Microkernel Architecture (Plugin-architecture Pattern)

Software Architecture
3rd Year – Semester 1
By Udara Samaratunge

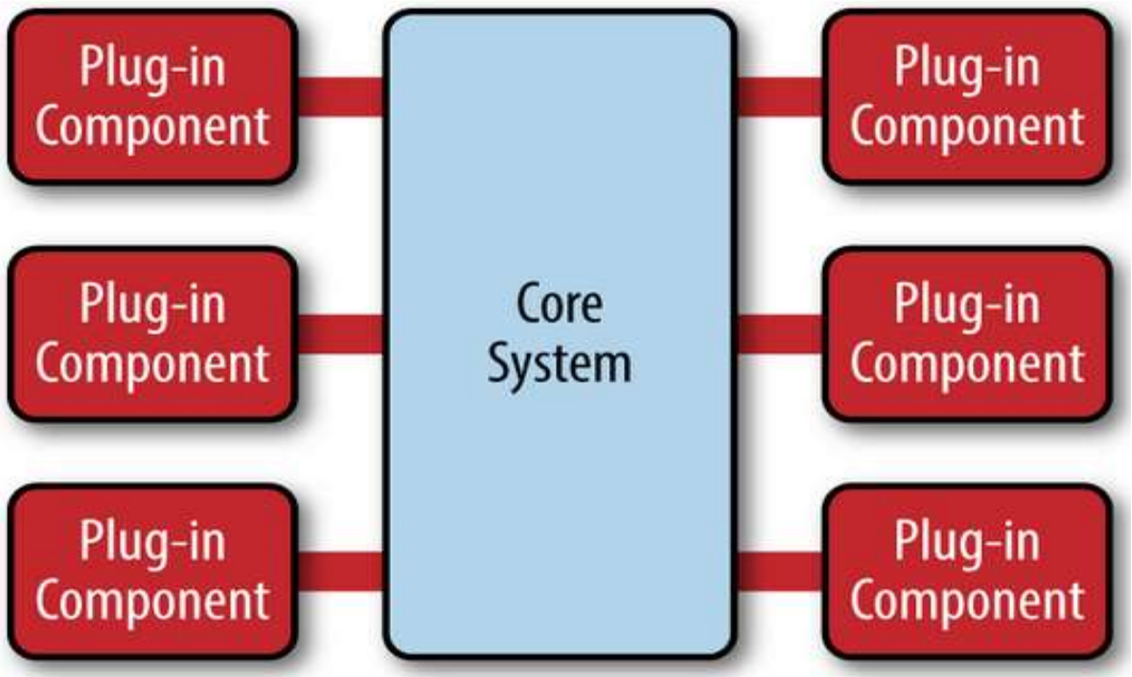
Plugin-architecture Pattern

- Two Components
  - Core System
  - Plugin module

- The plug-in modules are stand-alone, independent components.

- Core system needs to know about which plug-in modules are available (uses Plugin registry)

- OSGi represents Microkernel Architecture for its plugin development.



What is OSGi?

- OSGi (Open Services Gateway initiative)
- OSGi is a **framework** which allows modular development of applications using java.
- A Java framework for developing (remotely) deployed service applications, that require:
  - Reliability
  - Large scale distribution
  - Wide range of devices
  - Collaborative
- Created through a collaboration of industry leaders
  - IBM, Ericsson, Nokia, Sony, Telcordia, Samsung, ProSyst,
  - Gatespace, BenQ, Nortel, Oracle, Sybase, Espial, and many more

- OSGi containers allow you to break your application into individual modules. (are jar files with additional meta information and called bundles in OSGi terminology)

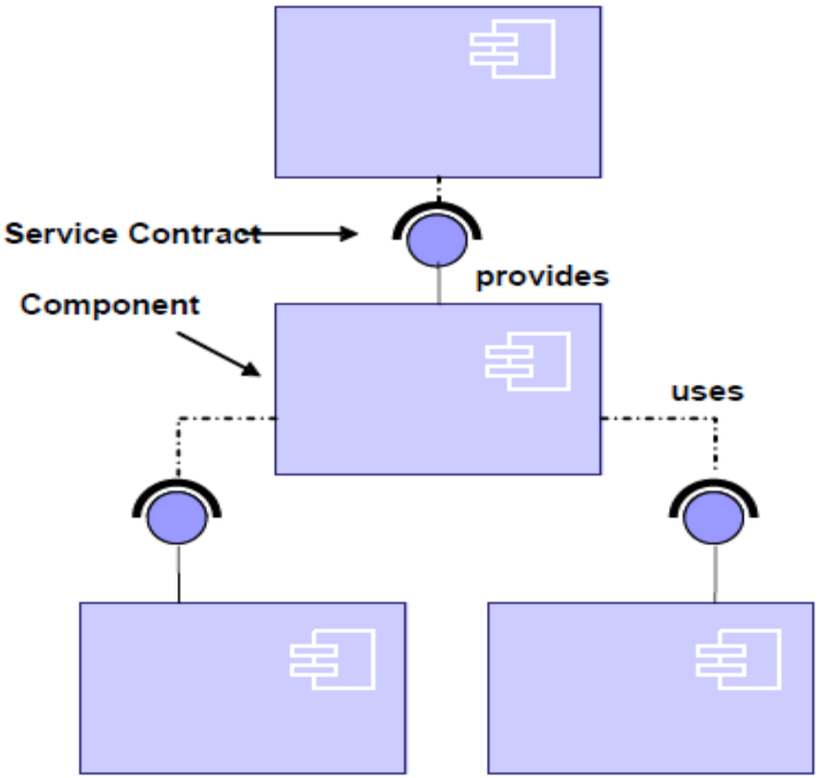
- Manage the cross-dependencies between modules.

- An OSGi framework then offers you dynamic loading/unloading, configuration and control of these bundles - without requiring restarts.

- Major Framework vendors are
  - ProSyst,
  - Gate space Telematics, and
  - IBM
  - Siemens
  - Espial
- Open source implementations
  - Apache Felix
  - Eclipse Equinox
  - Gate space Knopflerfish

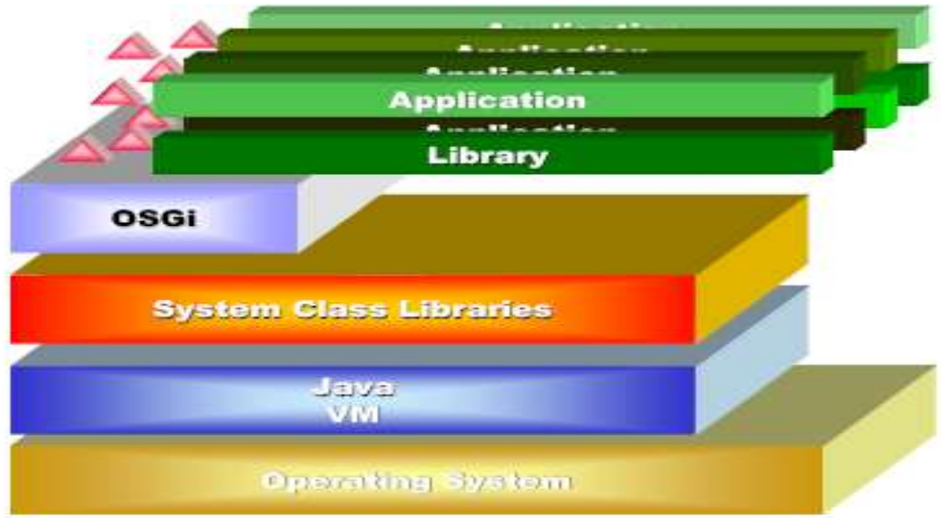
The OSGi Platform vs Service Oriented Architectures (SOA)

- Separate the contract from the implementation
- Allows alternate implementations
- Dynamically discover and bind available implementations
- Binding based on contract (interface definitions)
- Components are reusable
- Components are not coupled to implementation details of other components, only their independent interfaces have to be known



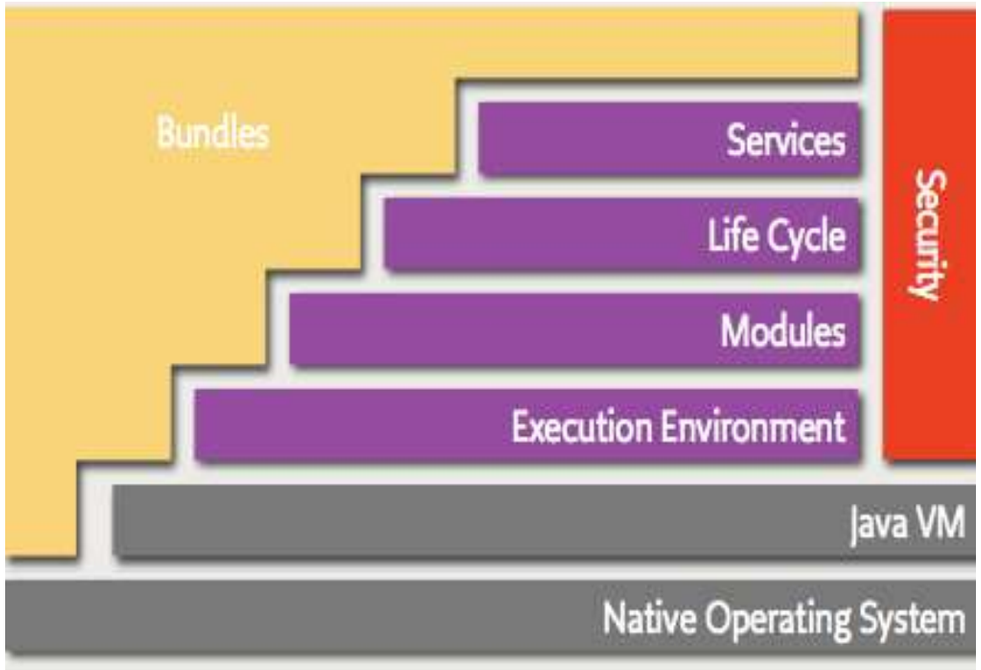
The OSGi Framework Architecture

- Allows applications to share a single Java VM.
- Handles all class loading in a much better defined way than standard Java.
  - Versioning!
- Gives isolation/security between applications.



- Mediates between communication & collaborations between applications.
- Provides life cycle management (install, start, stop, update, etc).
- Policy free
  - Policies are provided by bundles

OSGi Bundle Architecture



- The following list contains a short definition of the terms:
  - **Bundles** - Bundles are the OSGi components made by the developers.
  - **Services** - The services layer connects bundles in a dynamic way by offering a *publish-find-bind model* for *plain old Java objects*.
  - **Life-Cycle** - The API to install, start, stop, update, and uninstall bundles.
  - **Modules** - The layer that defines how a bundle can import and export code.
  - **Security** - The layer that handles the security aspects.
  - **Execution Environment** - Defines what methods and classes are available in a specific platform.

What is a Bundle?

- In Java terms, a bundle is a plain old JAR file.
- In standard Java everything in a JAR is completely visible to all other JARs.
- But OSGi hides everything in that JAR unless explicitly exported.
- **Reason for hiding** is to maintain multiple versions of the same library.
- By default, there is no sharing.

Practical Example

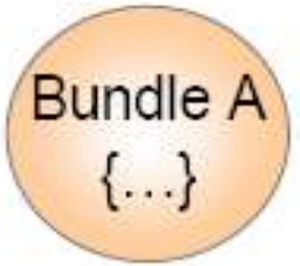
Eg :- I have an application that is interacting with underlying MySQL database and after few month I found that MySQL team has fixed a major bug in their new version of mysql-connector library release so in order to incorporate this new library in my traditional application I have to **stop my application** and **re-package** it (**or just replace the older one**)

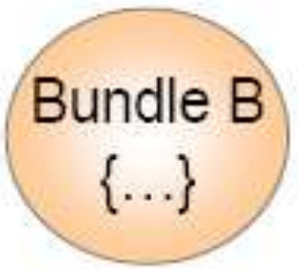
But, with OSGI we **don't need to stop the whole application** because **everything is exposed** either as a component or as a service therefore we **just need to install new component/service in OSGI container**.

when the services/components are updated in OSGI container there are various event listeners that propagate the service/component update event to service/component consumers.

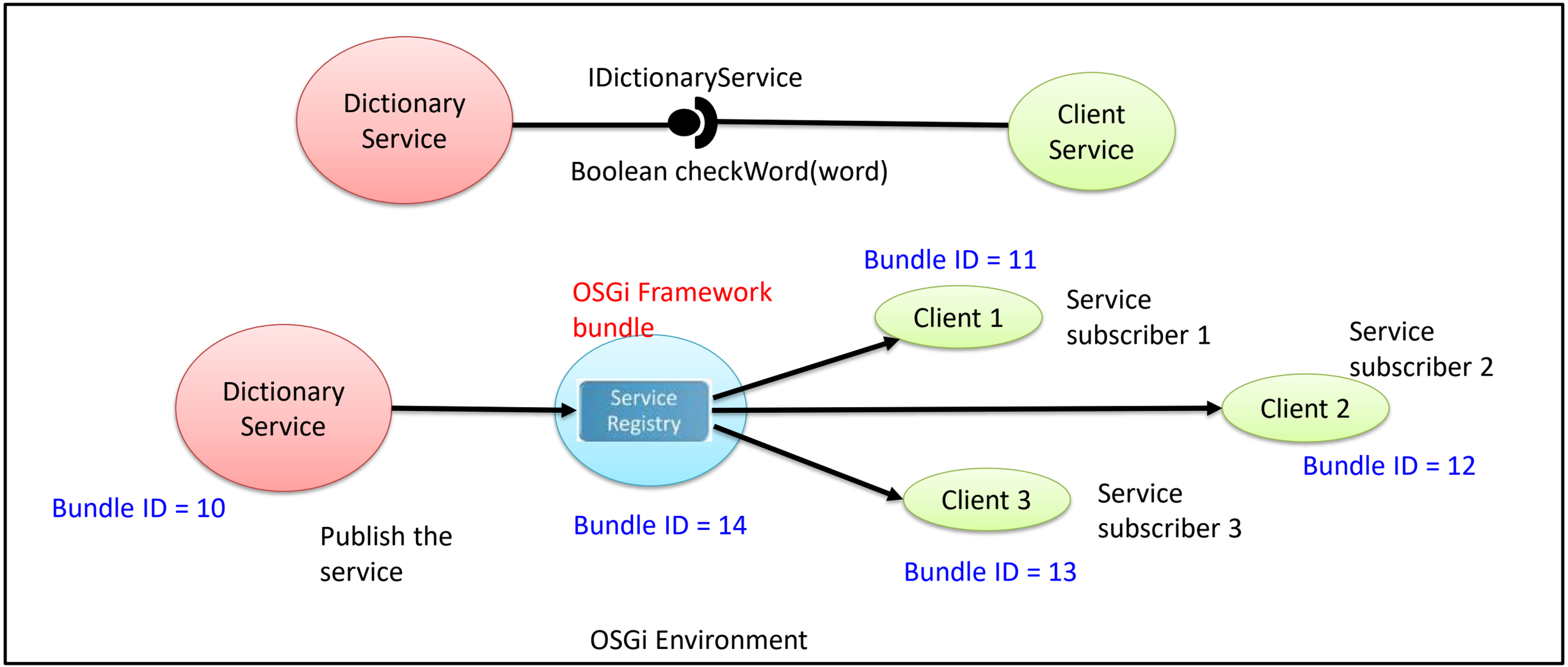
And accordingly consumers adapts themselves to use new version of web service (on the consumer side we need to listen for various events so that consumers can decide whether to respond for change or not.

- A *bundle* is the deliverable application
  - Like a Windows EXE file
  - Content is a JAR file
- A bundle registers zero or more services
  - A service is specified in a Java interface and may be implemented by multiple bundles
  - Services are bound to the bundle life-cycle
- Searches can be used to find services registered by other bundles
  - Query language

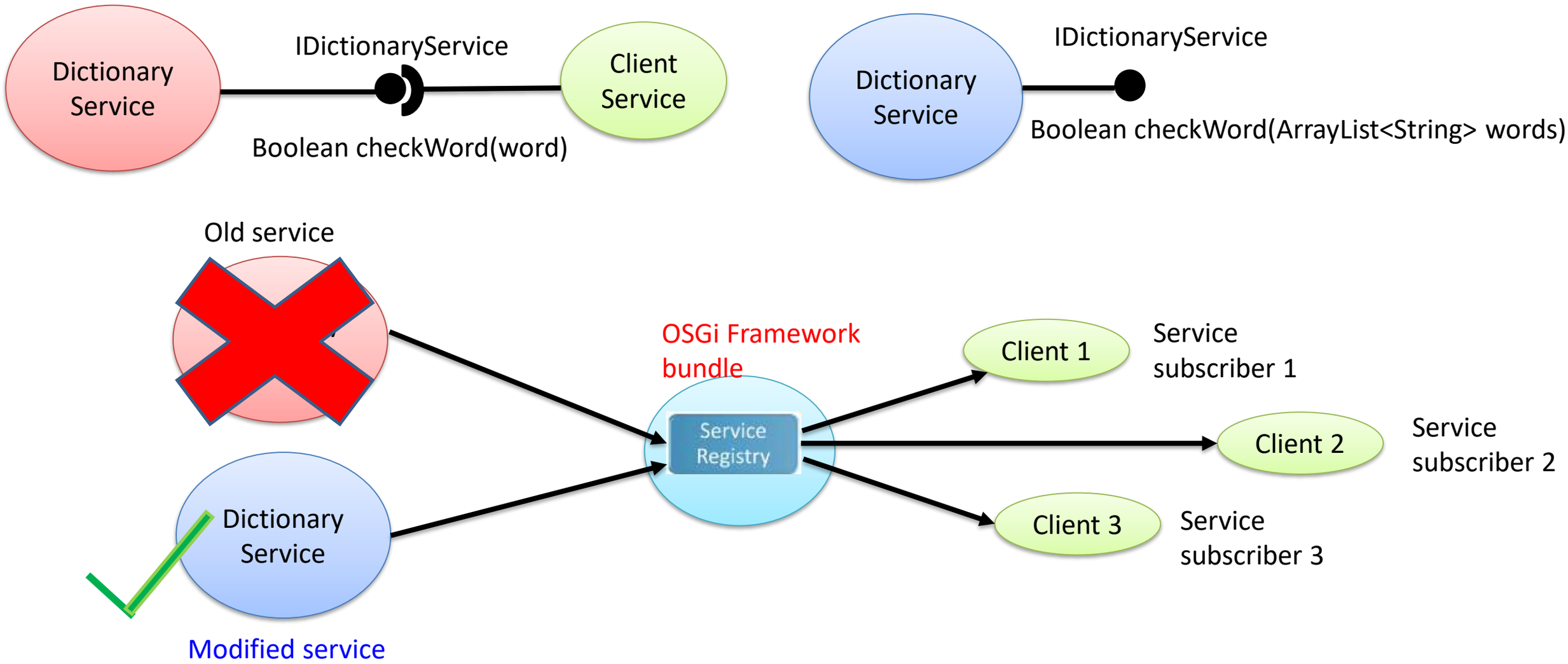




Practical Example



Practical Example



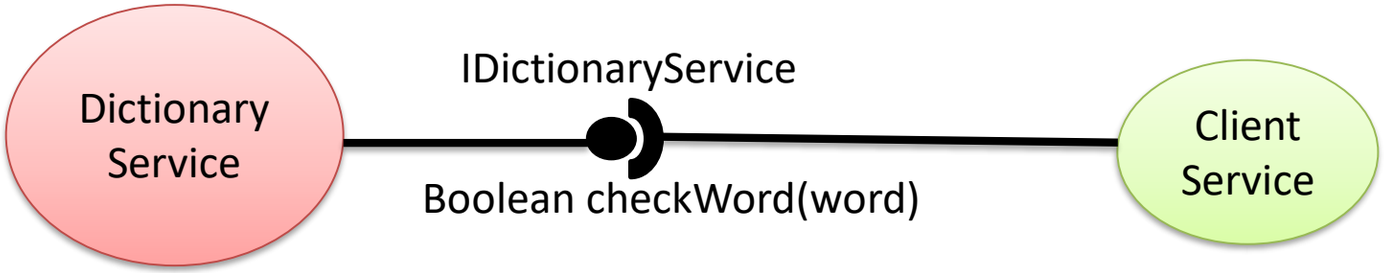
OSGi Environment

Manifest file

Service Activator

IDictionarySevice

DictionarySeviceImpl



Manifest file

Service Activator

Dictionary Service Manifest file

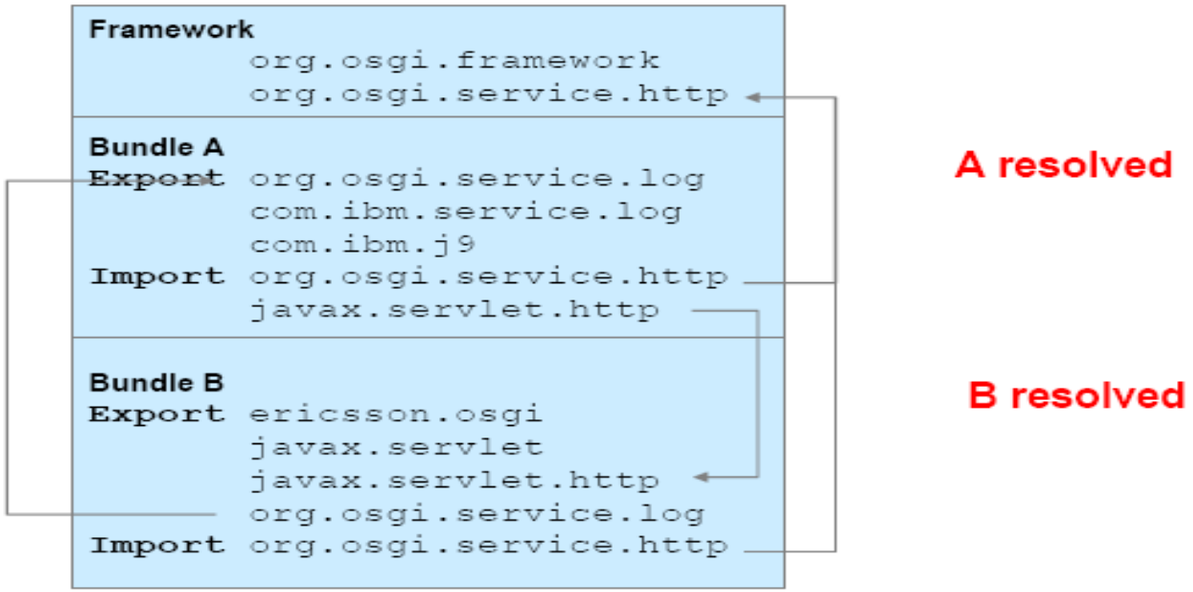
```
Bundle-Name: English dictionary
Bundle-Description: A bundle that registers an English dictionary service
Bundle-Vendor: Apache Felix
Bundle-Version: 1.0.0
Bundle-Activator: tutorial.example2.Activator
Export-Package: tutorial.example2.service
Import-Package: org.osgi.framework
```

Client Manifest file

```
Bundle-Name: Service Tracker-based dictionary client
Bundle-Description: A dictionary client using the Service Tracker.
Bundle-Vendor: Apache Felix
Bundle-Version: 1.0.0
Bundle-Activator: tutorial.example5.Activator
Import-Package: org.osgi.framework, org.osgi.util.tracker, tutorial.example2.service
```

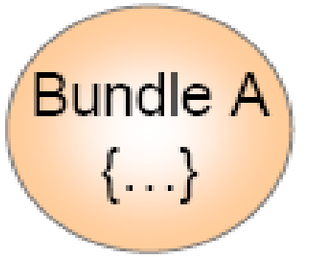
Bundle Deployment

- Bundles are deployed on an OSGi *framework*, the bundle runtime environment.
- This is not a container like Java Application Servers. It is a *collaborative environment*.



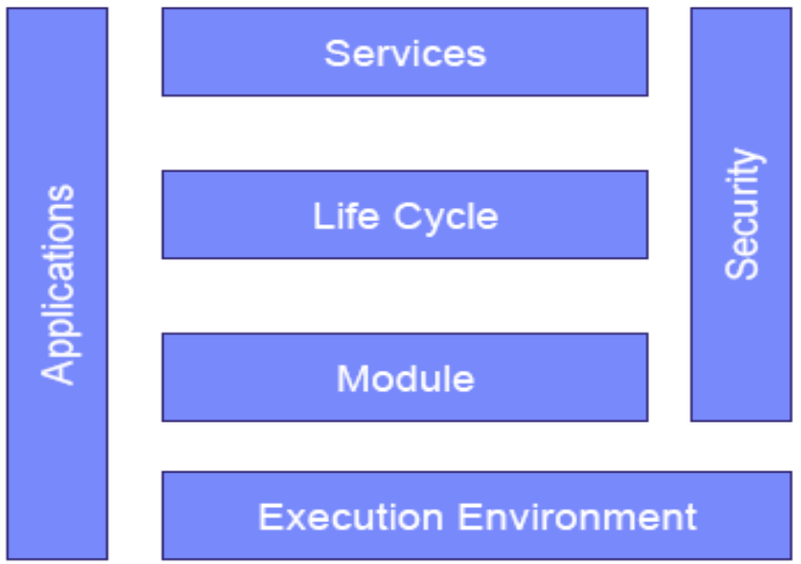
- Bundles run in the same VM and can actually share code.
- The framework uses the explicit imports and exports to wire up the bundles so they do not have to concern themselves with class loading.

- A Bundle contains (normally in a JAR file):
  - Manifest (bundle meta data)
  - Code (classes in packages)
  - Resources (other files in the JAR file)
- The Framework:
  - Reads the bundle's manifest
  - Installs the code and resources
  - Resolves dependencies
  - Controls the bundle life cycle
- During Runtime:
  - Calls the `Bundle Activator` to start the bundle
  - Manages java class path for the bundle as a network of class loaders
  - Handles the service dependencies
  - Calls the `Bundle Activator` to stop the bundle
  - Cleans up after the bundle



OSGi Service Platform Layering

- The OSGi Service Platform is divided in a number of layers.
- Execution Environment provides a defined context for applications.
- The Module layer provides class loading and packaging specifications.
- The Services layer provides a collaboration model.
- The extensive Security layer is embedded in all layers.



OSGi Service Layer

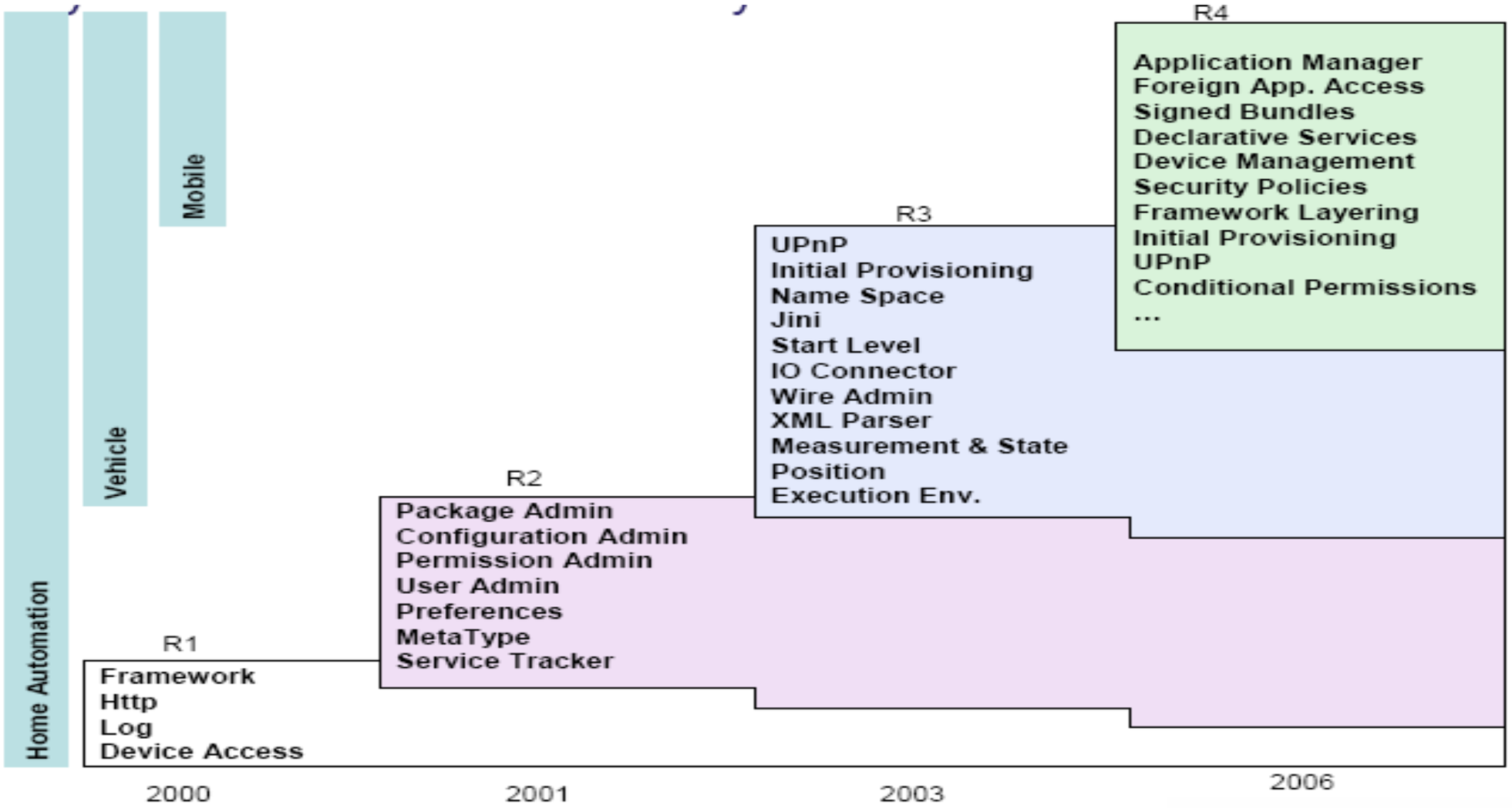
- Provides an inside-VM service model
  - Discover (and get notified about) services based on their interface or properties, no protocol required
  - Bind to one or more services by
    - program control,
    - default rules, or
    - deployment configuration

- Service Oriented Architectures (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

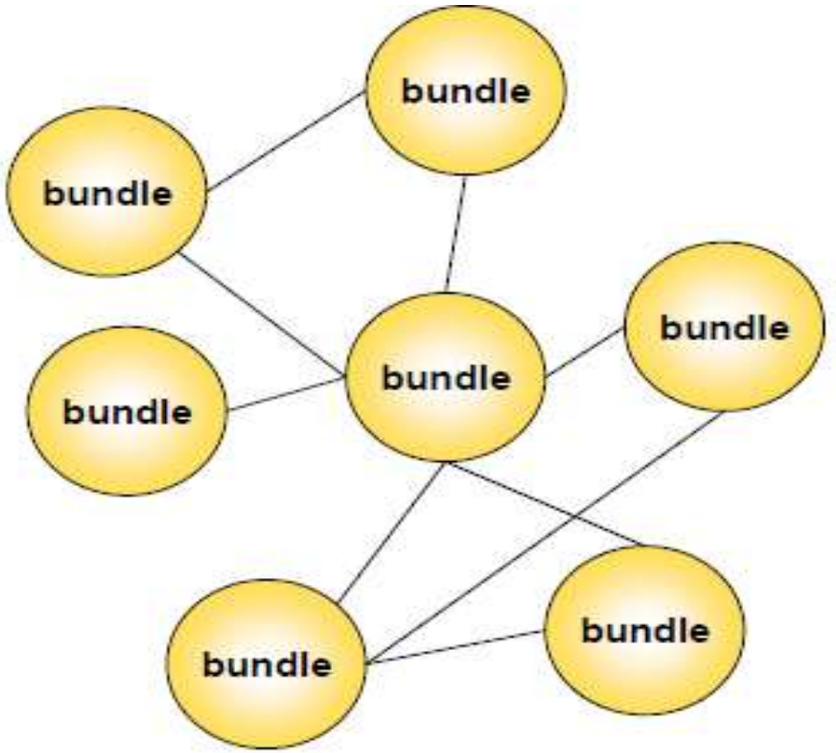
- OSGi defines a standard set of services
  - Other organizations can define more (AMI-C, Ertico, JCP)

The OSGi Service Layer Evolution

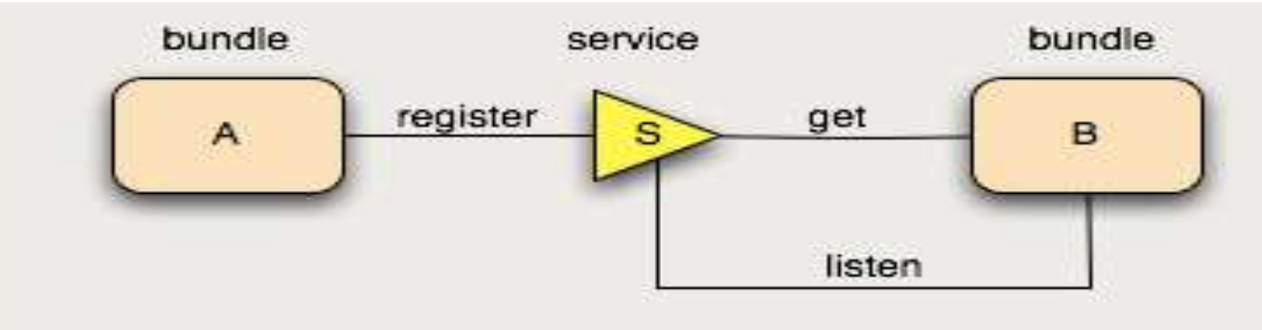


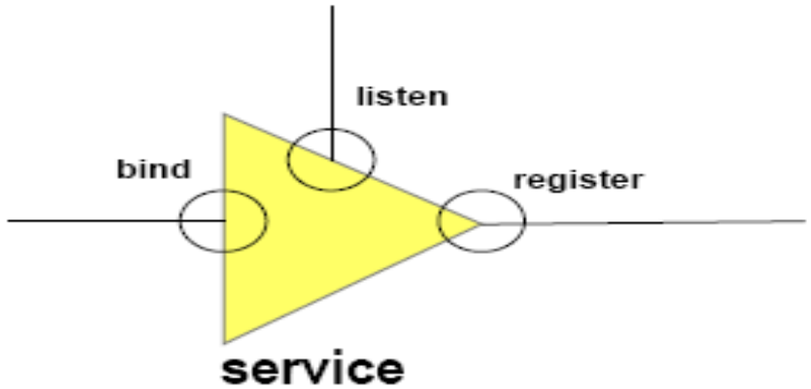
OSGi Module Layer

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



OSGi Services





- The Framework Service Registry is available to all bundles to collaborate with other bundles.

- Different bundles (from different vendors) can implement the same interface
  - Implementation is not visible to users
  - Allows operator to replace implementations without disrupting service

- A bundle can create an object and register it with the **OSGi service registry** under one or more interfaces.

- Other bundles can go to the registry and list all objects that are registered under a specific interfaces or class.
- A bundle can therefore *register a service*, it can *get a service*, and it can *listen for a service* to appear or disappear.
- Multiple bundles can register objects under the same interface or class with the same name.
- Services are associated with properties
  - Powerful query language to find appropriate service
  - Bundles can update the properties of a service dynamically
- A bundle can decide to withdraw its service from the registry while other bundles are still using this service

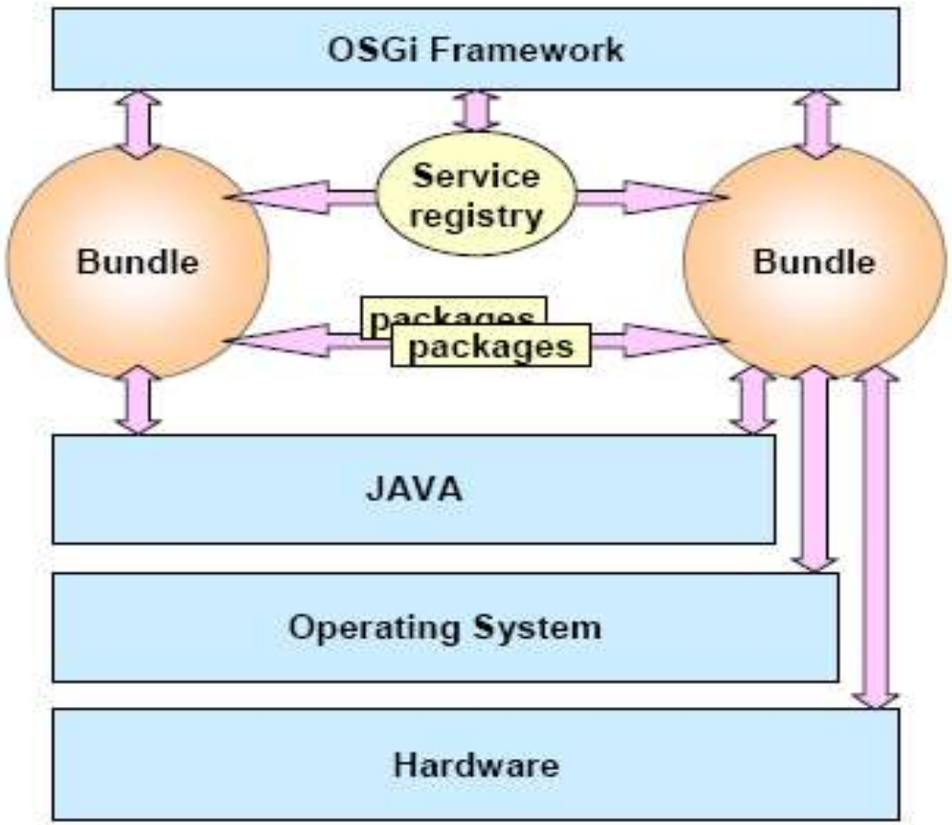
- A bundle can use a service (bind to) with any cardinality
  - 1..1, 0..1, 0..n

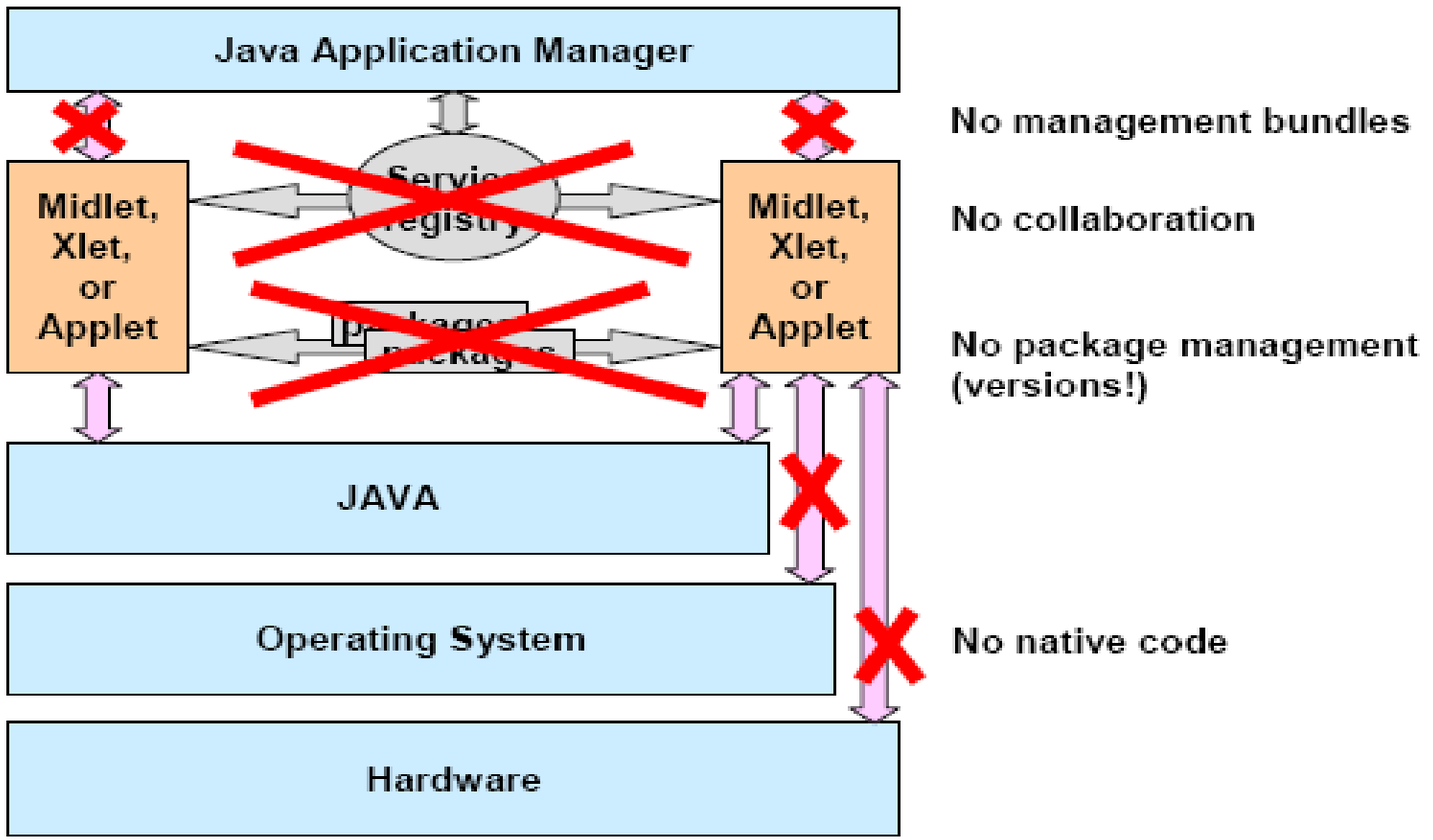
- A service can be discovered dynamically
  - Active search with query filter
  - Listener interface

- Services are dynamic
  - A bundle can decide to withdraw its service from the registry while other bundles are still using this service. Bundles using such a service must then ensure that they no longer use the service object and drop any references.

- Services can go away at any time! This is very dynamic!

Component interaction and collaboration





Functionalities supported by OSGi

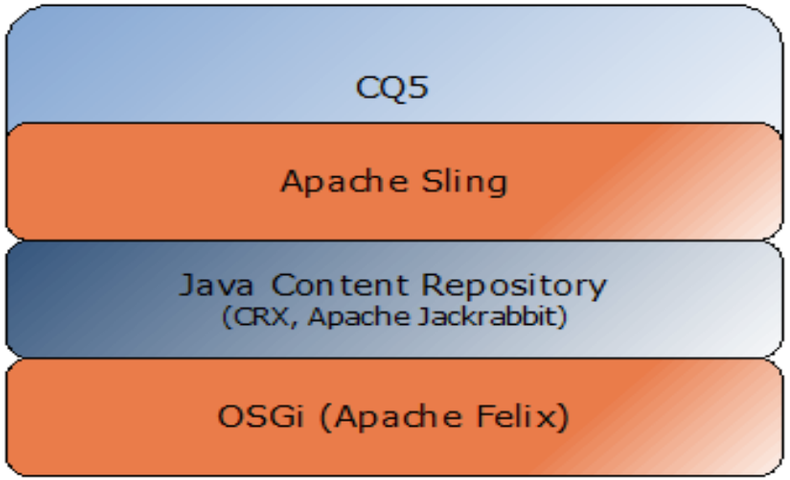
1) Reduces the complexity of the system.
2) Managing service/component dependencies.
3) Makes the components loosely-coupled and easy to manage.
4) Increases the performance of the system.

OSGi increase performance.

- Consider a scenario where you have a large application which uses a logging framework.
- This logging framework can be deployed as an OSGi Bundle, which can be managed independently.
- Therefore, it can be started when required by our application and can be stopped when not in use.
- Also the OSGi container makes these **bundles available as services**, which can be subscribed by other parts of application.

Custom products uses OSGi

Adobe CQ (CMS app)



- CQ5, uses the Apache Felix implementation of OSGI.
- Apache Felix is a open-source project to implement the OSGi R4 Service Platform.
- That includes
  - OSGi framework and standard services.
  - OSGi-related technologies.

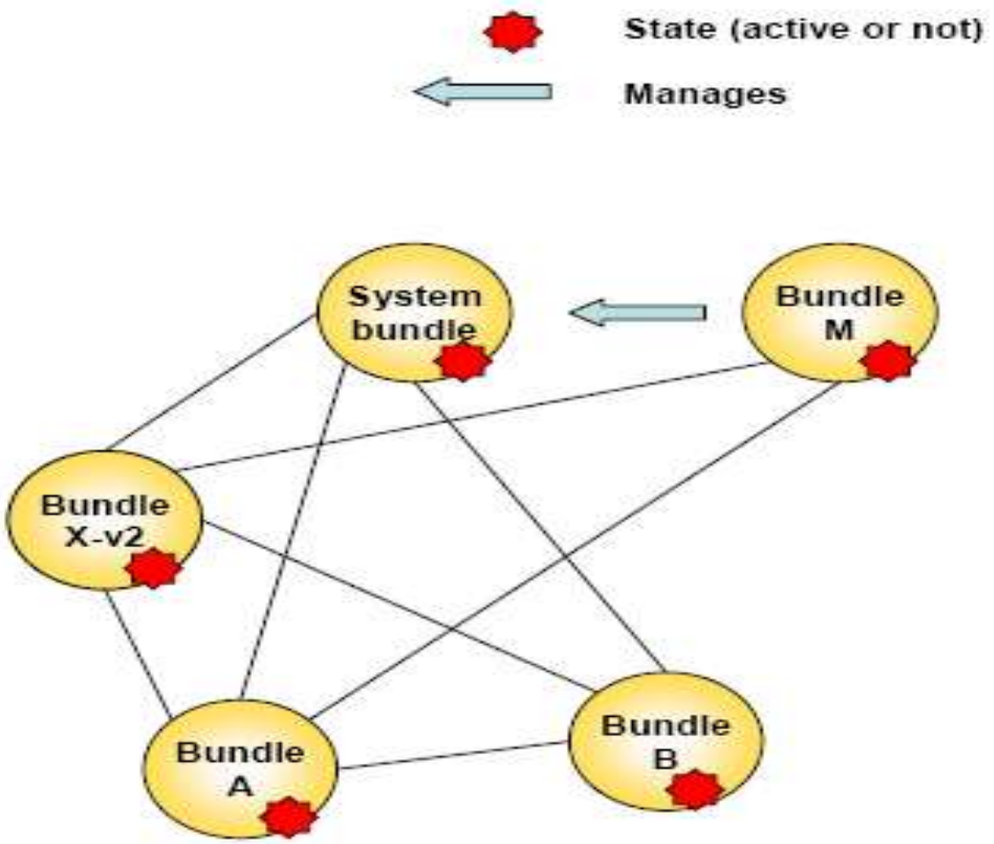
What is a Carbon Component

- A set of OSGi Bundles.
- Lives in the Carbon Framework. Hence should conform to rules define in the Carbon Framework.

- Develop the Carbon component
  - Back-end component (BE OSGi bundles)
  - Front-end component (FE OSGi bundles)
  - Common bundles, if any

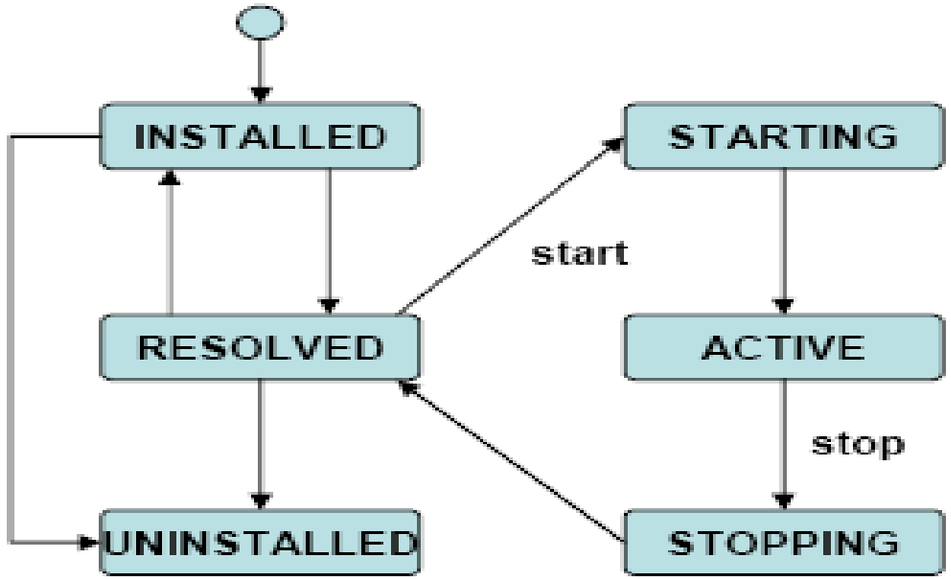
Layers: OSGi Life Cycle Layer

- System Bundle represents the OSGi Framework
- Provides an API for managing bundles
  - Install
  - Resolve
  - Start
  - Stop
  - Refresh
  - Update
  - Uninstall



Layers: OSGi Life Cycle Layer

- Bundle is started by the *Bundle Activator* class
- Header in the JAR manifest file refer to this class
- *Bundle Activator* interface has 2 methods
  - Start: Initialize and return immediate
  - Stop: Cleanup
- The *Bundle Activator* gets a *Bundle Context* that provides access to the OSGi Framework functions.
- The Framework provides the Start Level service to control the start/stop of groups of applications.



OSGI life cycle methods

org.osgi.framework

BundleActivator

- start(BundleContext) : void
- stop(BundleContext) : void

States of bundles

- Installed – finish bundle installation
- Active – start the bundle
- Resolved – stop the bundle

org.osgi.framework

BundleContext

- addBundleListener(BundleListener) : void
- addFrameworkListener(FrameworkListener) : void
- addServiceListener(ServiceListener) : void
- addServiceListener(ServiceListener, String) : void
- createFilter(String) : Filter
- getAllServiceReferences(String, String) : ServiceReferer
- getBundle() : Bundle
- getBundle(long) : Bundle
- getBundles() : Bundle[]
- getDataFile(String) : File
- getProperty(String) : String
- getService(ServiceReference) : Object
- getServiceReference(String) : ServiceReference
- getServiceReferences(String, String) : ServiceReferenc
- installBundle(String) : Bundle
- installBundle(String, InputStream) : Bundle
- registerService(String[], Object, Dictionary) : ServiceR
- registerService(String, Object, Dictionary) : ServiceReg
- removeBundleListener(BundleListener) : void
- removeFrameworkListener(FrameworkListener) : void
- removeServiceListener(ServiceListener) : void
- ungetService(ServiceReference) : boolean

Manipulating Services

- The *Bundle Context* provides the methods to manipulate the service registry
- Services registrations are handled by *Service Registration* objects
  - They can be used to unregister a service or modify its properties
- *Service Reference* objects give access to the service as well as to the service's properties
- Access to service objects is through the *getService* method. These services should be returned with the *ungetService* method.

```
ServiceRegistration registerService(
  String clss,
  Object srvc,
  Dictionary prprts)

ServiceReference[]
  getServiceReferences(
    String clss,
    String fltr)

Object getService(
  ServiceReference reference)

boolean ungetService(
  ServiceReference rfrnc);
```

References

- http://www.osgi.org/Main/HomePage
- http://www.osgi.org/Technology/WhatIsOSGi
- http://en.wikipedia.org/wiki/OSGi
- http://grepcode.com/file/repository.grepcode.com/java/eclipse.org/3.5/org.eclipse/osgi/3.5.0/org/osgi/framework/BundleContext.java#BundleContext.getBundle%28long%29
- *http://felix.apache.org/site/apache-felix-framework-usage-documentation.html*