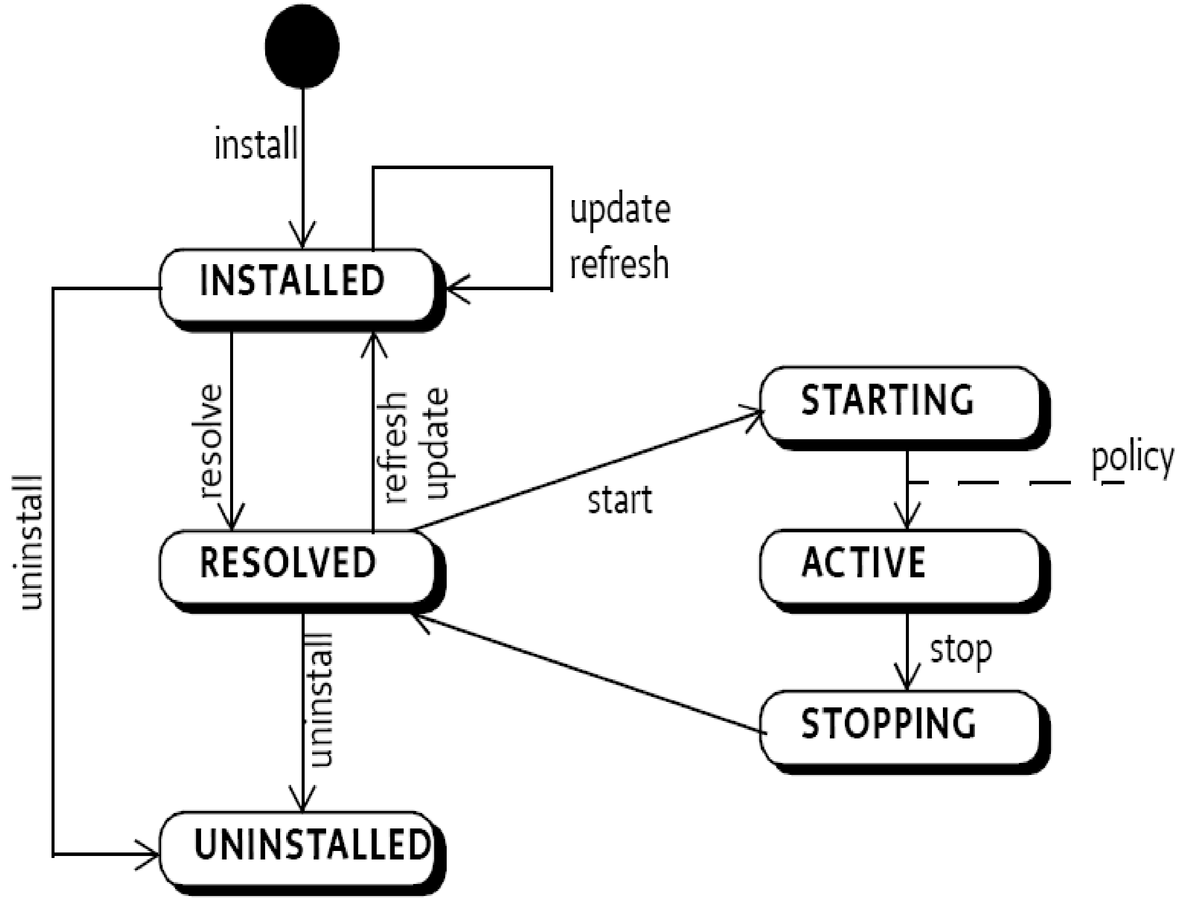
- Provides an API to manage bundles at runtime.
- This API can be used to install, uninstall, update, start, stop bundles.
- Provides a runtime model for bundles.

Bundle States

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.

Bundle States



Bundle Activator

- Bundle is activated by calling its Bundle Activator object(if any).
- BundleActivator interface defines methods that the framework invokes when it starts and stops the bundle.
- Bundle developer should declare Bundle-Activator manifest header in the manifest file, in order to inform the framework.
- The value of the header should be the fully qualified class name of the class which implements the BundleActivator interface

```
Bundle-Activator: org.sample.Activator

Public interface BundleActivator {
    public void start(BundleContext context) throws Exception;
    public void stop(BundleContext context) throws Excecption;
}
```

Bundle Context

- Represents the execution context of a single bundle within the OSGi platform.
- Act as a proxy between to the underlying framework.
- BundleContext object is created by the framework when a bundle is started.
- BundleContext object can be used to,
  - Install new bundles
  - Obtain registered services by other bundles,
  - Register services in the framework.
  - Subscribe or unsubscribe to events broadcast by the Framework

Demo

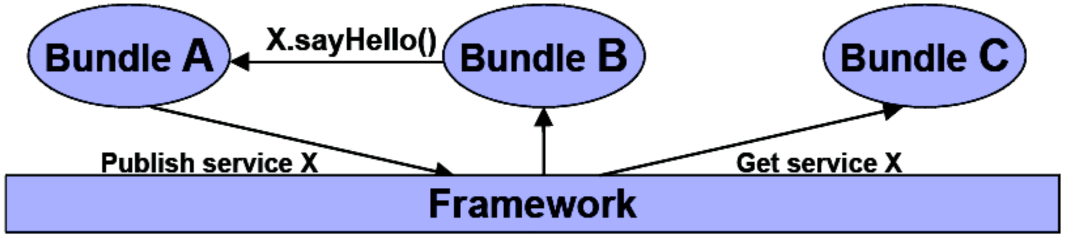
Service Layer

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

- 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.



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

- 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.

– 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!");
    }
}
```

Demo

Events and Listeners

- Framework fires ServiceEvents for following actions related to services.
  - Registering a service.
  - Unregistering a service.
  - Modifying service properties.
- It is highly recommended for bundles which uses services, to listen to these events and carefully handle them.
- Why?

Stale References

Services are Dynamic

Stale References

- Stale reference is reference to a Java object that
  - belongs to the class loader of a bundle that is stopped
  - is associated with a service object that is unregistered.
- Potential harmful because they may result in significantly increased memory usage.
- Removing stale references is a responsibility of the bundle developer.
- Bundles should listen to the **service events** of obtained services and act accordingly.

Services are Dynamic

- A service can come and go at any time.
- A bundle developer must not assume the availability of the service at any moment.
- Bundle can decide to withdraw its service from the registry while other bundles are still using this service.
- A Service may not be registered at the other bundles trying to use it.
  - this depends on the start order of bundles.
  - it is highly recommended not do depend on the starting order of bundles.
- Bundle developer should write code to handle this dynamic behavior of services.

Monitoring Services

- Monitoring services or listening to service events is the only way to handle dynamic behavior of services.
- Following mechanisms can be used for this purposes
  - Service Listeners
  - Service Trackers
  - Declarative Service
  - iPOJO
  - Blueprint services

Service Listener

- Introduced in R1 → from the beginning of OSGi
- ServiceListener is a listener interface that may be implemented by a bundle developer.

```
Public interface ServiceListener{
    public void serviceChanged(ServiceEvent event);
}
```

- When a ServiceEvent is fired, it is synchronously delivered to a ServiceListener.

Service Listener

```
public class HelloServiceListener implements ServiceListener{

    public void start(BundleContext context) throws Exception {
        context.addServiceListener(this);
    }

    public void stop(BundleContext context) throws Exception {
        context.removeServiceListener(this);
    }

    public void serviceChanged(ServiceEvent event) {
        switch(event.getType()){
            case ServiceEvent.UNREGISTERING:
            break;
            case ServiceEvent.REGISTERED:
            break;
            case ServiceEvent.MODIFIED:
            break;
        }
    }
}
```

Demo

If the service is registered before adding the listener, listener will not get the REGISTERED event.

Now what?

Service Listener

```
public class HelloServiceListener implements ServiceListener{

    public void start(BundleContext context) throws Exception {
        ref = context.getServiceReference(HelloService.class.getName());
        if(ref != null){
            helloService = (HelloService)context.getService(ref);
        }

        context.addServiceListener(this);
    }
}
```

- First try to get the service, then register the listener.
- We still have a problem here..
- Race conditions.

Service Tracker

- Introduced in R2 specification.
- Defines a utility class, ServiceTracker which significantly reduces the complexities of service listeners.
- ServiceTracker can be customized by implementing the interface ServiceTrackerCustomizer or by sub-classing the ServiceTracker class.
- Ideal solution for tracking one service.
- A better solution to remove the start level dependency.

Service Tracker

```
public class Activator implements BundleActivator {
    public void start(BundleContext bc) {

        tracker = new ServiceTracker(bundleContext,
                              HelloService.class.getName(), null );
        tracker.open();

        HelloService service = (HelloService) tracker.getService();
        if (service!=null) {
            service.sayHello(“Service Tracker”);
            service = null;
        }
    }

    public void stop(BundleContext bc) {
        tracker.close()
    }
}
```

Demo

Services are Dynamic

- Service listeners
  - race conditions
  - listener leaks.
- Service Trackers
  - must be closed otherwise listener leaks occur.
  - Writing a customizer to handle more than one service is complicated.
- Working with the OSGi service model using the programmatic API can be complex and error prone.
- Bundle developers tend to make optimistic assumptions regarding the availability of services in an attempt to simplify their code.

Declarative Services

Declarative Services

- Introduced in R4.
- Alternative approach for using OSGi services programming API.
- Is a way for a bundle to declare, in an XML file, the services it registers and acquires.
- Provides a simplified programming model for developers. They need to write less code.
- Runtime portion of the declarative service is called Service Component Runtime(SCR).
- Allows developers to keep OSGi code away from domain logic.

Services Component

- A service component contains a description that is interpreted at run time to create and dispose objects depending on the
  - availability of other services
  - need for such an object
  - available configuration data.
- Can optionally provide as OSGi service.
- DS specification uses the generic term component to refer to a service component
- Component is a normal Java class(POJO) and it is declared in an XML document.

Concepts

Component Description – The declaration of a service component. It is contained within an XML document in a bundle.

Component Properties – A set of properties which can be specified by the component description, Configuration Admin service and from the component factory.

Component Configuration – A component configuration represents a component description parameterized by component properties. It is the entity that tracks the component dependencies and manages a component instance. An activated component configuration has a component context.

Component Instance – An instance of the component implementation class. A component instance is created when a component configuration is activated and discarded when the component configuration is deactivated. A component instance is associated with exactly one component configuration.

Declaring a Service

A component requires the following artifacts in the bundle:

1) An XML document that contains the component description.

```
/OSGI-INF/example.xml
```

2) The Service-Component manifest header which names the XML documents that contain the component descriptions.

```
Service-Component: OSGI-INF/example.xml
```

3) An implementation class that is specified in the component description.

Example 1

Description of a component which reference a service.

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="helloservice.listen">
    <implementation class="org.sample.HelloComponent”/>
    <reference name="HS"
            interface="org.helloworld.HelloService"
            bind="setHelloService"
            unbind="unsetHelloService” />
</component>
```

Example 1

Component implementation class.

```
public class HelloComponent {
    HelloService hs;
    protected void setHelloService(HelloService s) { hs = s; }
    protected void setHelloService(HelloService s) { hs = null; }
    protected void activate(ComponentContext ctxt) {...}
    protected void deactivate(ComponentContext ctxt) {...}
}
```

Example 2

Description of a component which publish a service.

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="example.handler">
    <implementation class="org.helloworld.HelloServiceImpl"/>
    <service>
        <provide interface="org.helloworld.HelloService"/>
    </service>
<component>
```

Demo

SCR

Service Component Runtime

- The actor/implementation that manages the components and their life cycle.
- Listens for bundles that become active(Active state) and detects Component Descriptions
- The SCR is responsible for activating and deactivating Component Configurations

Component Life Cycle

Enabled

- Life cycle of a component contained within the life cycle of its bundle.
- Initial enabled state of a component is specified in the component description, using the enabled attribute.
- A component is enabled if the bundle is started and the enabled attribute is set to true. The default value is “true”.
- A component should become enabled before it can be used.

Component Life Cycle

Satisfied

- A component can become satisfied, if the following conditions are met
- The component is enabled.
- Using the component properties of the component configuration, all the component’s references are satisfied. A reference is satisfied when the reference specifies optional cardinality or there is at least one target service for the reference.

Activation and Deactivation

- SCR must activate a component configuration when the component is enabled and the component configuration is satisfied and a component configuration is needed. During the life time of a component configuration, SCR can notify the component of changes in its bound references.
- SCR will deactivate a previously activated component configuration when the component becomes disabled, the component configuration becomes unsatisfied, or the component configuration is no longer needed.

Types of Components

- **Delayed Component**
  - A component whose component configurations are activated when their service is requested.
- **Immediate Component**
  - A component whose component configurations are activated immediately upon becoming satisfied.
- **Factory Component**
  - A component whose component configurations are created and activated through the component’s component factory.

Immediate Component

- A component is an immediate component if it is not a factory component and either does not specify a service or specifies a service and the immediate attribute of the component element set to true.
- An immediate component is activated as soon as its dependencies are satisfied.
- If an immediate component has no dependencies, it is activated immediately.

Immediate Component

Component description

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="example.activator">
    <implementation class="org.sample.HelloComponent"/>
</component>
```

Component implementation class

```
public class HelloComponent {
    protected void activate(ComponentContext ctxt) {...}
    protected void deactivate(ComponentContext ctxt) {...}
}
```

Delayed Component

- A delayed component
  - specifies a service
  - is not specified to be a factory component
  - does not have the immediate attribute of the component element set to true.
- If a delayed component configuration is satisfied, SCR must register the component configuration as a service in the service registry but the activation of the component configuration is delayed until the registered service is requested.
- This is achieved by using a ServiceFactory

Delayed Component

Component description

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="example.handler">
    <implementation class="org.helloworld.HelloServiceImpl"/>
    <service>
        <provide interface="org.helloworld.HelloService"/>
    </service>
<component>
```

Component implementation class

```
public class HelloServiceImpl implements HelloService {
    public void sayHello() {...}
}
```

Accessing Services

Event Strategy

Component description.

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="helloservice.listen">
    <implementation class="org.sample.HelloComponent”/>
    <reference name="HS"
            interface="org.helloworld.HelloService"
            bind="setHelloService"
            unbind="unsetHelloService” />
</component>
```

Component implementation class.

```
    public class HelloComponent {
        HelloService hs;
        protected void setHelloService(HelloService s) { hs = s; }
        protected void setHelloService(HelloService s) { hs = null; }
        protected void activate(ComponentContext ctxt) {...}
        protected void deactivate(ComponentContext ctxt) {...}
    }
```

Accessing Services

Lookup Strategy

Component description.

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="helloservice.listen">
    <implementation class="org.sample.HelloComponent”/>
    <reference name="HS"
            interface="org.helloworld.HelloService"/>
</component>
```

Component implementation class.

```
public class HelloComponent {
    HelloService hs;
    protected void activate(ComponentContext ctxt) {
        hs = (HelloService) cxtx.locateService(“HS”);
    }
    protected void deactivate(ComponentContext ctxt) {...}
}
```

References to Services

- cardinality for a referenced service
  - 0..1 – optional and unary,
  - 1..1 – mandatory and unary (Default) ,
  - 0..n – optional and multiple,
  - 1..n – mandatory and multiple.
- Reference policy
  - static
  - dynamic
- selecting target services
  - By specifying a filter in the target property, the set of services that should be part of the target services can be constrained

References to Services

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="helloservice.listen">
    <implementation class="org.sample.HelloComponent"/>
    <reference name="HS"
        interface="org.sample.HelloService"
        cardinality="0..n"
        policy="dynamic"
        target="(language=en)"
        bind="setHelloService"
        unbind="setHelloService" />
</component>
```

Demo

Questions ??

Thank you