Introduction to OSGi

The Dynamic Module System for Java

Luís Osório

Value proposal

- Run Multiple applications on the same JVM
  - For Java applications that comply with OSGi, as discussed by IBM [ref]
- Dependency management
  - Bundle (interface/implementation) or Service [ref] as strategies to manage dependency (service concept makes clients loose-coupled to its implementation)
- Version control
  - Versions dependency declaration into MANIFEST file (Bundle/jar manifest file)
  - Helps to manage component/bundle or service lifecycle management
- Possible to install and uninstall
  - Uninstall a bundle of some version not maintained anymore without stopping running applications
  - Start or stop runnable elements

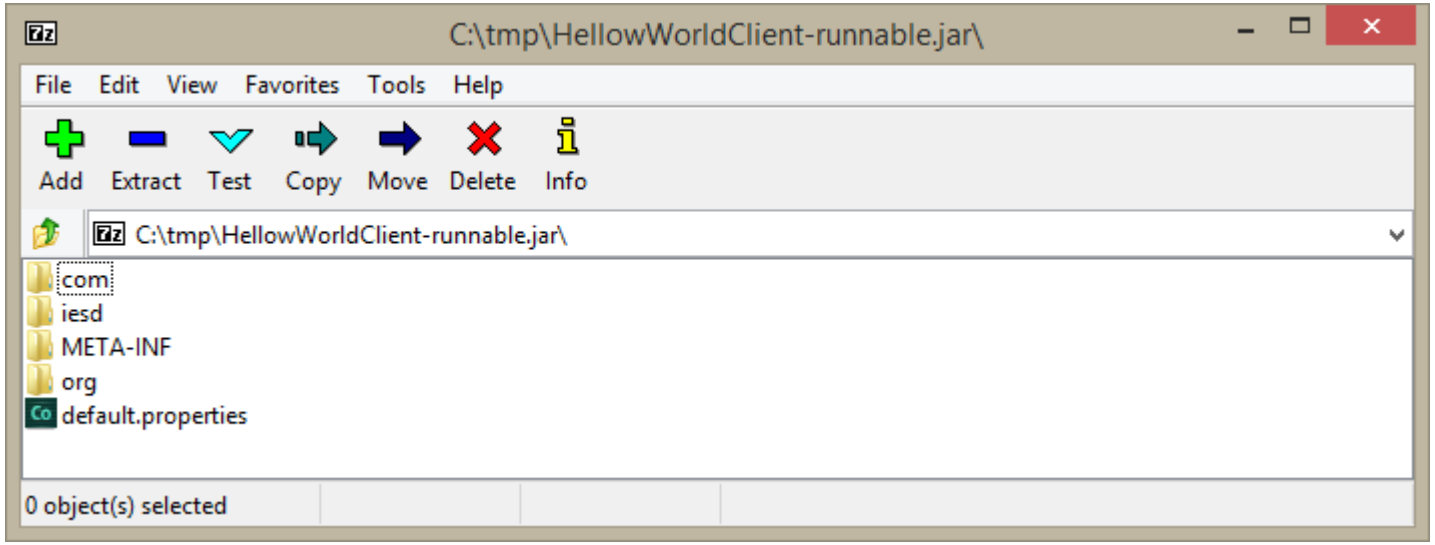
The OSGi Alliance (www.osgi.org)

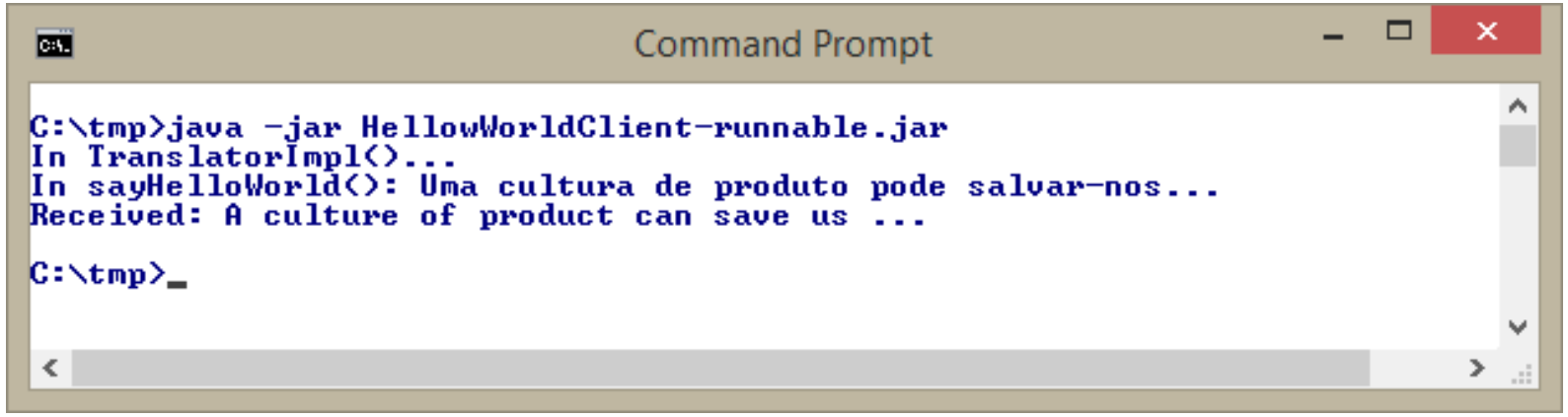
- The OSGi™ Alliance, March 1999, now Eclipse.org.
  - Its mission is to create open specifications for the network delivery of managed services to local networks and devices.
- OSGi defines a component and service model for Java
  - Where components and services can be dynamically installed/uninstalled, activated/deactivated and updated.
  - Defines an OSGi Framework and Services platform
- Reference implementations:
  - Apache Felix,
  - Equinox (reference implementation and the base for the Eclipse platform),
  - Knopflerfish
- Management tools
  - Apache Karaf, and specialized Eclipse plug-ins

Other Component Frameworks

- OMG/CORBA
  - ORB, Internet Inter-ORB Protocol (IIOP), CORBA Component Model (CCM); Ver.4.0, 2006
- Microsoft
  - The Component Object Model (COM) e Distributed Component Object Model (DCOM)
  - COM+ (Component Services), WCF/.NET Remoting
  - Managed Extensibility Framework (MEF)
- OASIS (Advanced Open Standards for the Information Society)
  - Service Component Architecture (SCA)
  - Web Services Component Model (WSCM)
    - OASIS Web Services Interactive Applications TC
- Research Component Frameworks
  - BIP (Behavior, Interaction, Priority)
  - CES; more recently evolved to ISoS with ISystem, CES (Cooperation enabled Services), and Service elements
- A research question: How to establish a standard component model for open informatics system (ISystem)
  - (i.e. multi-vendor ISystem/components as parts of composite informatics solutions)?

Execution from the Java command line





Execution from the Java command line

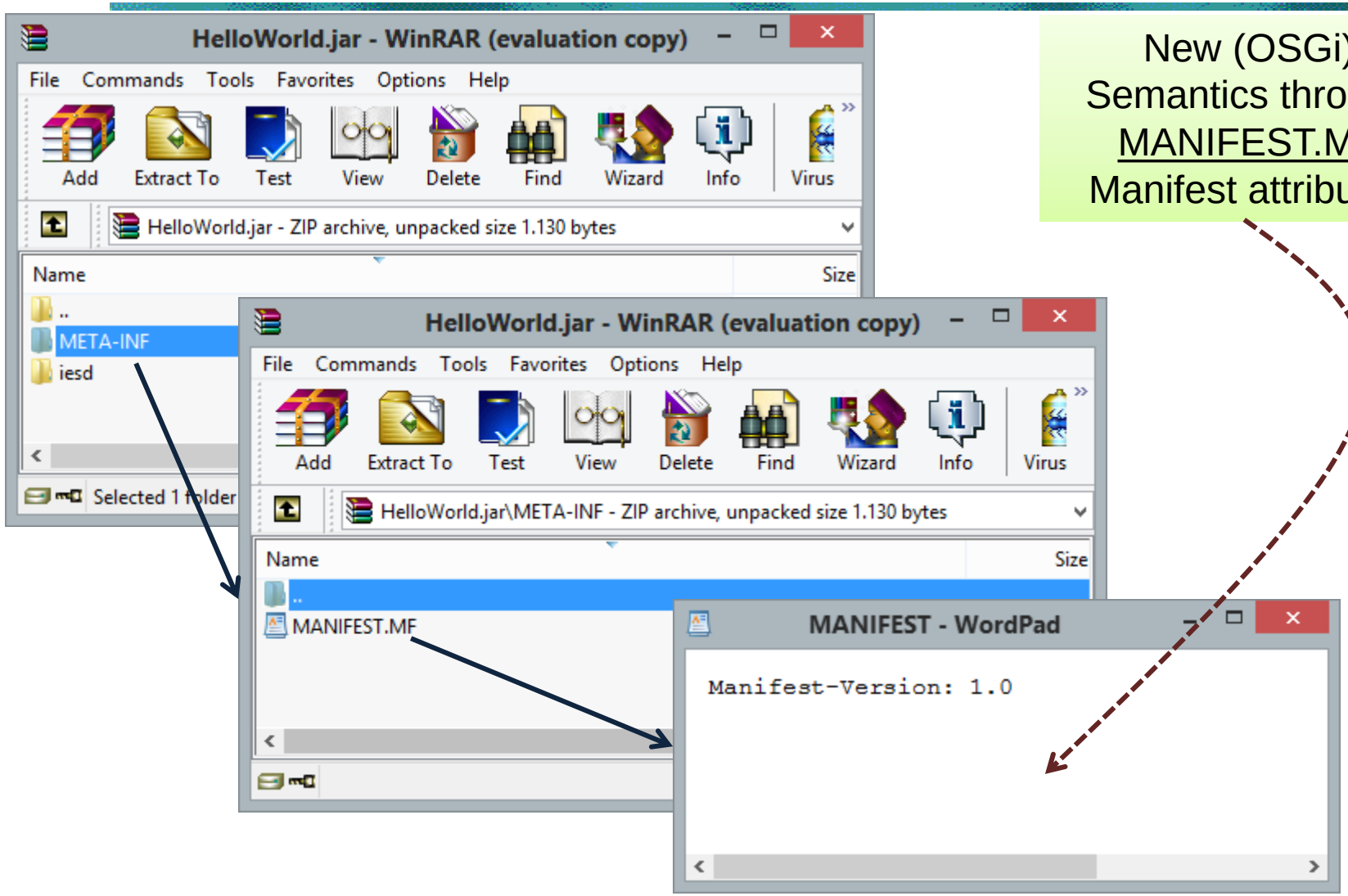
Command Prompt

```
C:\tmp>java -cp HelloWorld.jar;HelloWorldImpl.jar;Translator.jar;TranslatorImpl.
jar;microsoft-translator-java-api-0.5-jar-with-dependencies.jar;HelloWorldClient
.jar iesd.osgi.helloworld.cliente.HelloWorldClient
In TranslatorImpl()...
In sayHelloWorld(): Uma cultura de produto pode salvar-nos...
Received: A culture of product can save us ...

C:\tmp>
```

C:\tmp> java -classpath HelloWorld.jar;HelloWorldImpl.jar;Translator.jar;HelloWorldClient.jar;TranslatorImpl.jar;microsoft-translator-java-api-0.5-jar-with-dependencies.jar iesd.osgi.helloworld.cliente.HelloWorldClient

All the JAR are in the c:\tmp directory

Reuse JAR files



New (OSGi) Semantics through MANIFEST.MF Manifest attributes

Simple HelloWorld OSGi/Java Application

MANIFEST.MF

```
Manifest-version 1.0
Bundle-ManifestVersion: 2
Bundle-Name: HelloWorldClient Bundle
Bundle-SymbolicName: HelloWorldClient
Bundle-Version: 0.0.1
Import-Package: org.mdeos.osgi.helloworld.api;version="[0.0,1)",org.md
 eos.osgi.helloworld.impl;version="[0.0,1)“, org.osgi.framework;version="[1.8,2)“
Bundle-Activator: org.mdeos.osgi.helloworld.client.HelloWorldClient
```

HelloWorldClient.jar

```
META-INF/MANIFEST.MF
org/mdeos/osgi/helloworld/client/HelloWorldClient.class
```

HelloWorldImpl.java

```
public class HelloWorldImpl implements  HelloWorld {
public String sayHello(String msg) {
  System.out.println("In sayHello(): " + msg);
  return msg.toUpperCase();
}
```

HelloWorld.java

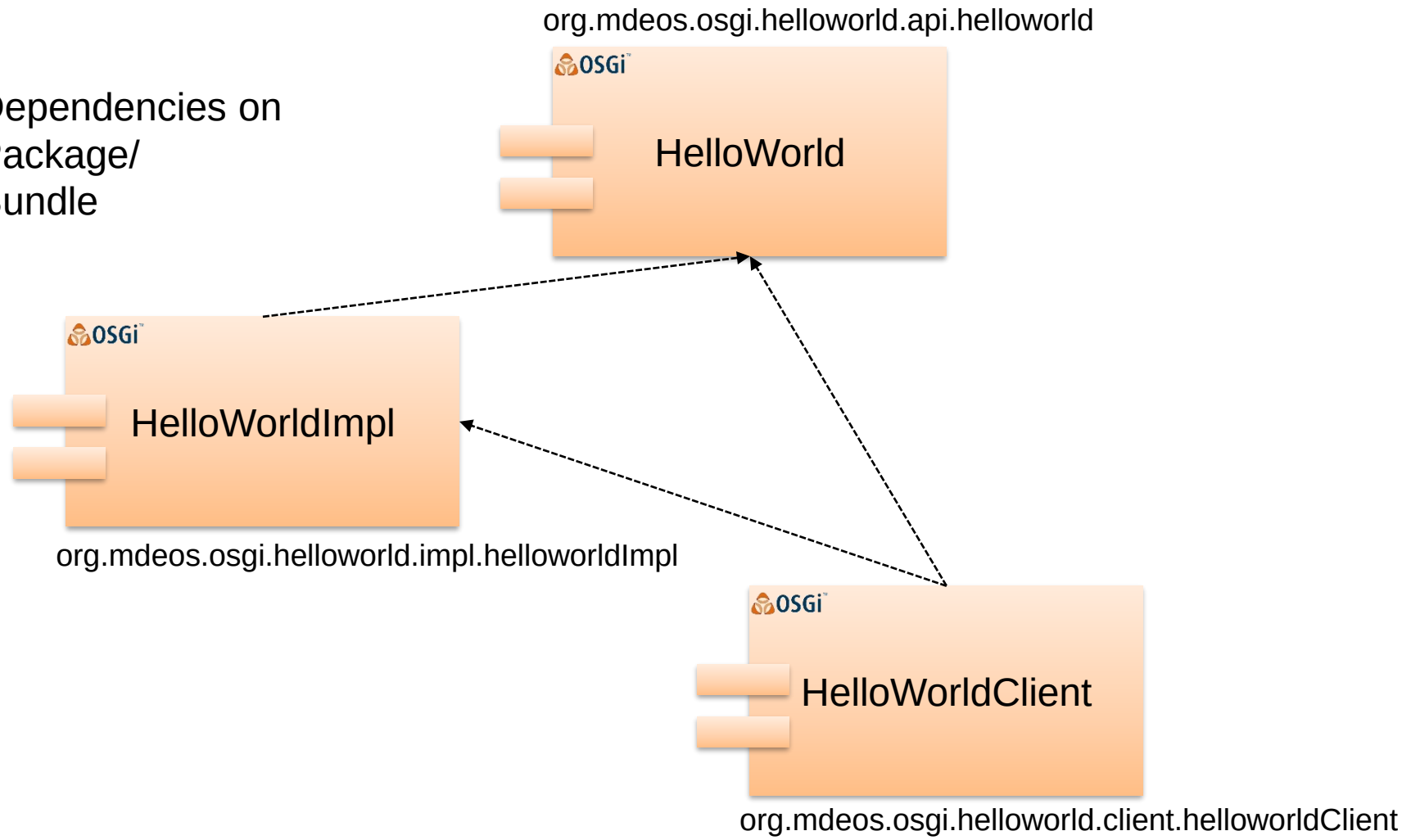
```
public interface HelloWorld {
    public String sayHello(String msg);
}
```

HelloWorldClient.java

```
public class HelloWorldClient implements BundleActivator {
  public void start(BundleContext context) throws Exception {
    System.out.println("start(): Starting ShowMessage bundle...");
    HelloWorld  helloWorld = new HelloWorldImpl().sayHello(“Hello World…”);
}
  public void stop(BundleContext context) throws Exception {
    System.out.println("Stopping HelloWorldClient bundle...");
}
```

OSGi Bundle, package and services dependencies

Dependencies on Package/ Bundle

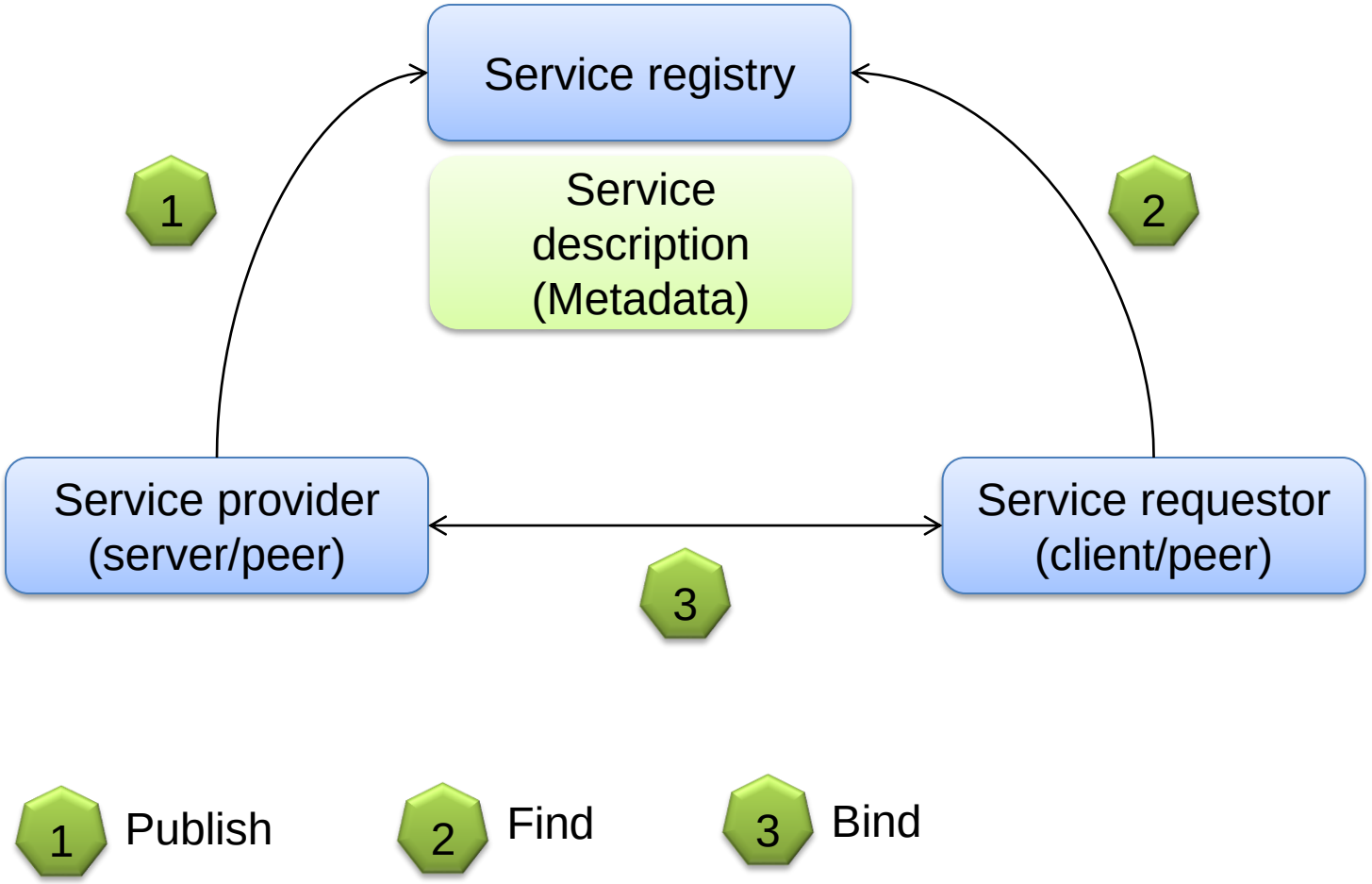


Running in Appache Felix

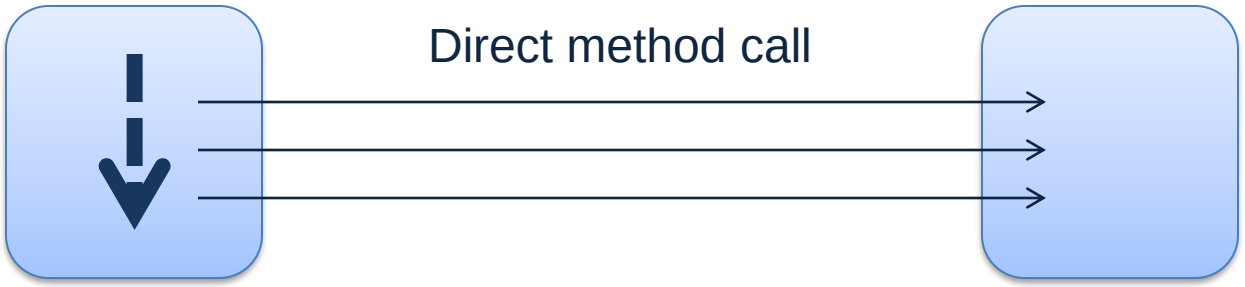
```
C:\Java\felix-framework-5.6.1>java -jar -jar bin\felix.jar
____________________________
Welcome to Apache Felix Gogo

g! install file:c:/tmp/HelloWorld/HelloWorld-0.0.1.jar
Bundle ID: 6
g! install file:c:/tmp/HelloWorld/HelloWorldImpl-0.0.1.jar
Bundle ID: 7
g! install file:c:/tmp/HelloWorld/HelloWorldClient-0.0.1.jar
Bundle ID: 8
g! lb Hello
START LEVEL 1
   ID|State      |Level|Name
    6|Installed  |    1|HelloWorld Bundle (0.0.1)|0.0.1
    7|Installed  |    1|HelloWorldImpl Bundle (0.0.1)|0.0.1
    8|Installed  |    1|HelloWorldClient Bundle (0.0.1)|0.0.1
g! start 8
In sayHelloWorld(): Hello World!
start(): HELLO WORLD!
g! lb Hello
START LEVEL 1
   ID|State      |Level|Name
    6|Resolved   |    1|HelloWorld Bundle (0.0.1)|0.0.1
    7|Resolved   |    1|HelloWorldImpl Bundle (0.0.1)|0.0.1
    8|Active     |    1|HelloWorldClient Bundle (0.0.1)|0.0.1
g!
```

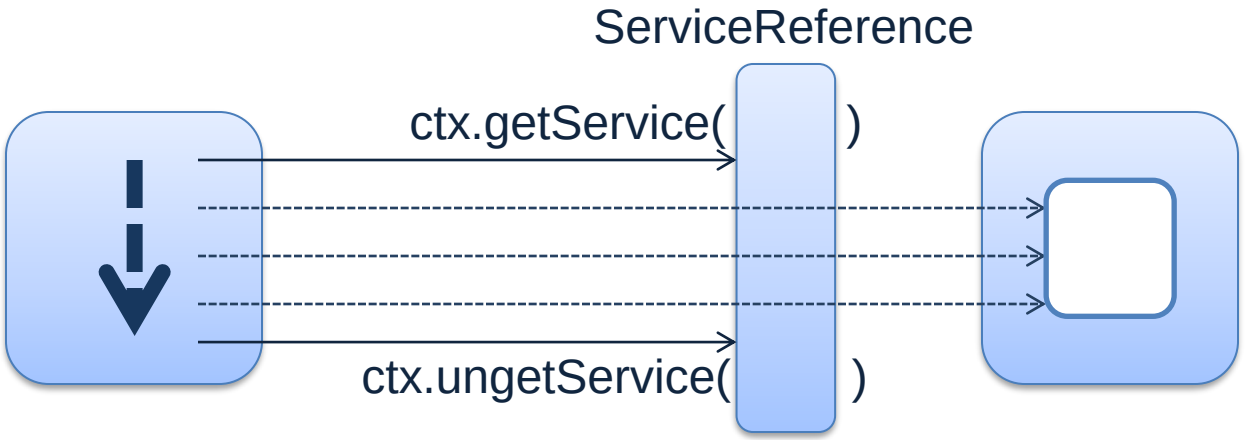
Install a Bundle (jar file)

OSGi Services

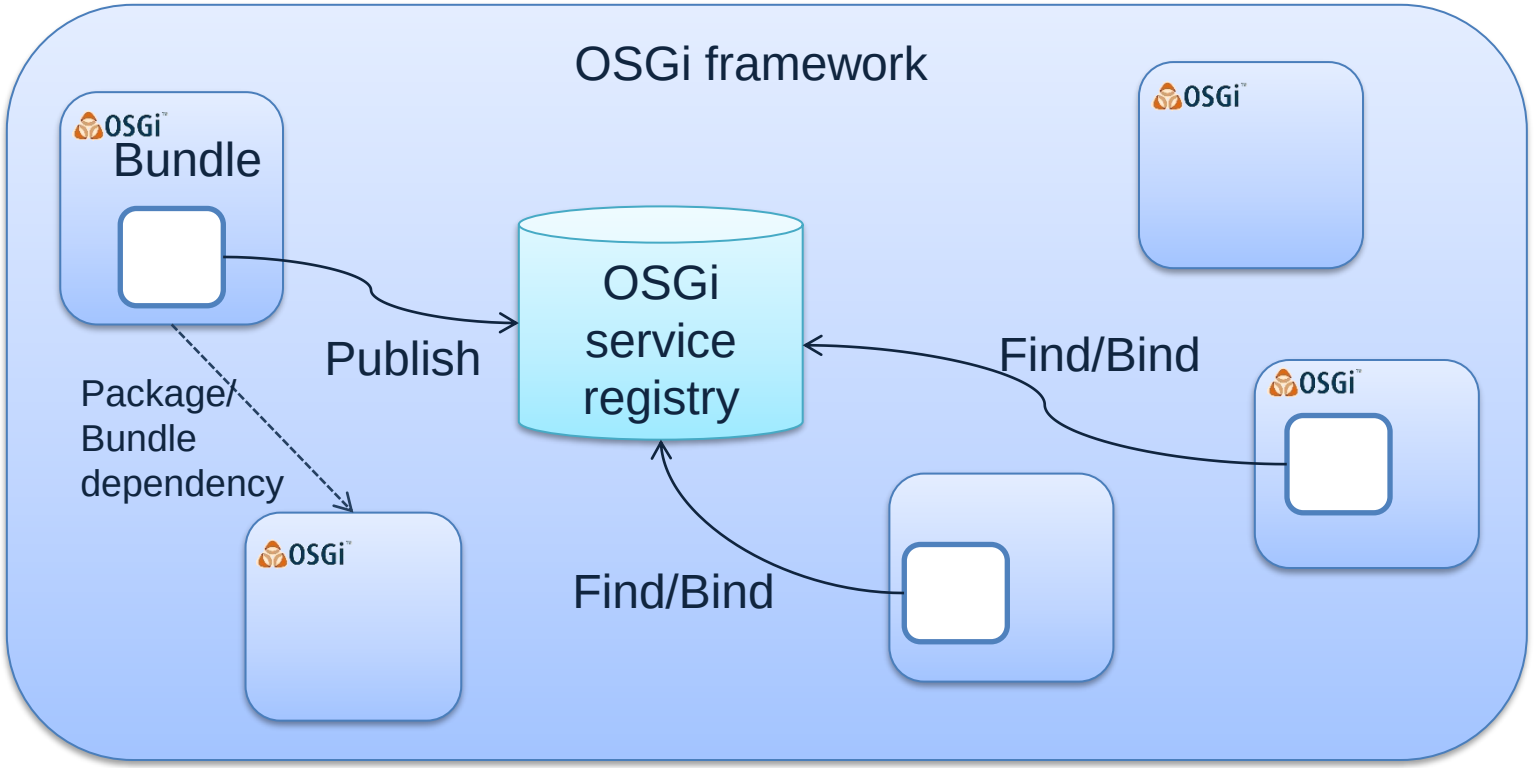


OSGi Services invocation vs method calls





OSGi Services Publish/Bind/Find



Service HelloWorld OSGi/Java Application

HelloWorldActivator.java

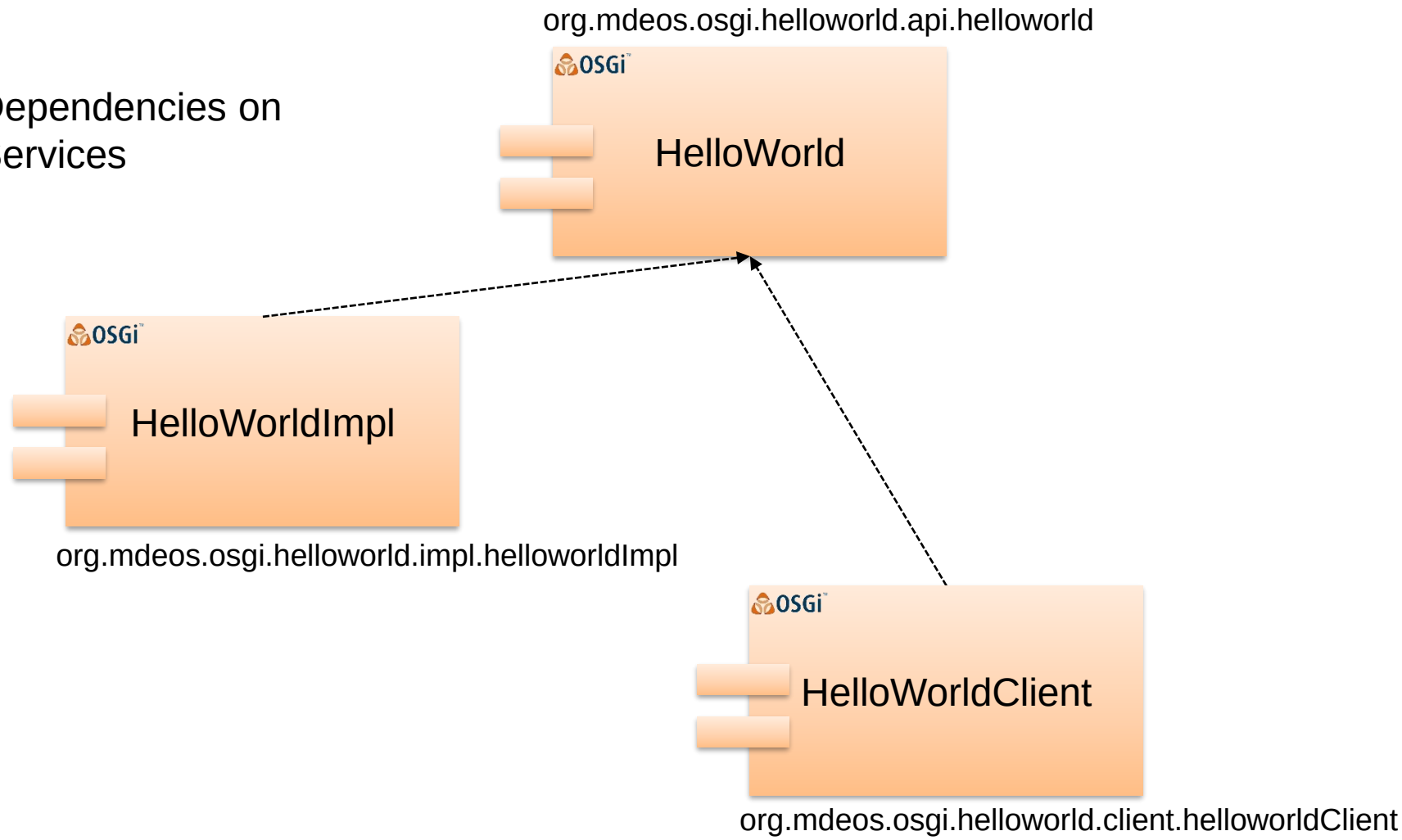
```
public class HelloWorldActivator implements BundleActivator  {
 Dictionary<String, String>  metadata = new Hashtable<String, String>();

public void start(BundleContext context) throws Exception {
    HelloWorld helloWorld = new HelloWorldImpl();
    metadata.put("ServiceName", "HelloWorld");
    metadata.put("ServiceVersion", "0.0.1");
    metadata.put("ServiceProvider", "MDEOS.examples");
    context.registerService(HelloWorld.class.getName(), helloWorld, metadata);
} …
```

HelloWorldClient.java

```
public class HelloWorldClient implements BundleActivator {
  private HelloWorld helloWorld = null;
  private ServiceReference<?> serviceReference = null;

  public void start(BundleContext context) throws Exception {
    serviceReference = context.getServiceReference(HelloWorld.class.getName());
    helloWorld = (HelloWorld) context.getService(serviceReference);
    System.out.println("start(): " + helloWorld.sayHello("HelloWorld!"));
} …
```

OSGi Bundle, package and services dependencies

Dependencies on Services



Declarative Services (DS)

- It is a model similar to the alternative OSGi specification Blueprints, also a Dependency Injection/Inversion of Control pattern implementation
  - A component is declared by a XML configuration file positioned at the Bundle's root OSGI-INF directory and identified in the Bundle's manifest attribute:
    - **Service-Component**: OSGI-INF/HelloWorld-Component.xml
    - A configuration file can declare one or more components
  - A component can declare zero or more services (interfaces) and reference zero or more services

- The Apache Felix Service Component Runtime (SCR)
  - Manages a suite of annotations for Java code making easier the generation of a component/service declaration file.
    - @Component, @Activate, @Deactivate, @Modified, @Service, @Property, @Reference

DS Service HelloWorld OSGi/Java Application

HelloWorldClient.java

```
public class HelloWorldClient {
  HelloWorld helloWorld;

  void activate(ComponentContext componentContext) { // role of start()
    System.out.println("HelloWorldClient.activate()...");
    this.componentContext = componentContext;
    // call sayHello() example
    System.out.println("activate(): " + helloWorld.sayHello("Hello World!"));
  }  …
  protected void bindHelloWorld(HelloWorld helloWorld) {
    System.out.println("HelloWorldClient.bindHelloWorld()...");
    this.helloWorld = helloWorld;
  }
…
}
```

```
public class HelloWorldImpl implements HelloWorld {
  protected void activate(ComponentContext ctx) {
    System.out.println("HelloWorldImpl.activate()...");
  }
  protected void deactivate(ComponentContext ctx) {
    System.out.println("HelloWorldImpl.deactivate()...");
  }
  public String sayHello(String msg) {
    System.out.println("In sayHelloWorld(): " + msg);
    return msg.toUpperCase();
}}
```

HelloWorldImpl.java

DS Service HelloWorld OSGi/Java Application

<bundle>/OSGI-INF/org.mdeos.osgi.helloworld.client.HelloWorldClient.xml

```
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
activate="activate" configuration-policy="optional" deactivate="deactivate"
enabled="true" name="HelloWorldClient">
   <scr:implementation class="org.mdeos.osgi.helloworld.client.HelloWorldClient"/>
   <scr:reference
    name="helloWorld"
    interface="org.mdeos.osgi.helloworld.api.HelloWorld"
    policy="static"
    cardinality="1..n"
    bind="bindHelloWorld"
    unbind="unbindHelloWorld"/>
</scr:component>
```

HelloWorldClient.jar

```
Manifest-Version: 1.0
…
Bundle-Name: HelloWorldClient Bundle
Bundle-SymbolicName: HelloWorldClient
…
Service-Component: OSGI-
INF/org.mdeos.osgi.helloworld.client.
HelloWorldClient.xml
```

<bundle>/OSGI-INF/org.mdeos.osgi.helloworld.client.HelloWorldImpl.xml

```
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
     activate="activate" configuration-policy="ignore" deactivate="deactivate"
     enabled="true" name="HelloWorld">
   <scr:implementation class="org.mdeos.osgi.helloworld.impl.HelloWorldImpl"/>
    <property name="ServiceName" value="HelloWorld"/>
    <property name="ServiceVersion" value="0.0.3"/>
    <property name="ServiceProvider" value="mdeos.osgi.examples"/>
   <scr:service>
      <scr:provide interface="org.mdeos.osgi.helloworld.api.HelloWorld"/>
   </scr:service>
</scr:component>
```

DSa Service HelloWorld OSGi/Java Application

HelloWorldImpl.java

```
@Component(name = "HelloWorld Component", immediate = true,
  service = org.mdeos.osgi.helloworld.api.HelloWorld.class,
  property = {"ServiceName = HelloWorld",
              "ServiceVersion = 0.0.4",
              "ServiceProvider = mdeos.osgi.examples"
  }
)public class HelloWorldImpl implements HelloWorld {
  @Activate
  protected void activate(Map<String,Object> properties) {
    System.out.println("HelloWorldImpl.activate()...");
  }
  @Deactivate
  protected void deactivate(Map<String,Object> properties) {
    System.out.println("HelloWorldImpl.deactivate()...");
  }
  public String sayHello(String msg) {
    System.out.println("In sayHello(): " + msg);
    return msg.toUpperCase();
}}
```

The Declaration File is created by the annotations processor

/META-INF/MANIFEST.MF
/OSGI-INF/org.mdeos.osgi.helloworld.impl.HelloWorldImpl.xml
…

DSa Service HelloWorld OSGi/Java Application

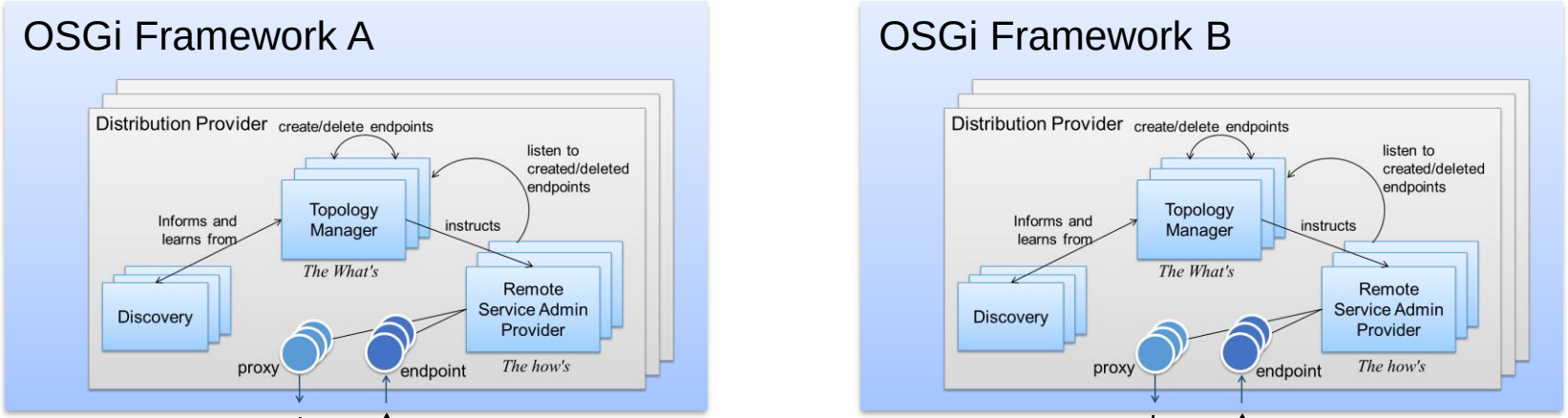
HelloWorldClient.java

```
@Component(name = "HelloWorldClient Component", immediate = true)
public class HelloWorldClient {
  @Reference (bind="bindHelloworld")
  HelloWorld helloWorld;
  ComponentContext componentContext = null;

  @Activate
  void activate(ComponentContext componentContext) {    // role of start()
    System.out.println("HelloWorldClient.activate()...");
    this.componentContext = componentContext;
 // call sayHello() example
    System.out.println("activate(): " + helloWorld.sayHello("Hello World!"));  }
…
  public void bindHelloworld(HelloWorld helloWorld) {
    System.out.println("HelloWorldClient.bindHelloworld called...");
    this.helloWorld = helloWorld;
…
}}
```

/META-INF/MANIFEST.MF
/OSGI-INF/org.mdeos.osgi.helloworld.client.HelloWorldClient.xml
…

The Declaration File is created by the annotations processor

OSGi Remote Services



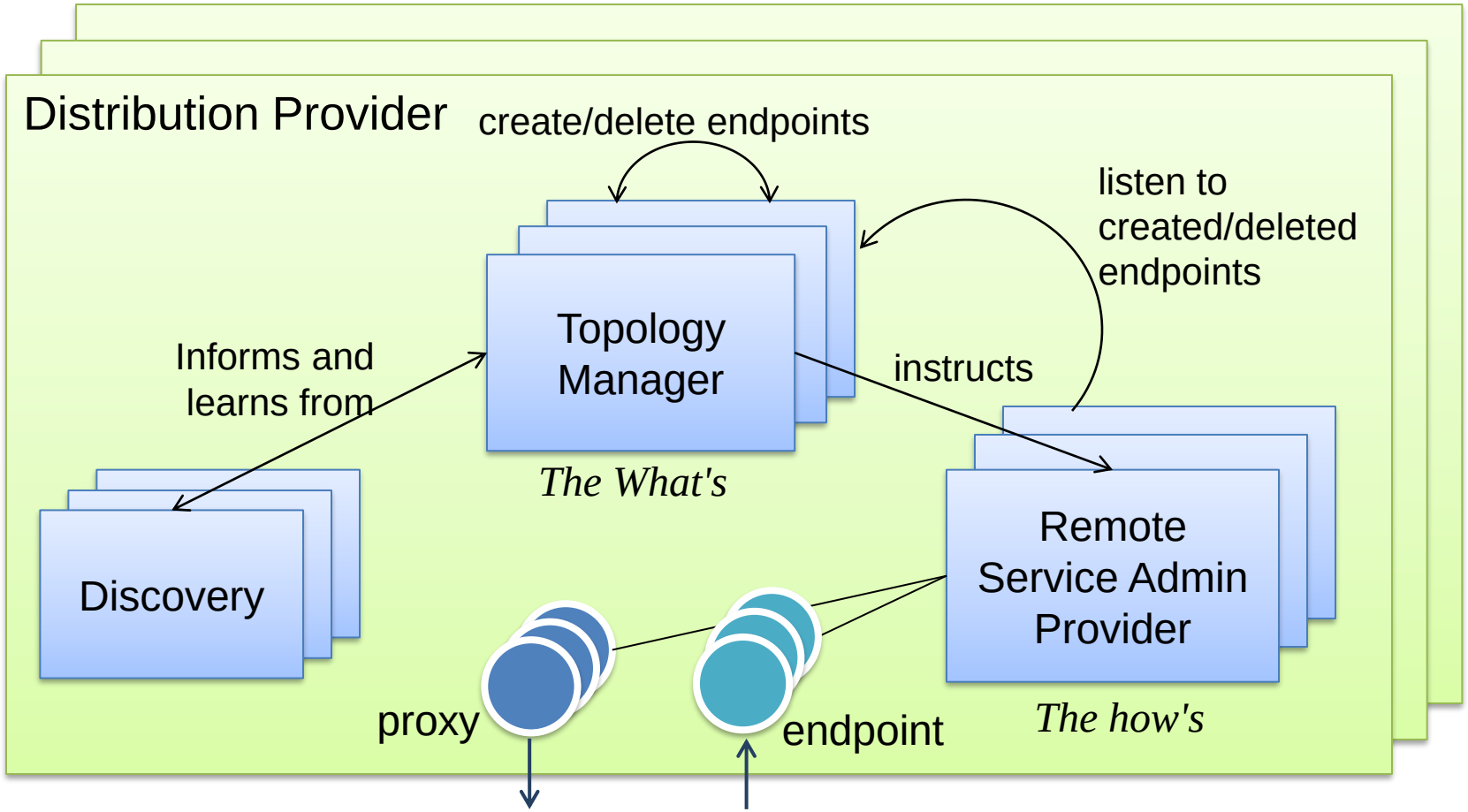
Adapt to multiple remoting frameworks
(RMI, SOAP/Rest, JMS, XMPP, …)

RSdsa Service HelloWorld OSGi/Java Application

HelloWorldImpl.java

```
package org.mdeos.osgi.helloworld.impl;
…
@Component(name = "org.mdeos.osgi.helloworld.impl.HelloWorld", immediate = true,
  service = org.mdeos.osgi.helloworld.api.HelloWorld.class,
              // Remote Services configuration
  property = {"service.exported.interfaces=*",
              "service.exported.configs=org.apache.cxf.ws",
              "org.apache.cxf.ws.address=http://192.168.56.101:9010/HelloWorld",
             // Service meta-data definition
             "ServiceName=HelloWorld",
             "ServiceVersion=0.0.5",
             "ServiceProvider=mdeos.osgi.examples" })
public class HelloWorldImpl implements HelloWorld {
  @Activate
  void activate(ComponentContext componentContext) { … }
  @Deactivate
  void deactivate(ComponentContext componentContext) { … }

  public String sayHello(String msg) {
    System.out.println("In sayHello(): " + msg);
    return msg.toUpperCase();
  }
}
```

Remote Services (100) and Remote Services Admin (122)



Execution on Eclipse platform (Plugin project)

Java EE - HelloWorldClient/src/iesd/osgi/helloworld/cliente/HelloWorldClient.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access | Java EE

Project Explorer

- HelloWorld
- HelloWorldClient
  - src
    - iesd.osgi.helloworld.cliente
      - HelloWorldClient.java
  - JRE System Library [jre1.8.0_31
- HelloWorldImpl
- Translator
- TranslatorImpl
  - src
    - iesd.osgi.translator.impl
      - TranslatorImpl.java
  - JRE System Library [jre1.8.0_31
  - microsoft-translator-java-api-

HelloWorldClient.java

```
12
13      public static void main(String[] args) {
14          String result;
15
16          HelloWorldImpl helloWorldImpl =  new HelloWorldImpl(
17          result = helloWorldImpl.sayHelloWorld(MSG);
18          System.out.println("Received: " + result);
19      }
20  }
21
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console

<terminated> HelloWorldClient [Java Application] C:\Program Files\Java\jre1.8.0_31\bin\javaw.exe (Feb 16, 2015, 10:37:38 PM)

```
In TranslatorImpl()...
In sayHelloWorld(): Uma cultura de produto pode salvar-nos...
Received: A culture of product can save us ...
```

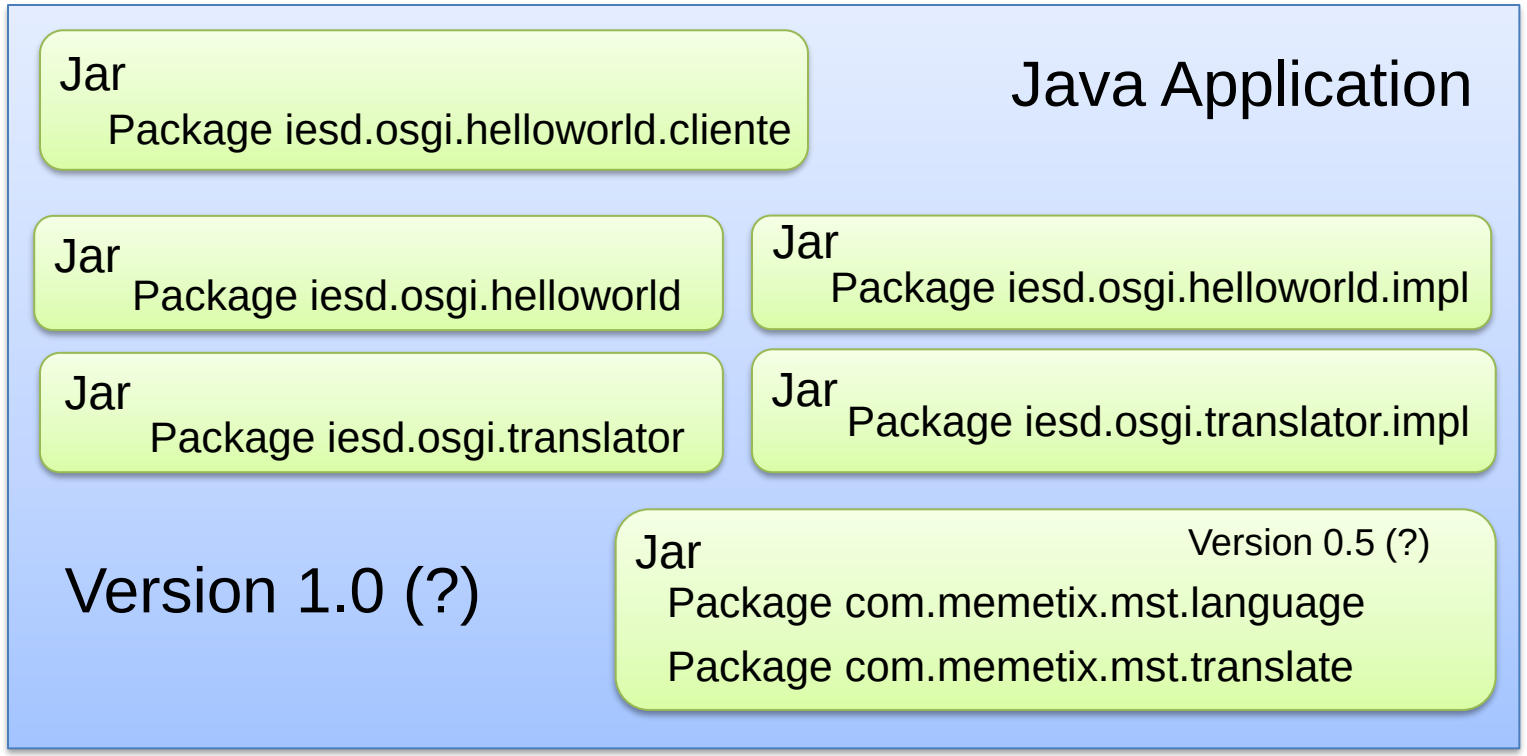
Outline | Task List

Connect Mylyn

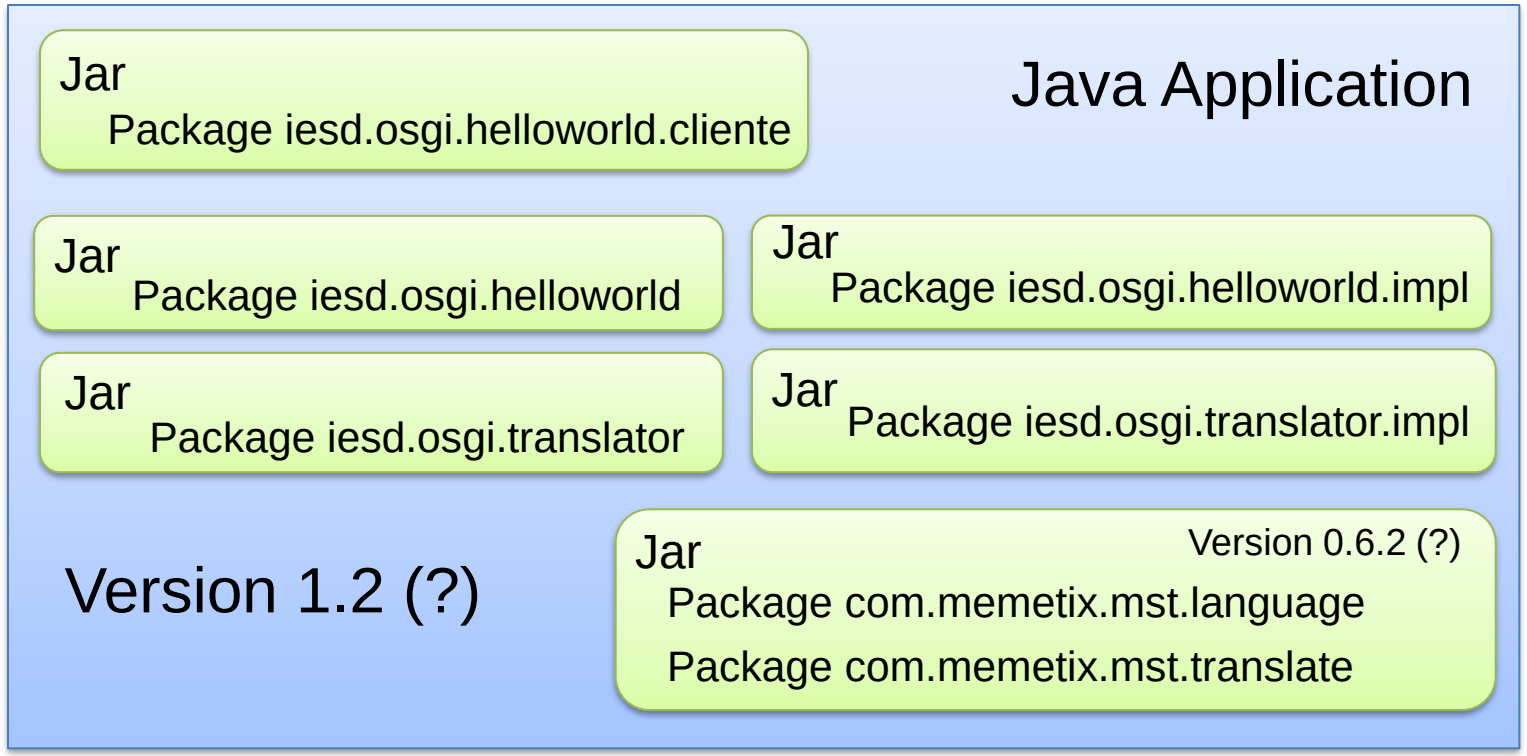
Connect to your task and ALM tools or create a local task.

0 items selected

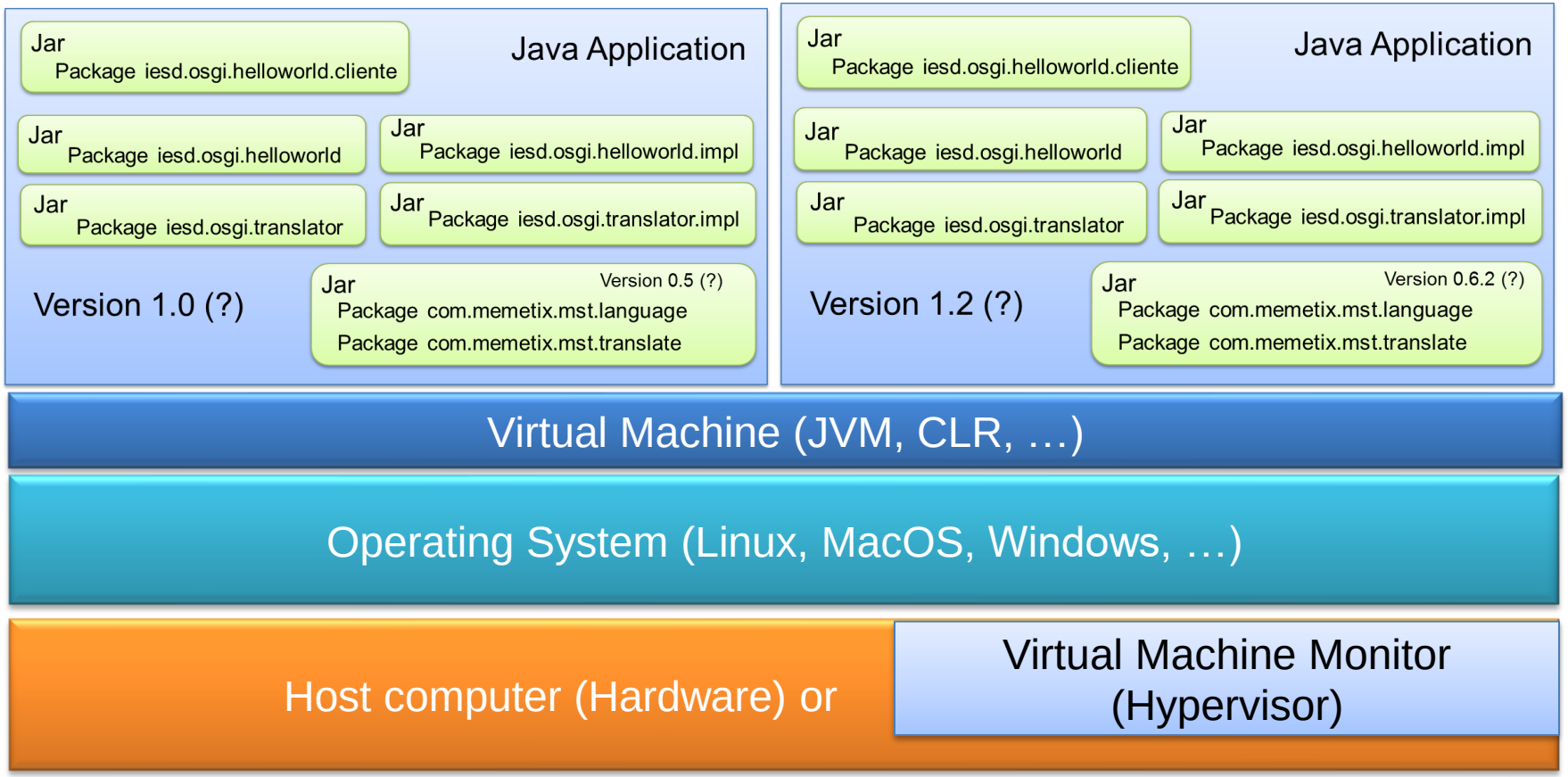
Java HelloWorld example



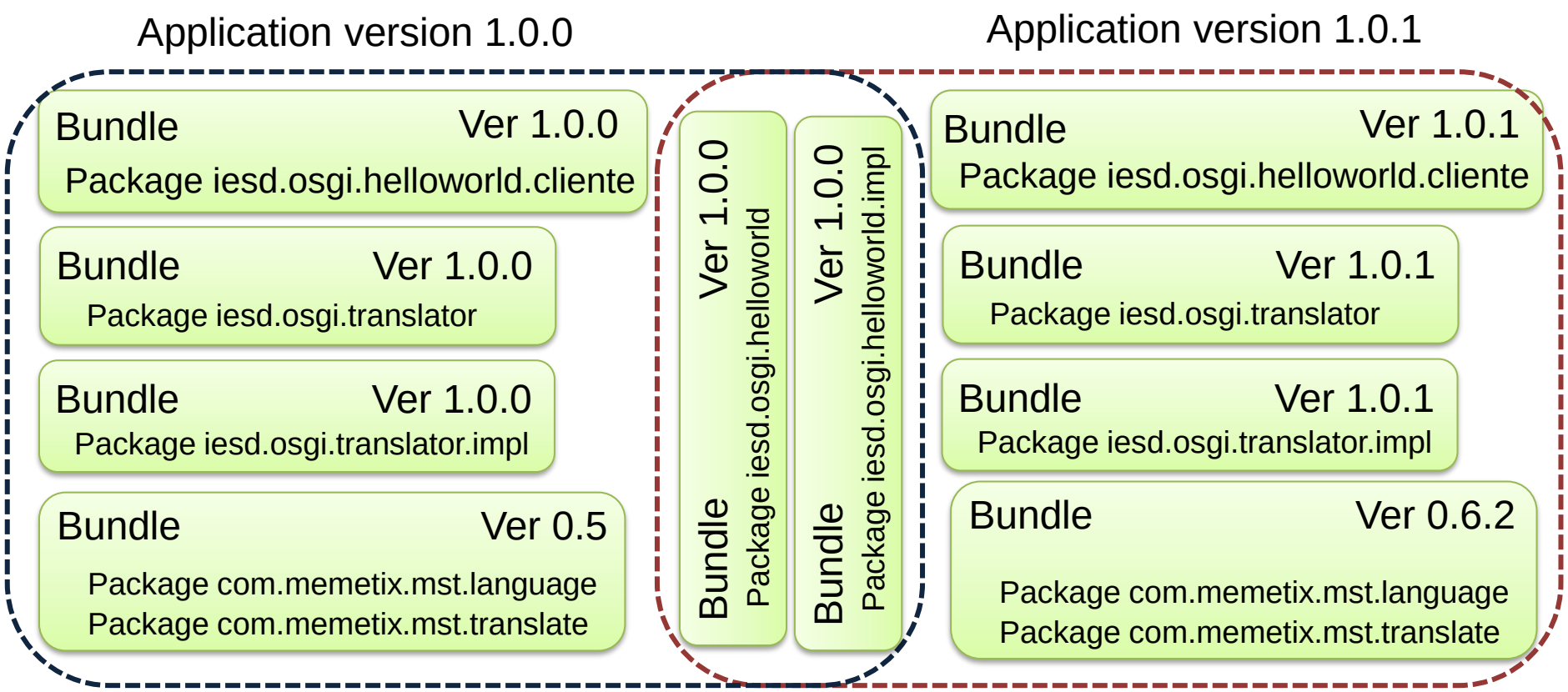
Java HelloWorld example



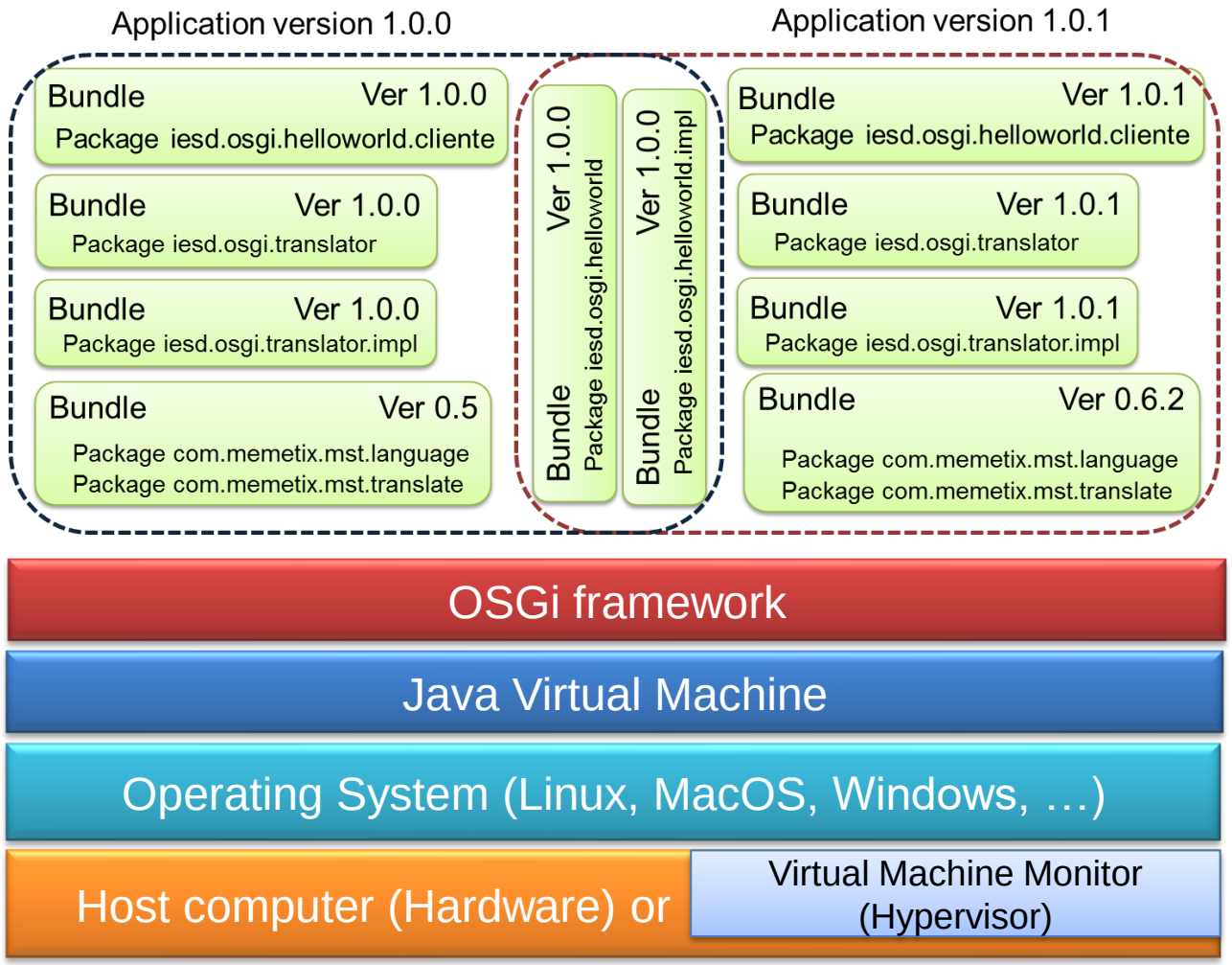
The Java (class/object based) modularity model



OSGi Bundle versions for the HelloWorld example

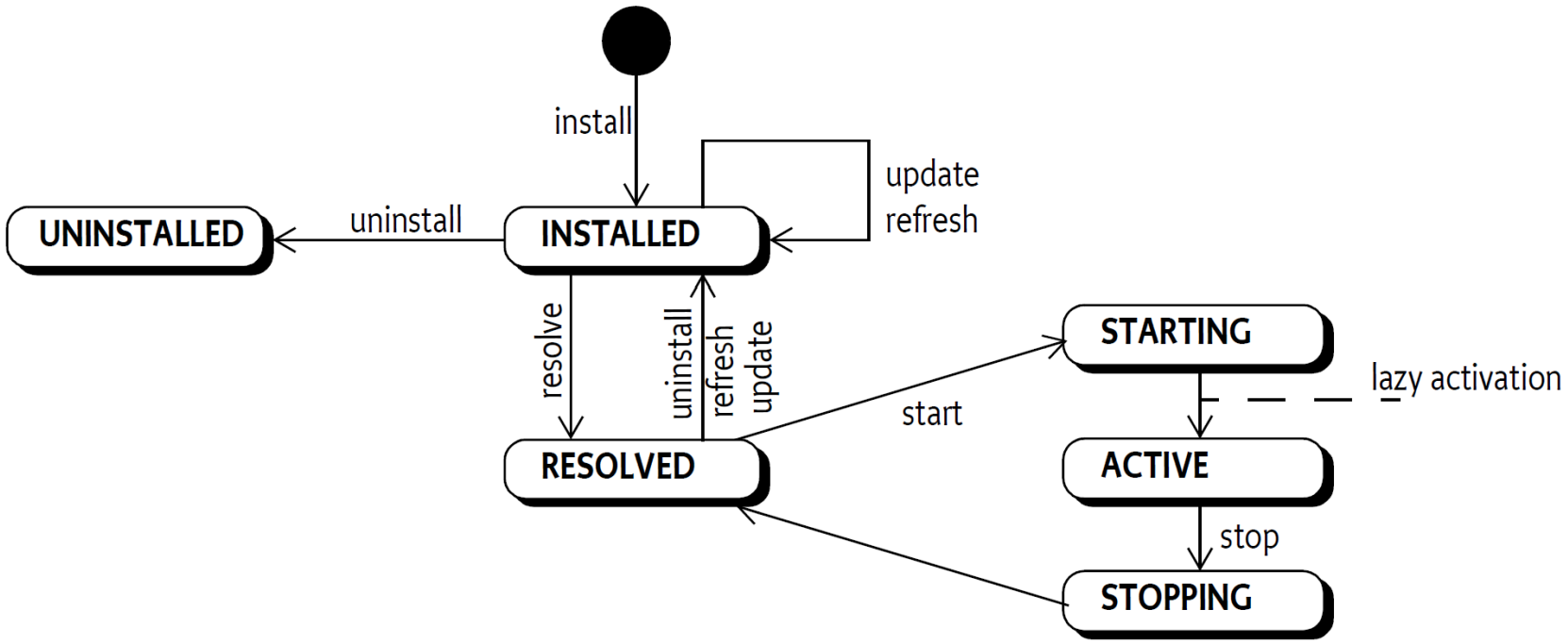


The OSGi Bunle versions



Bundle life cycle (versão 8.0)

Figure 4.4 *State diagram Bundle*



From: https://docs.osgi.org/download/r8/osgi.core-8.0.0.pdf

Bundle life cycle (versão 8.0)

- Bundle states
  - UNINSTALLED
    - A Bundle is non on a OSGI framework run-time
  - INSTALLED
    - A Bundle is on a OSGi runtime
  - RESOLVED
    - Bundle (identifiable) dependencies are resolved
  - STARTING
    - Bundle is under starting state
  - STOPPING
    - Bundle is under stopping state
  - ACTIVE
    - Bundle is in active state meaning the BundleActivator start() method was called
- Active threads of execution only in the states
  - STARTING, ACTIVE, or STOPPING