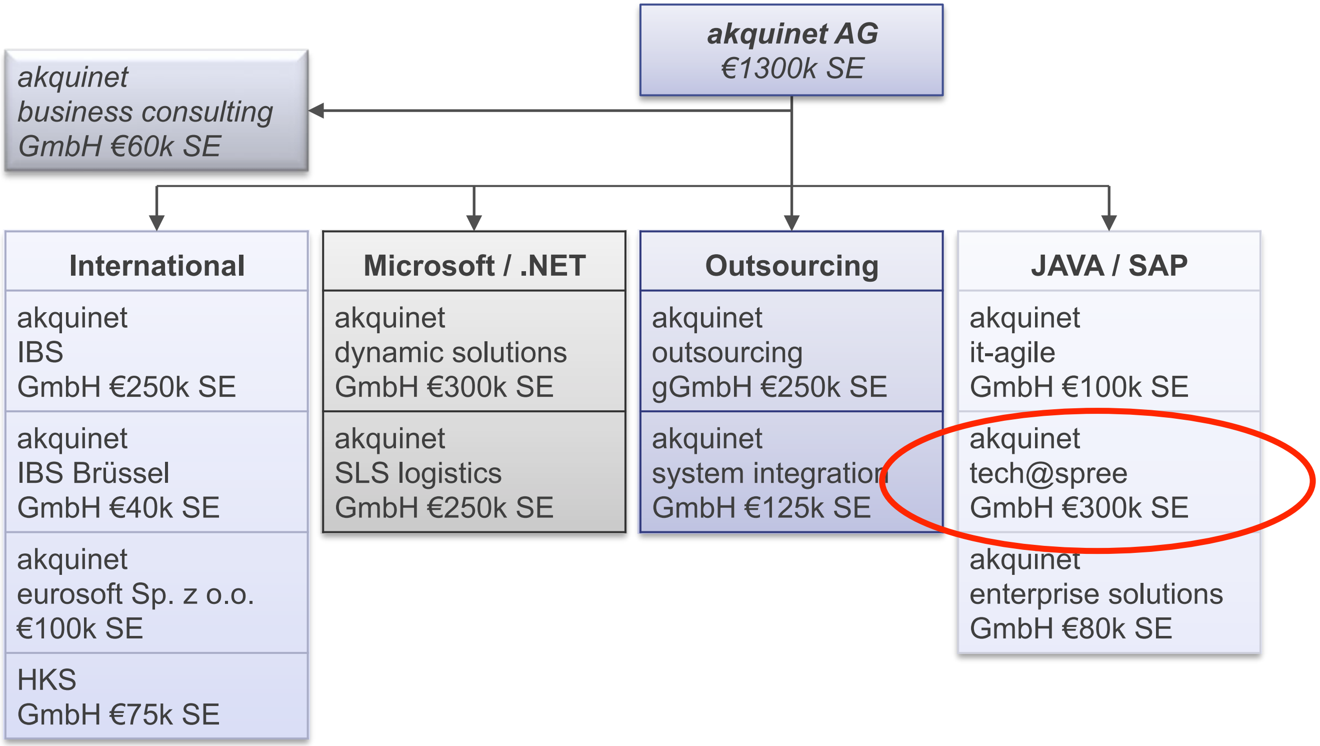
OSGi™ : Beyond the Myth

Clement Escoffier, akquinet A.G.

What about me ?

- Solution Architect in the Modular and Mobile CC
- Apache Software Foundation
  - PMC Apache Felix, Apache Ace
  - Apache Felix iPOJO project leader
- OW2
  - Chameleon project leader
- A lot of others contributions
  - maven-android-plugin
  - H-UBU

akquinet



akquinet AG
€1300k SE

akquinet
business consulting
GmbH €60k SE

International

akquinet
IBS
GmbH €250k SE

akquinet
IBS Brüssel
GmbH €40k SE

akquinet
eurosoft Sp. z o.o.
€100k SE

HKS
GmbH €75k SE

Microsoft / .NET

akquinet
dynamic solutions
GmbH €300k SE

akquinet
SLS logistics
GmbH €250k SE

Outsourcing

akquinet
outsourcing
gGmbH €250k SE

akquinet
system integration
GmbH €125k SE

JAVA / SAP

akquinet
it-agile
GmbH €100k SE

akquinet
tech@spree
GmbH €300k SE

akquinet
enterprise solutions
GmbH €80k SE

Modular and Mobile Solutions

Competence Center focusing on

- Modular Systems
  - Modularization expertise
  - OSGi-based
  - Sophisticated, Large scale, Distributed systems
- Mobile Solutions
  - *In the large*
    - Mobile devices, Interactions middleware, Server-side …
    - M2M, B2B

Open Technologies

- OSGi (Apache Felix, Apace Ace, OW2 Chameleon, Apache Sling…)
- Android
- Apache Maven
- Java EE (JBOSS, OW2 JOnAS)

Offering

Architecture, Consulting, Training and Mentoring on

- Systems using OSGi and/or mobile devices
- Modularization
- Development infrastructure, Build process
- Remote management, Provisioning solutions

Project realization

- Machine to Machine applications
  - RFID, Device interaction, Data collection, Control-loop
- Mediation / Integration
  - Data processing and mediation, Horizontal Mediation (ESB)
- Mobile applications
  - Android
- Desktop applications
  - User experience
- Web applications

OSGi in 2010

“OSGi No Thanks”, *MuleSoft*

Not sure what's happen in GlassFish v3 but it seems #Atmosphere users have trouble with it :-(..Works perfectly well in v2...I hate #OSGi!

Spending 2nd day trying just to upgrade lucene version. I hate OSGi, it makes simple things complex.

At least someone can say the king is naked. Thanks! Don't EJB2 story teach those guys any lesson? #OSGi #fail

Implementation Maturity

- Equinox, Felix
- New Specification in progress

Tools

- Injection Framework (iPOJO, Blueprint, SCR)
- IDE (bndtools, PDE)
- Build tools (maven, ant…)
- Tests (junit4osgi, pax:exam)

Eclipse

- Eclipse IDE
- P2, RCP …

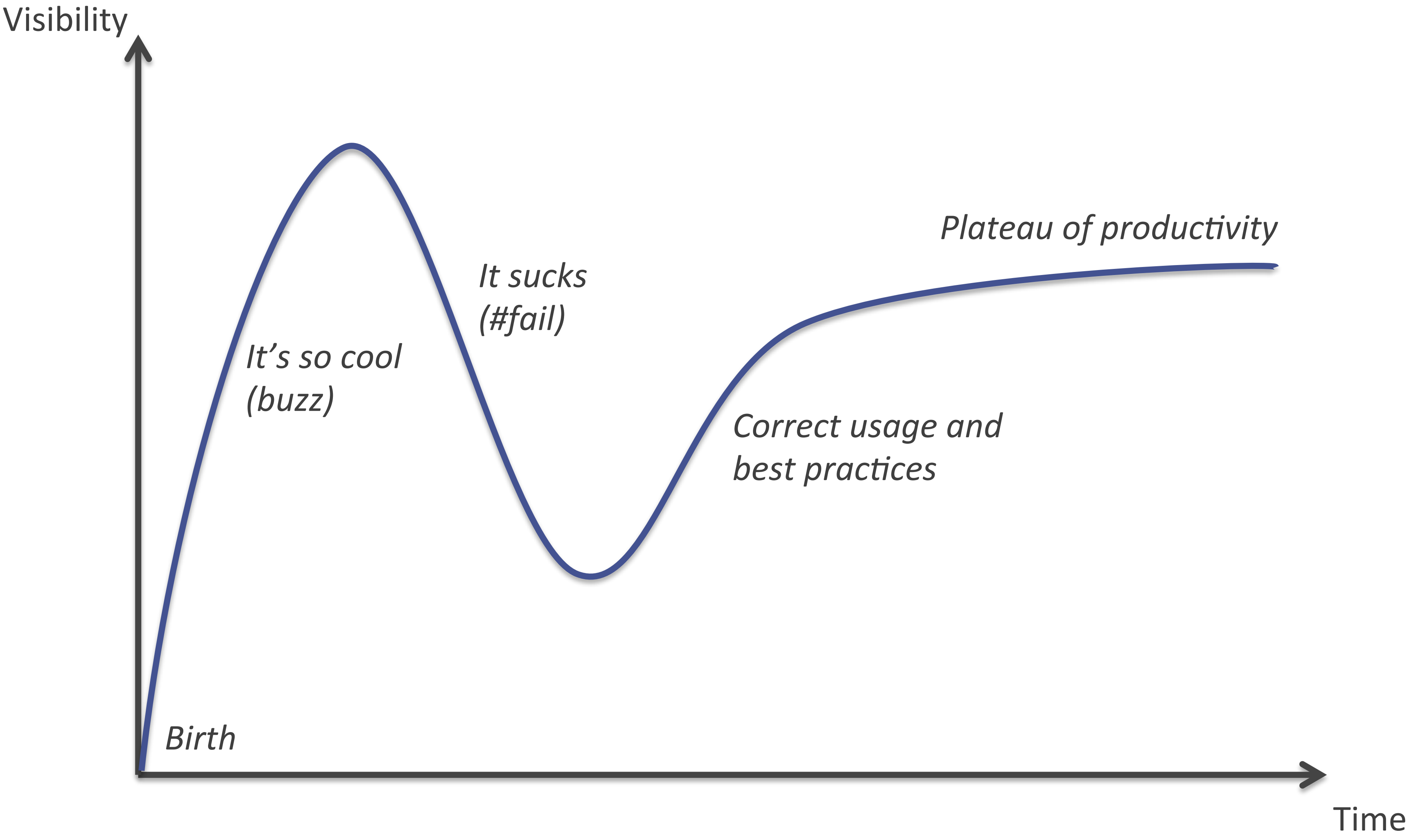
Application Servers

- Glassfish, JOnAS, Websphere
- Jboss A.S.

Others

- Service Mix / Fuse
- WSO2 (ESB, Integration)
- Sling

~~OSGi~~ Success Evolution



Why OSGi ?

My software is bigger than yours !

Java

- TCK: Over 1 Million LOC
- Harmony: 1.25 Million LOC

DVD player

- can contain 1 Million LOC

A BMW

- car can contain up to 50 networked computerized devices

Eclipse IDE

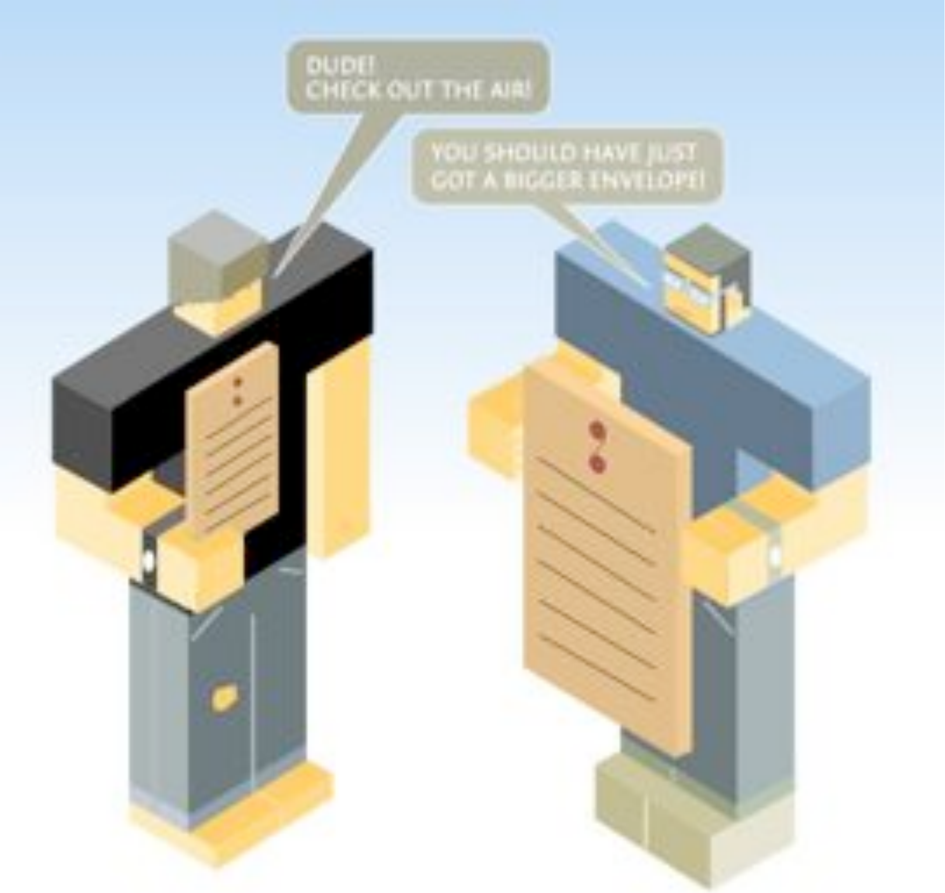
- 3.5 Million LOC

Space shuttle

- <0.5 Million lines

@ 10 lines a day

Libraries are a necessity, but …



Limits of OOP

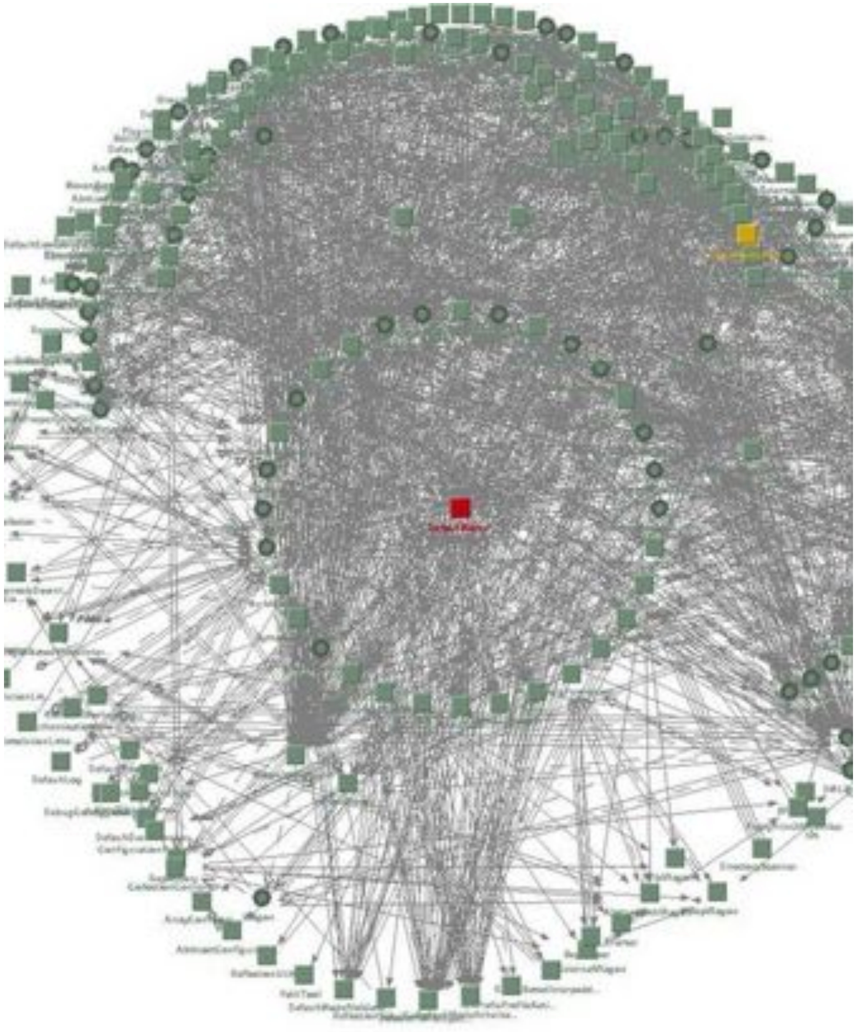
Coupling severely limits reusability

- Using a generic object, can drag in a large number of other objects

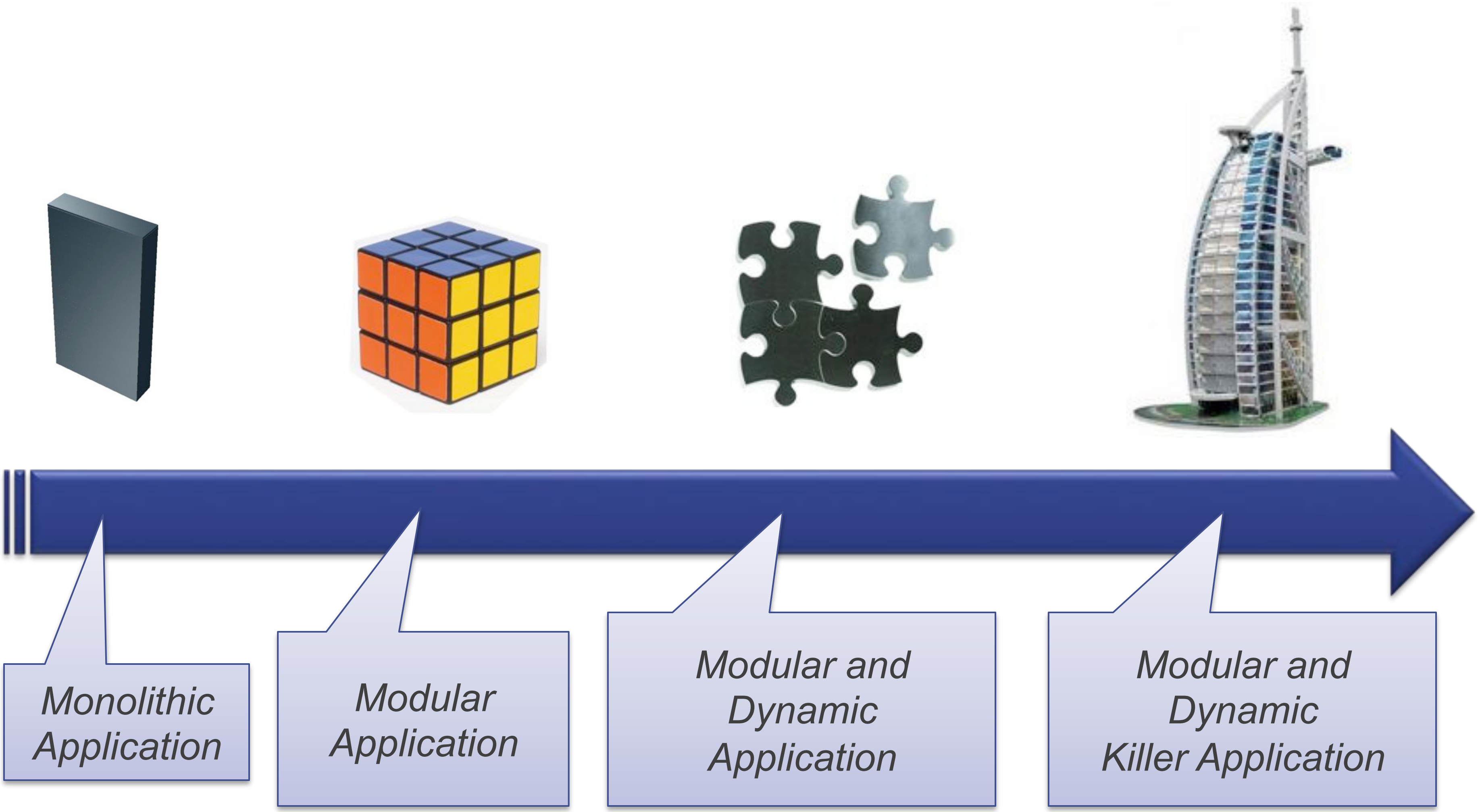
Creates overly large systems after a certain complexity is reached

Flexibility must be built in by the programmer

- Plugin architectures
- Factories, Dependency Injection



Modularize !



Monolithic Application

Modular Application

Modular and Dynamic Application

Modular and Dynamic Killer Application

Once upon a time, the modularity

- ~~JSR 277: Java Module System~~
- ~~JSR 294 : Improved Modularity in the Java Programming Language~~

- Jigsaw: Modularization of the JDK
  - May or may not be standardized
  - Java SE 8 ?
  - Should we really wait, or do we have something already robust enough ?

Why OSGi ?

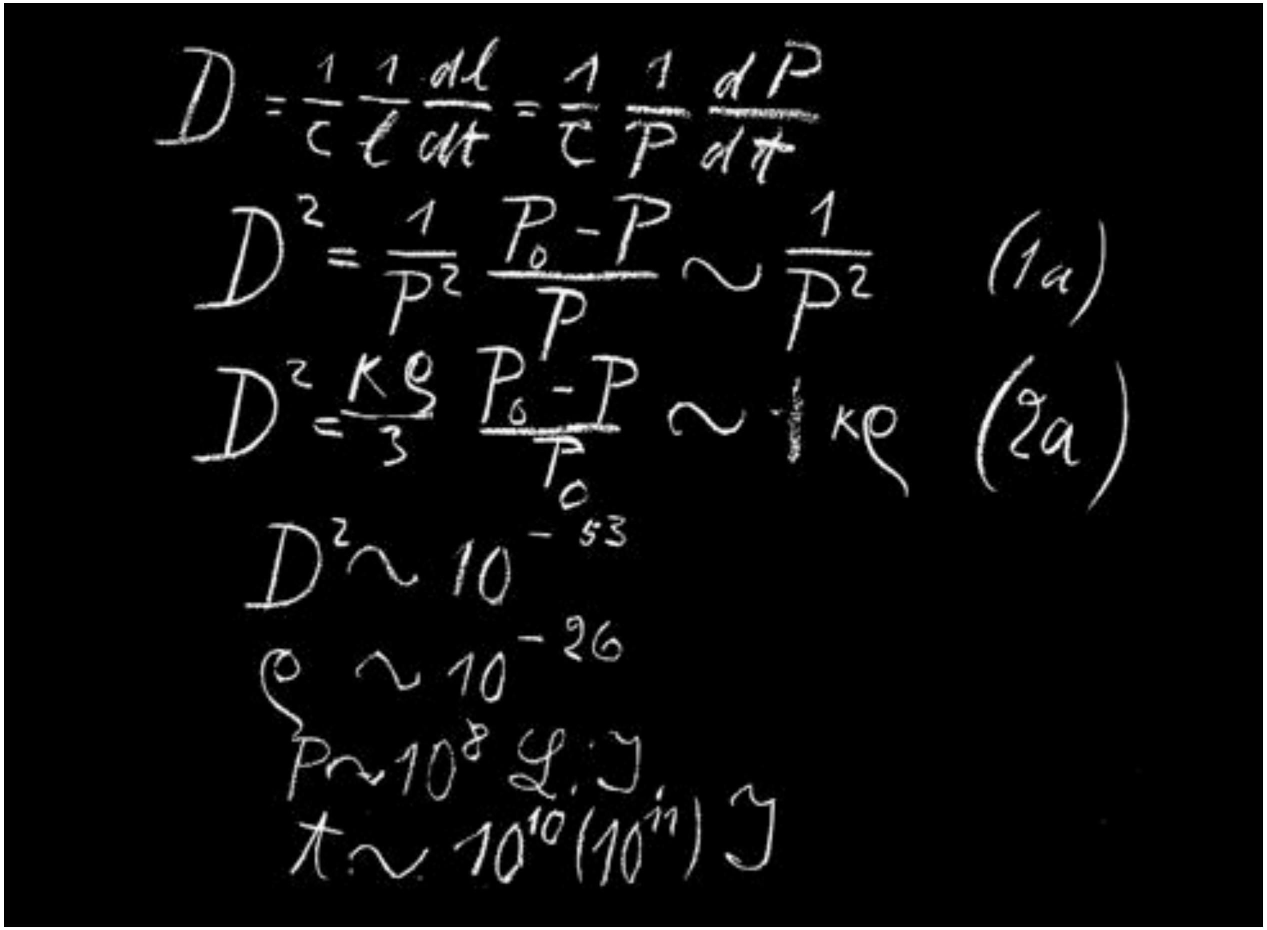
Need simpler ways to construct software systems

- OSGi is about **software construction**: building systems out of smaller components …
- OSGi is about **components that work together** …
- OSGi is about **managing and updating** components …
- OSGi is about “Universal **Middleware**”

Why OSGi ?

Need simpler ways to construct software systems

- OSGi is about **software construction**: building systems out of smaller components …
- OSGi is about **components that work together** …
- OSGi is about **managing and updating** components …
- OSGi is about “Universal **Middleware**”



$$D = \frac{1}{c}\frac{1}{l}\frac{dl}{dt} = \frac{1}{c}\frac{1}{P}\frac{dP}{dt}$$

$$D^2 = \frac{1}{P^2}\frac{P_0 - P}{P} \sim \frac{1}{P^2} \qquad (1a)$$

$$D^2 = \frac{\kappa\rho}{3}\frac{P_0 - P}{P_0} \sim \frac{1}{3}\kappa\rho \qquad (2a)$$

$$D^2 \sim 10^{-53}$$

$$\rho \sim 10^{-26}$$

$$P \sim 10^{8}\ L.J.$$

$$t \sim 10^{10}\,(10^{11})\ J$$

What is OSGi ?

OSGi Alliance

Industry consortium

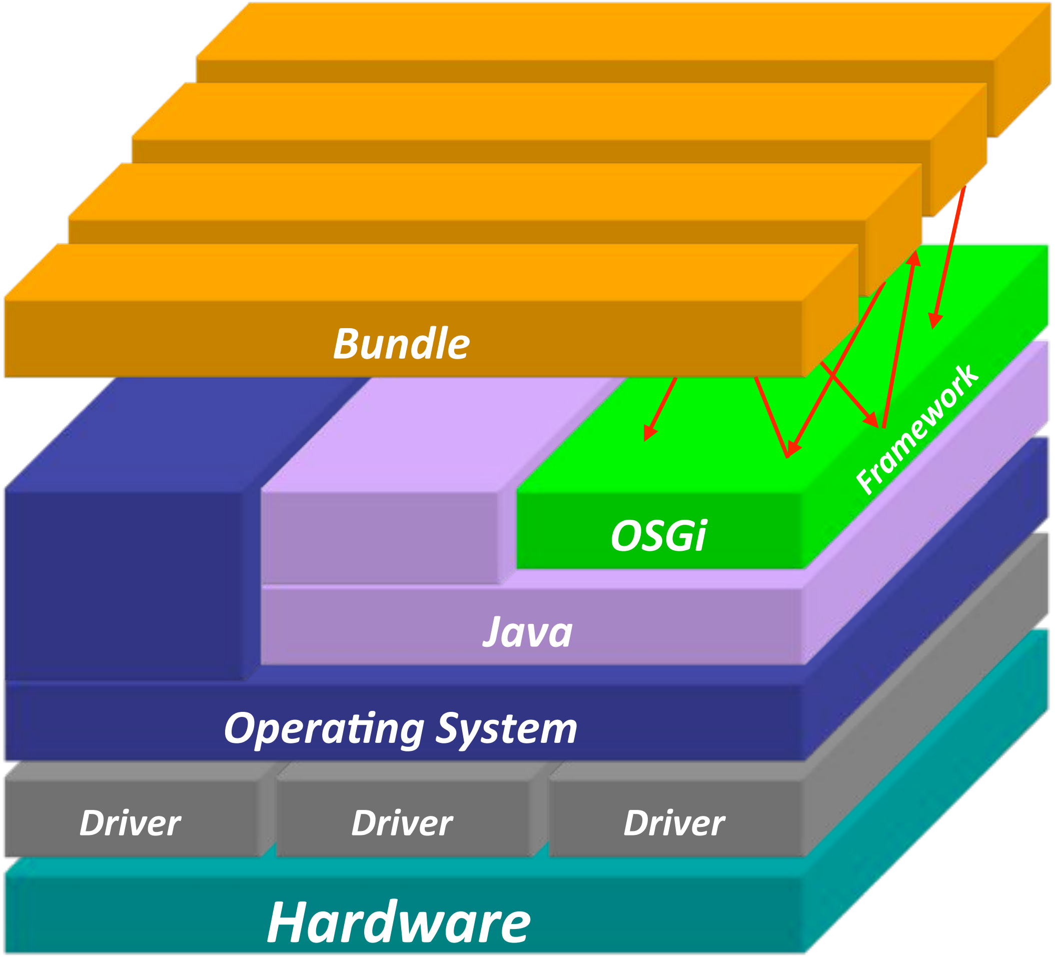
OSGi Service Platform specification

- Framework specification for hosting dynamically downloadable services
- Standard service specifications

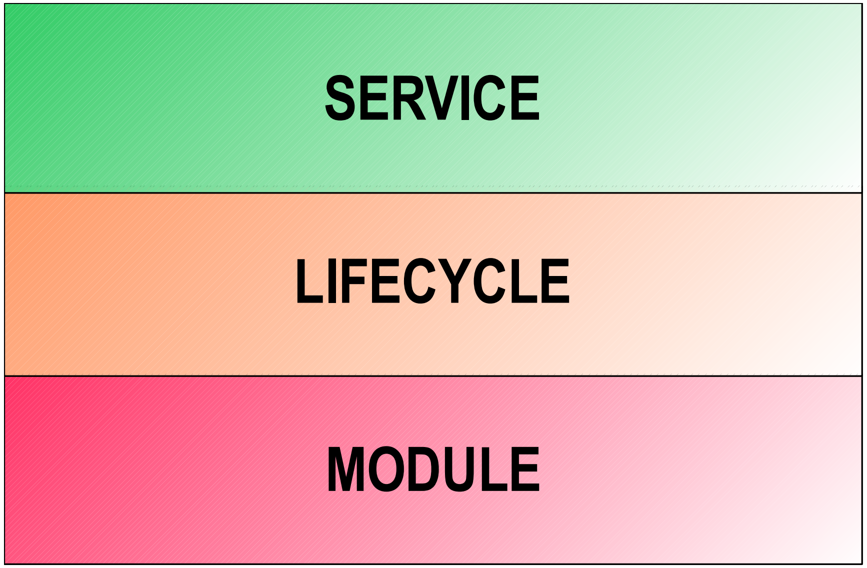
Several expert groups define the specifications

- Core Platform Expert Group (CPEG)
- Mobile Expert Group (MEG)
- Vehicle Expert Group (VEG)
- Enterprise Expert Group (EEG)

OSGi Architectural Overview



OSGi Framework Layering



L3 *– Provides a publish/find/bind service model to decouple bundles*

L2 *- Manages the lifecycle of bundle in a bundle repository without requiring the VM be restarted*

L1 *- Creates the concept of bundles that use classes from each other in a controlled way according to constraints*

OSGi Framework (1/2)

Component-oriented framework

- Bundles (i.e., modules/components)
- Package sharing and version management
- Life-cycle management and notification

Service-oriented architecture / computing

- Publish/find/bind intra-VM service model

Open remote management architecture

- No prescribed policy or protocol

OSGi Framework (2/2)

Runs multiple applications and services

Single VM instance

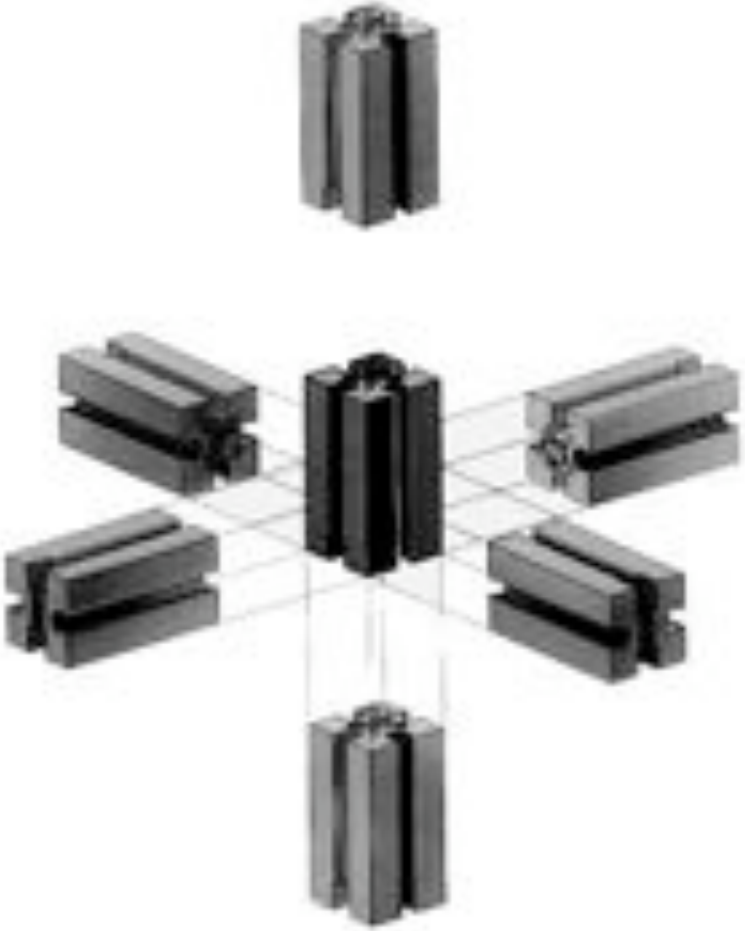
Separate class loader per bundle

- Class loader graph
- Independent namespaces
- Class sharing at the Java package level

Java Permissions to secure framework

Explicitly considers dynamic scenarios

- Run-time install and uninstall



The Module Layer

Modularity

What?

Modularity

What?

- Separation of concerns
- Structure
- Encapsulation
- Focuses on
  - Cohesion (low is bad, high is good)
  - Coupling (low is good, high is bad)

Why?

Modularity

What?

- Separation of concerns
- Structure
- Encapsulation
- Focuses on
  - Cohesion (low is bad, high is good)
  - Coupling (low is good, high is bad)

Why?

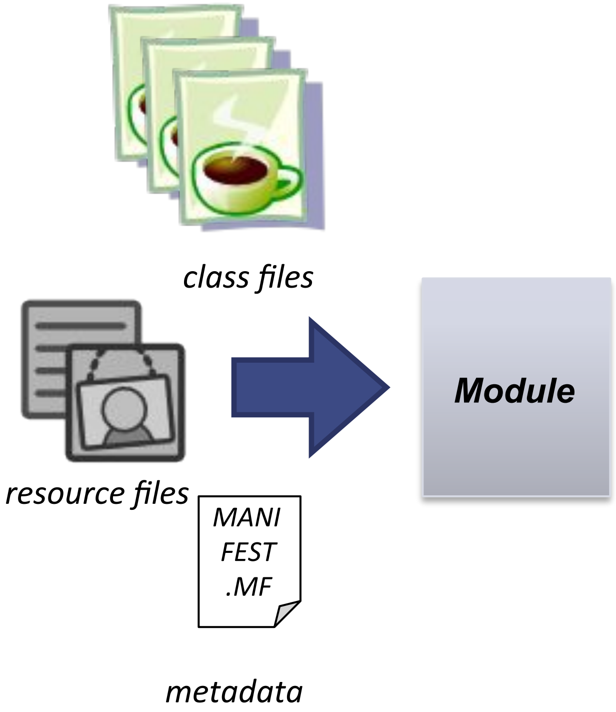
- Independent development
- Independent maintenance and evolution
- Improve reusability

OSGi Bundle

A bundle is a module in OSGi terminology

A bundle is a JAR file containing

- Code
- Resources
- Metadata



Code Visibility Metadata

A bundle is a JAR file containing code

- What code in the JAR file is visible to other code in the JAR file?
- What code in the JAR file is visible to code outside the JAR file?
- What code outside the JAR file is visible to code inside the JAR file?

Unlike standard JAR files, OSGi metadata explicitly answers all of these questions

Internal Code Visibility

Internal code in standard JARs can see all root-relative packages

- Not the case with bundles

Internal Code Visibility

Internal code in standard JARs can see all root-relative packages

- Not the case with bundles

Bundles must specify Bundle-ClassPath

- Comma-delimited list indicating where to search in the JAR file when looking for classes

Internal Code Visibility

Internal code in standard JARs can see all root-relative packages

- Not the case with bundles

Bundles must specify Bundle-ClassPath

- Comma-delimited list indicating where to search in the JAR file when looking for classes

To get standard JAR behavior

- Bundle-ClassPath: .

Internal Code Visibility

Internal code in standard JARs can see all root-relative packages

- Not the case with bundles

Bundles must specify Bundle-ClassPath

- Comma-delimited list indicating where to search in the JAR file when looking for classes

To get standard JAR behavior

- Bundle-ClassPath: .

May also include embedded JARs and directories

Examples

- Bundle-ClassPath: lib/foo.jar,classes/
- Bundle-ClassPath: lib/foo.jar,.

Exposing Internal Code (1/2)

Standard JAR files expose all internal root-relative packages

- Not the case with bundles

Exposing Internal Code (1/2)

Standard JAR files expose all internal root-relative packages

- Not the case with bundles

Bundles must specify Export-Package

- List of packages from the bundle class path to expose
- Uses common OSGi syntax mentioned earlier

Why do this?

Exposing Internal Code (1/2)

Standard JAR files expose all internal root-relative packages

- Not the case with bundles

Bundles must specify Export-Package

- List of packages from the bundle class path to expose
- Uses common OSGi syntax mentioned earlier

Why do this?

- It separates internal visibility from external visibility
- In other words, it allows bundles to have private content

Accessing External Code (1/2)

Standard JARs implicitly see everything other class on the class path

- Not the case with bundles

Accessing External Code (1/2)

Standard JARs implicitly see everything other class on the class path

- Not the case with bundles

Bundles must specify Import-Package

- List of packages needed from other bundles
- Uses common OSGi syntax mentioned earlier

Accessing External Code (1/2)

Standard JARs implicitly see everything other class on the class path

- Not the case with bundles

Bundles must specify Import-Package

- List of packages needed from other bundles
- Uses common OSGi syntax mentioned earlier

Bundles must import every needed package not contained in the bundle itself, except java.*

Why do this?

Accessing External Code (1/2)

Standard JARs implicitly see everything other class on the class path

- Not the case with bundles

Bundles must specify Import-Package

- List of packages needed from other bundles
- Uses common OSGi syntax mentioned earlier

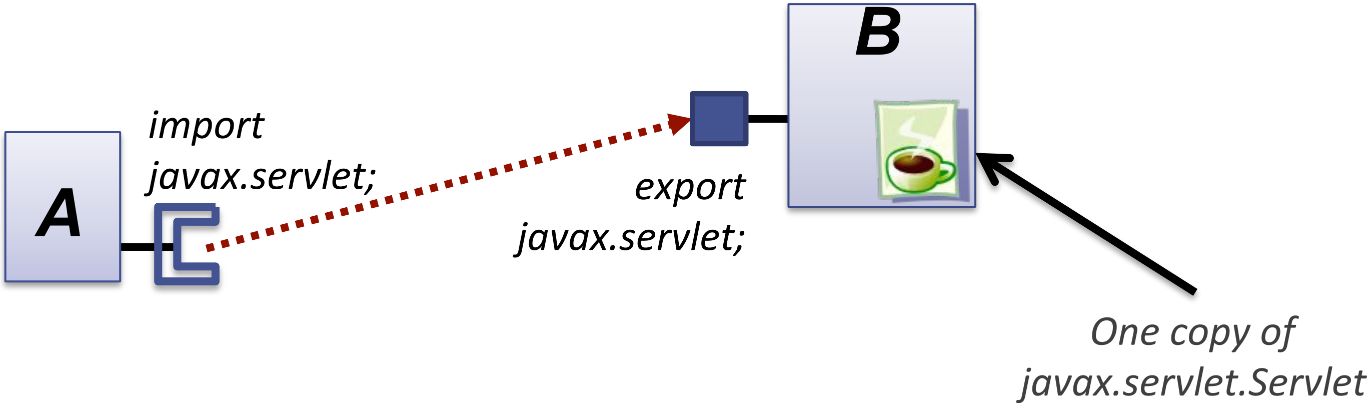
Bundles must import every needed package not contained in the bundle itself, except java.*

Why do this?

- Make dependencies **explicit**
- Make dependencies **manageable**

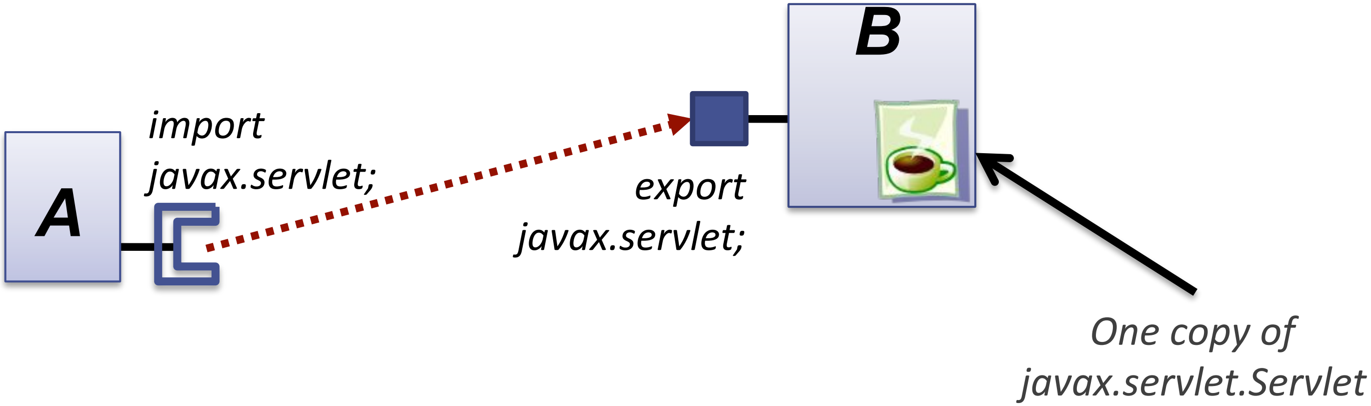
Importing & Exporting a Package (1/4)

Imagine bundle A somehow gets servlet instances from bundle B



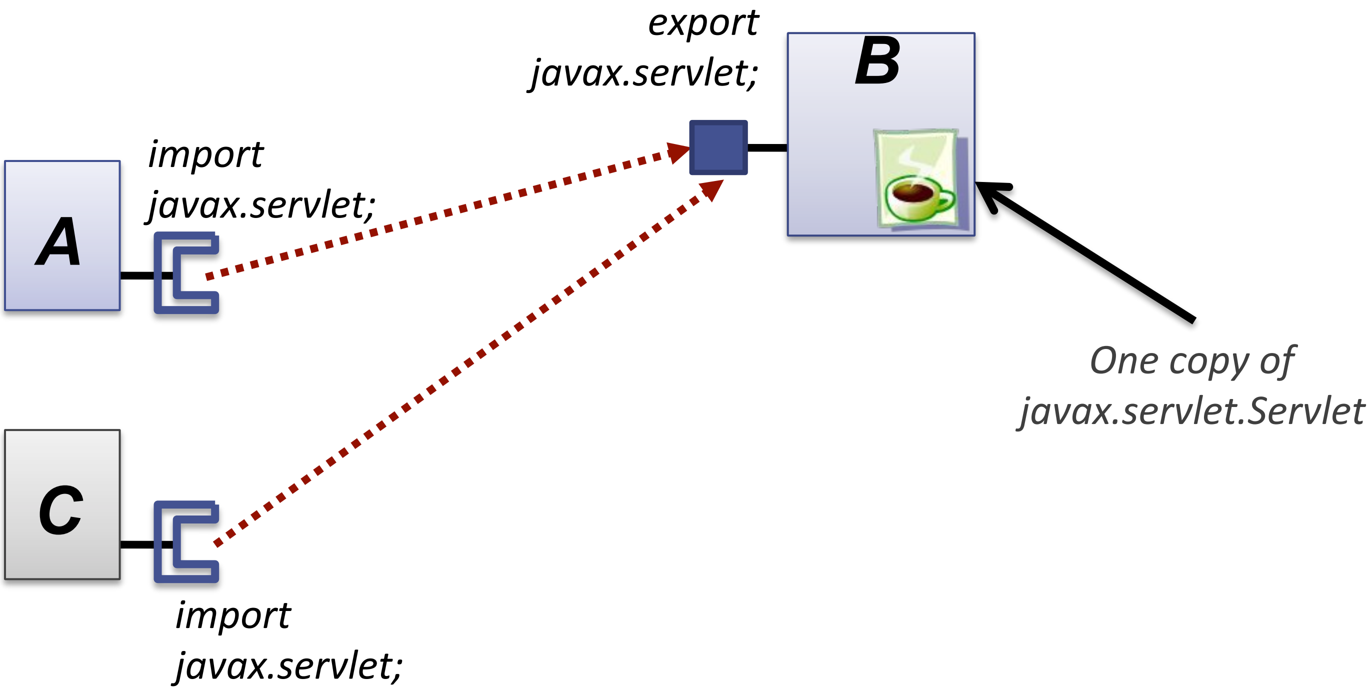
Importing & Exporting a Package (1/4)

What if bundle A also wanted to get servlet instances somehow from bundle C?



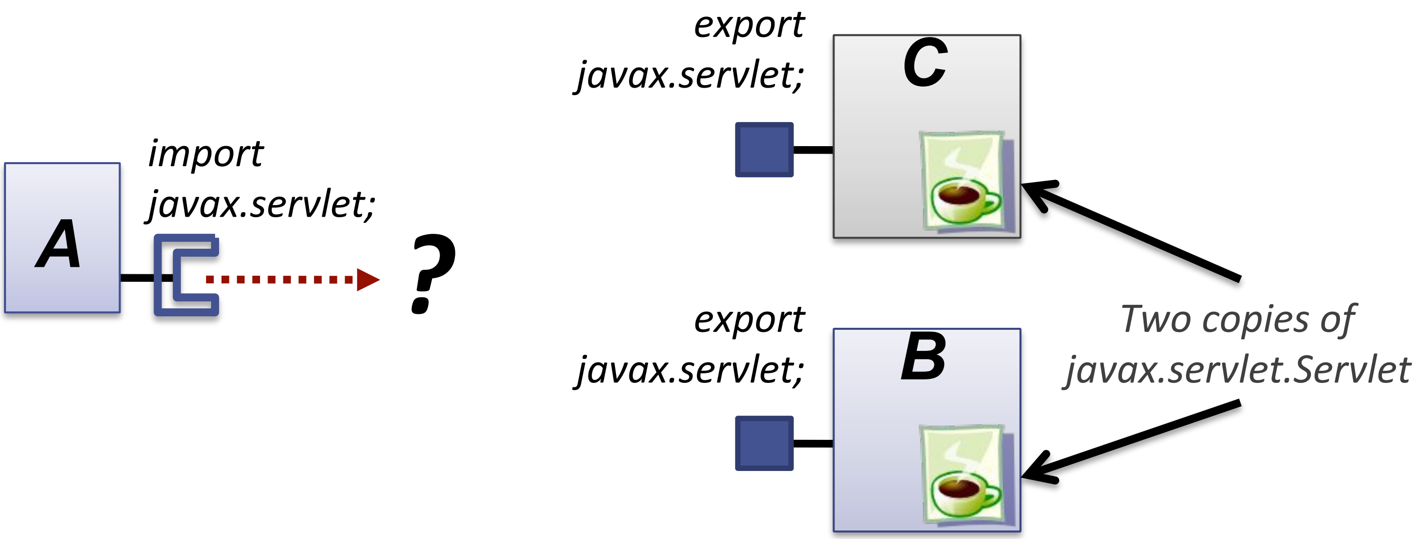
Importing & Exporting a Package (2/4)

Bundle C could import from bundle B, but then it is dependent on it



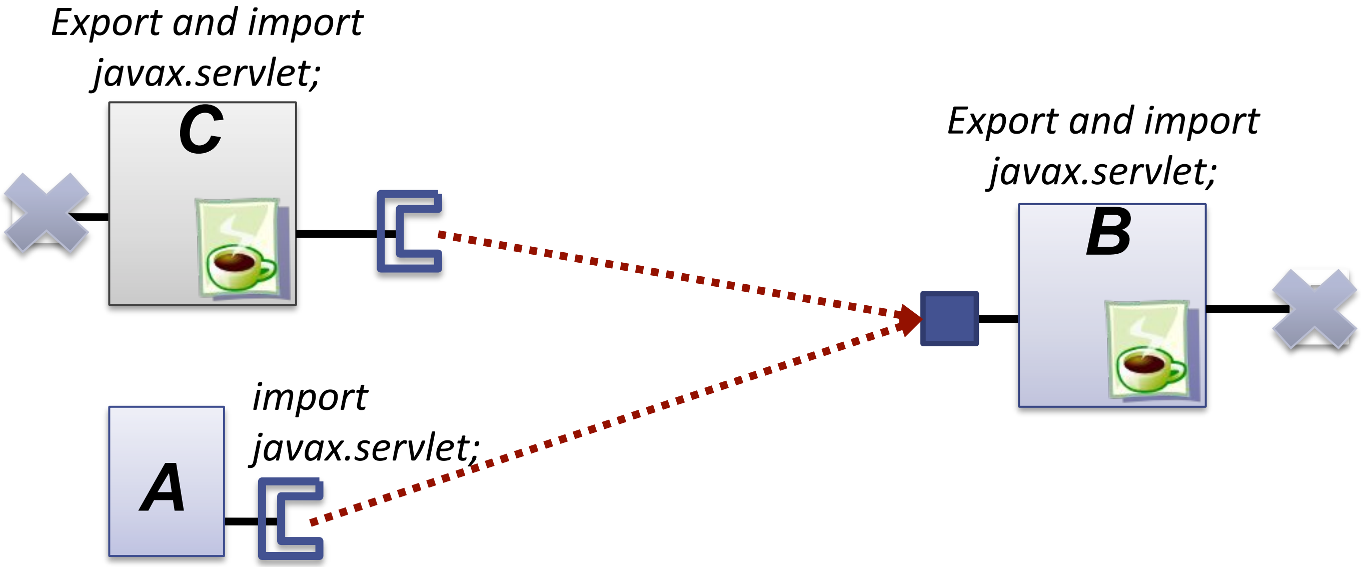
Importing & Exporting a Package (3/4)

Bundle C could export its own servlet package, but bundle A could only see either C or B



Importing & Exporting a Package (4/4)

Bundle C could import, contain and export servlet to solve the dilemma



Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1.0.0
Bundle-ClassPath: .,org/foo/embedded.jar
Import-Package:
 osgi.service.log; version="[1.0.0,1.1.0)",
 org.foo.service; version="1.1"
Export-Package:
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1.0.0
Bundle-ClassPat                oo/embedded.jar
Import-Packag
 osgi.service.                 "[1.0.0,1.1.0)",
 org.foo.service; version="1.1"
Export-Package:
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Indicates R4 semantics and syntax

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1.0.0
Bundle-ClassPath: .,org/foo/embedded.jar
Import-Package:
 osgi.service.log; v          "[1.0.0,1.1.0)",
 org.foo.service;                          "
Export-Package:
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Globally unique ID

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1.0.0
Bundle-ClassPath: .,org/foo/embedded.jar
Import-Package:
 osgi.service.lo     version="[1.0.0,1.1.0)",
 org.foo.                     n="1.1"
Export-Pa
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Internal bundle class path

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicN               mplebundle
Bundle-Version:
Bundle-ClassPath:              mbedded.jar
Import-Package:
 osgi.service.log; version="[1.0.0,1.1.0)",
 org.foo.service; version="1.1"
Export-Package:
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Import of a package version range

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1
Bundle-ClassPat                  dded.jar
Import-Package:
 osgi.service.log, version="[1.0.0,1.1.0)",
 org.foo.service; version="1.1"
Export-Package:
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Importing an exported package

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1.0.0
Bundle-ClassPath                    bedded.jar
Import-Package
 osgi.service.l                     0.0,1.1.0)",
 org.foo.service;                1.1"
Export-Package:
 org.foo.service; version="1.1";
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Exported package with version and arbitrary attribute

Bundle Manifest Example

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Version: 1.0.0
Bundle-ClassPath: .,org/foo/embedded.jar
Import-Package:
 osgi.service.log; ver                    1.1.0)",
 org.foo.service; ve
Export-Package:
 org.foo.service; versi                    ,
   vendor="org.foo",
 org.foo.service.bar; version="1.1";
   uses:="org.foo.service"
```

Provided package with dependency on exported package

OSGi Dependency Model

Package-level vs module-level dependencies

- Who vs what

OSGi Dependency Model

Package-level vs module-level dependencies

- Who vs what

Module-level dependencies

- Coarse grained
- Are brittle
- Hide the true dependencies

OSGi Dependency Model

Package-level vs module-level dependencies

- Who vs what

Module-level dependencies

- Coarse grained
- Are brittle
- Hide the true dependencies

Package-level dependencies

- Fine grained
- Flexible, enable refactoring
- Are the true dependencies (i.e., they're in the code)

OSGi Dependency Model

Package-level vs module-level dependencies

- Who vs what

Module-level dependencies

- Coarse grained
- Are brittle
- Hide the true dependencies

Package-level dependencies

- Fine grained
- Flexible, enable refactoring
- Are the true dependencies (i.e., they're in the code)

Package-level dependencies require packages to be atomic (i.e., in a single bundle)

Dependency Resolution (1/2)

Automatically managed by the OSGi framework

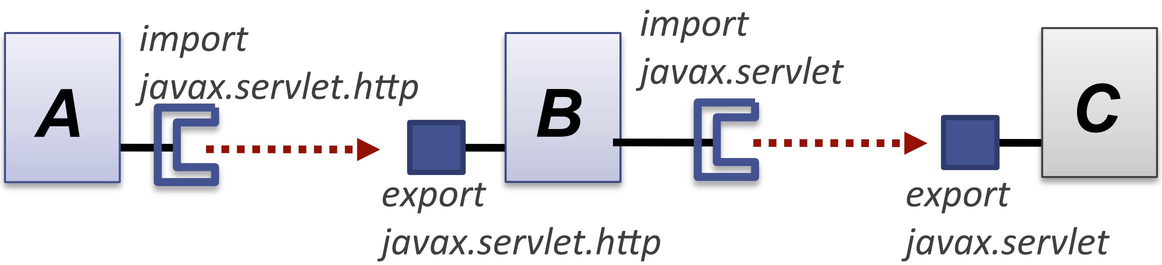
- Ensures a bundle's dependencies are satisfied before the bundle can be used

Dependency Resolution (1/2)

Automatically managed by the OSGi framework

- Ensures a bundle's dependencies are satisfied before the bundle can be used

In simple terms, resolving a bundle matches its imported packages to bundles providing them

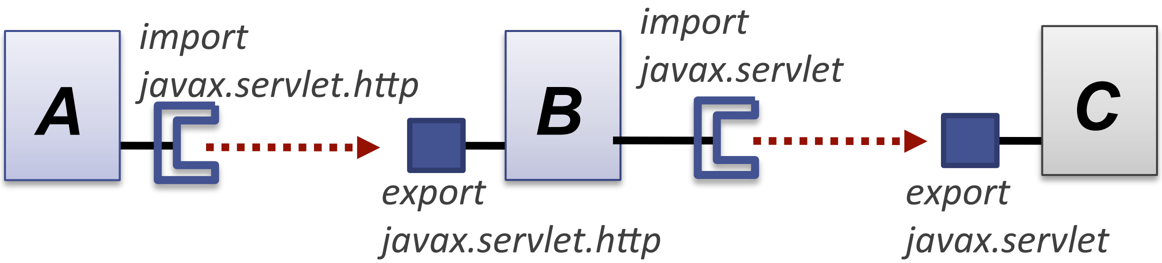


Dependency Resolution (1/2)

Automatically managed by the OSGi framework

- Ensures a bundle's dependencies are satisfied before the bundle can be used

In simple terms, resolving a bundle matches its imported packages to bundles providing them



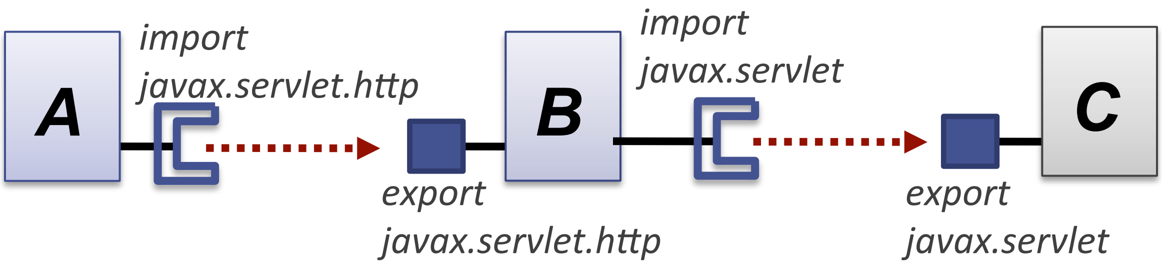
- Typically, resolving a bundle will result in other bundles being transitively resolved

Dependency Resolution (1/2)

Automatically managed by the OSGi framework

- Ensures a bundle's dependencies are satisfied before the bundle can be used

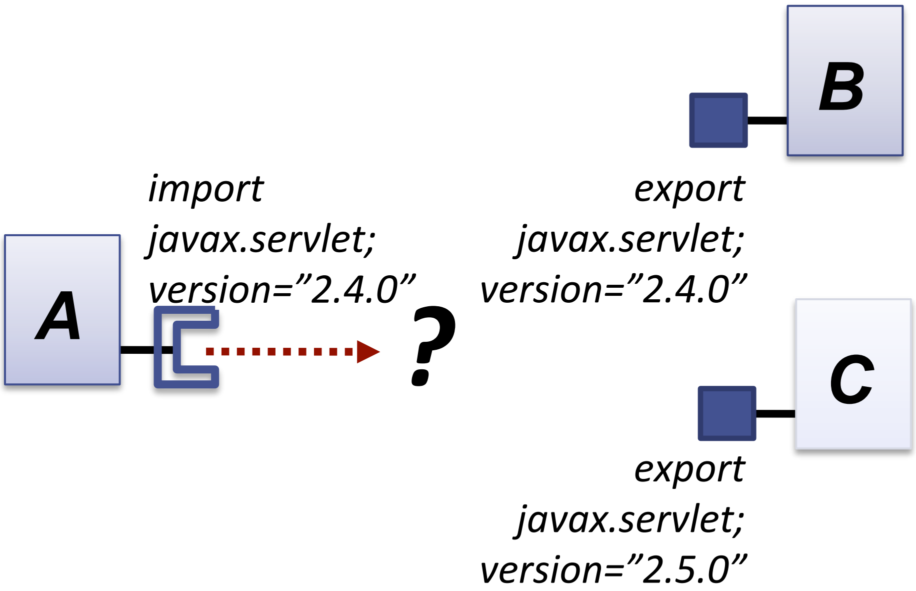
In simple terms, resolving a bundle matches its imported packages to bundles providing them



- Typically, resolving a bundle will result in other bundles being transitively resolved
- If a version or arbitrary attributes are specified on imports, then exports must match
  - Multiple attributes on an import are logically **ANDed**

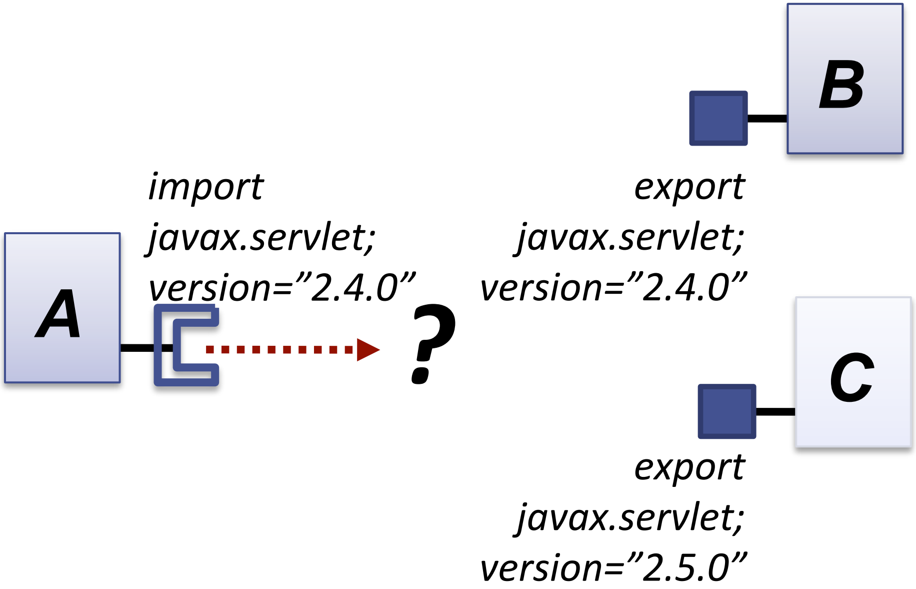
Dependency Resolution (2/2)

Multiple matching providers



Dependency Resolution (2/2)

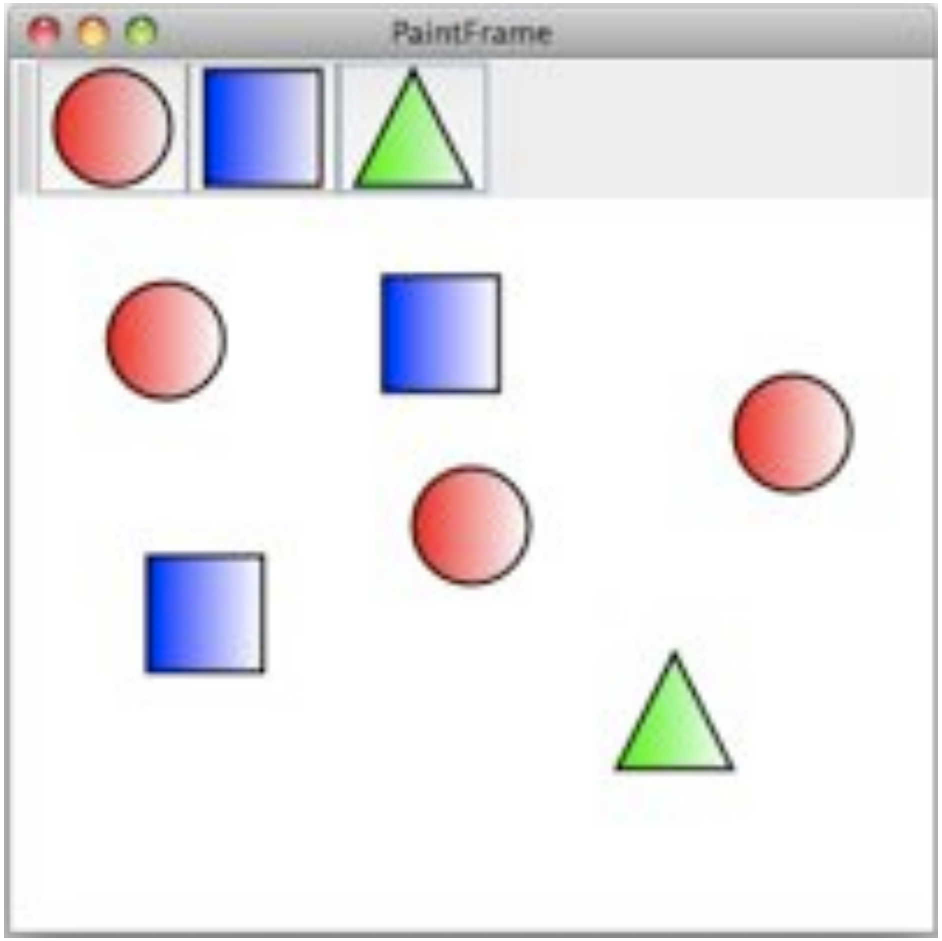
Multiple matching providers



- Resolution algorithm orders matching providers
  - Already resolved providers ordered by decreasing version
  - Unresolved providers ordered by decreasing version
  - If versions are equal, matching providers are ordered based on installation order

Non-Modular Paint Program (1/3)

We have a simple paint program



From the OSGi in Action book

Non-Modular Paint Program (2/3)

It is packaged as a single JAR file with the following contents:

```
META-INF/
META-INF/MANIFEST.MF
org/
org/foo/
org/foo/paint/
org/foo/paint/PaintFrame$1$1.class
org/foo/paint/PaintFrame$1.class
org/foo/paint/PaintFrame$ShapeActionListener.class
org/foo/paint/PaintFrame.class
org/foo/paint/SimpleShape.class
org/foo/paint/ShapeComponent.class
org/foo/shape/
org/foo/shape/Circle.class
org/foo/shape/circle.png
org/foo/shape/Square.class
org/foo/shape/square.png
org/foo/shape/Triangle.class
org/foo/shape/triangle.png
```

Non-Modular Paint Program (2/3)

It is packaged as a single JAR file with the following contents:

```
META-INF/
META-INF/MANIFEST.MF
org/
org/foo/
org/foo/paint/
org/foo/paint/PaintFrame$1$1.class
org/foo/paint/PaintFrame$1.class
org/foo/paint/PaintFrame$ShapeActionListener.class
org/foo/paint/PaintFrame.class
org/foo/paint/SimpleShape.class
org/foo/paint/ShapeComponent.class
org/foo/shape/
org/foo/shape/Circle.class
org/foo/shape/circle.png
org/foo/shape/Square.class
org/foo/shape/square.png
org/foo/shape/Triangle.class
org/foo/shape/triangle.png
```

Main implementation package is `org.foo.paint`

Non-Modular Paint Program (2/3)

It is packaged as a single JAR file with the following contents:

```
META-INF/
META-INF/MANIFEST.MF
org/
org/foo/
org/foo/paint/
org/foo/paint/PaintFrame$1$1.class
org/foo/paint/PaintFrame$1.class
org/foo/paint/PaintFrame$ShapeActi
org/foo/paint/PaintFrame.class
org/foo/paint/SimpleShape.class
org/foo/paint/ShapeComponent.class
org/foo/shape/
org/foo/shape/Circle.class
org/foo/shape/circle.png
org/foo/shape/Square.class
org/foo/shape/square.png
org/foo/shape/Triangle.class
org/foo/shape/triangle.png
```

Static main method in `PaintFrame`

Non-Modular Paint Program (2/3)

It is packaged as a single JAR file with the following contents:

```
META-INF/
META-INF/MANIFEST.MF
org/
org/foo/
org/foo/paint/
org/foo/paint/PaintFrame$1$1.class
org/foo/paint/PaintFrame$1.class
org/foo/paint/PaintFrame$ShapeAction
org/foo/paint/PaintFrame.class
org/foo/paint/SimpleShape.class
org/foo/paint/ShapeComponent.class
org/foo/shape/
org/foo/shape/Circle.class
org/foo/shape/circle.png
org/foo/shape/Square.class
org/foo/shape/square.png
org/foo/shape/Triangle.class
org/foo/shape/triangle.png
```

Interface `SimpleShape` supports multiple shape implementations

Non-Modular Paint Program (2/3)

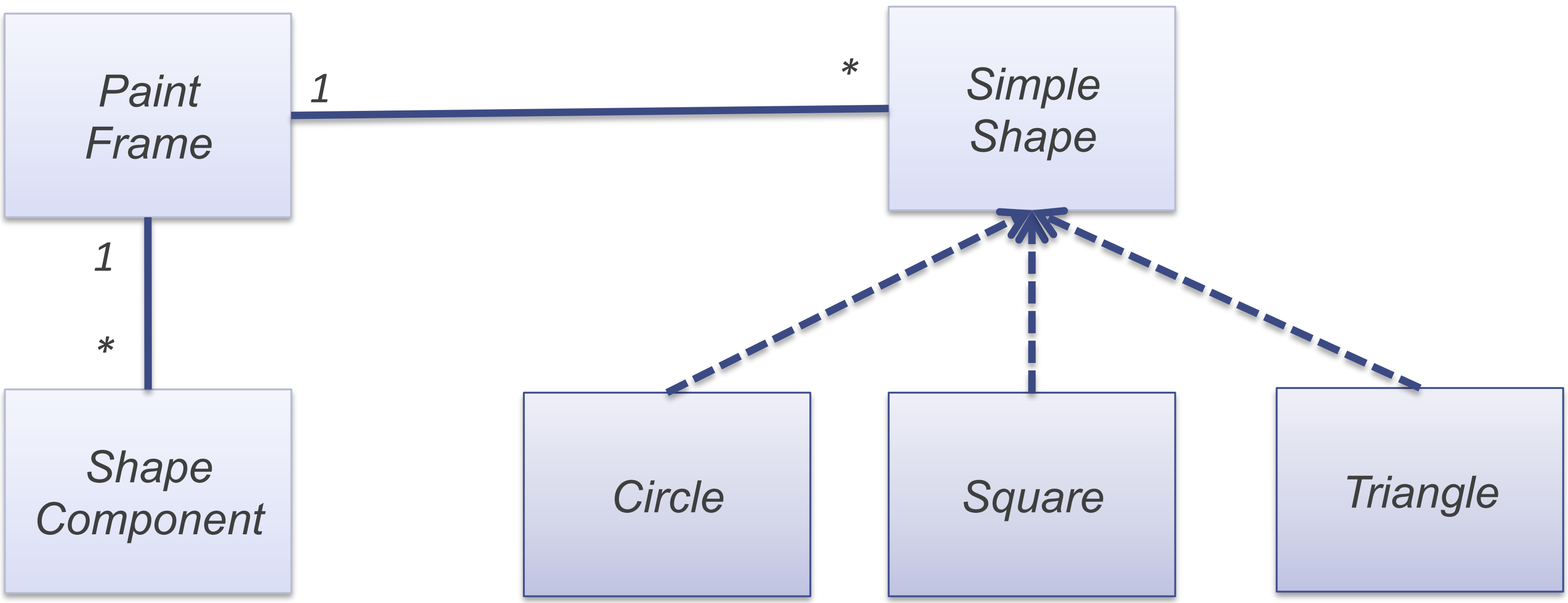
It is packaged as a single JAR file with the following contents:

```
META-INF/
META-INF/MANIFEST.MF
org/
org/foo/
org/foo/paint/
org/foo/paint/PaintFrame$1$1.class
org/foo/paint/PaintFrame$1.class
org/foo/paint/PaintFrame$ShapeActionListener.class
org/foo/paint/PaintFrame.class
org/foo/paint/SimpleShape.class
org/foo/paint/ShapeComponent.class
org/foo/shape/
org/foo/shape/Circle.class
org/foo/shape/circle.png
org/foo/shape/Square.class
org/foo/shape/square.png
org/foo/shape/Triangle.class
org/foo/shape/triangle.png
```

Shape implementations defined in `org.foo.shape`

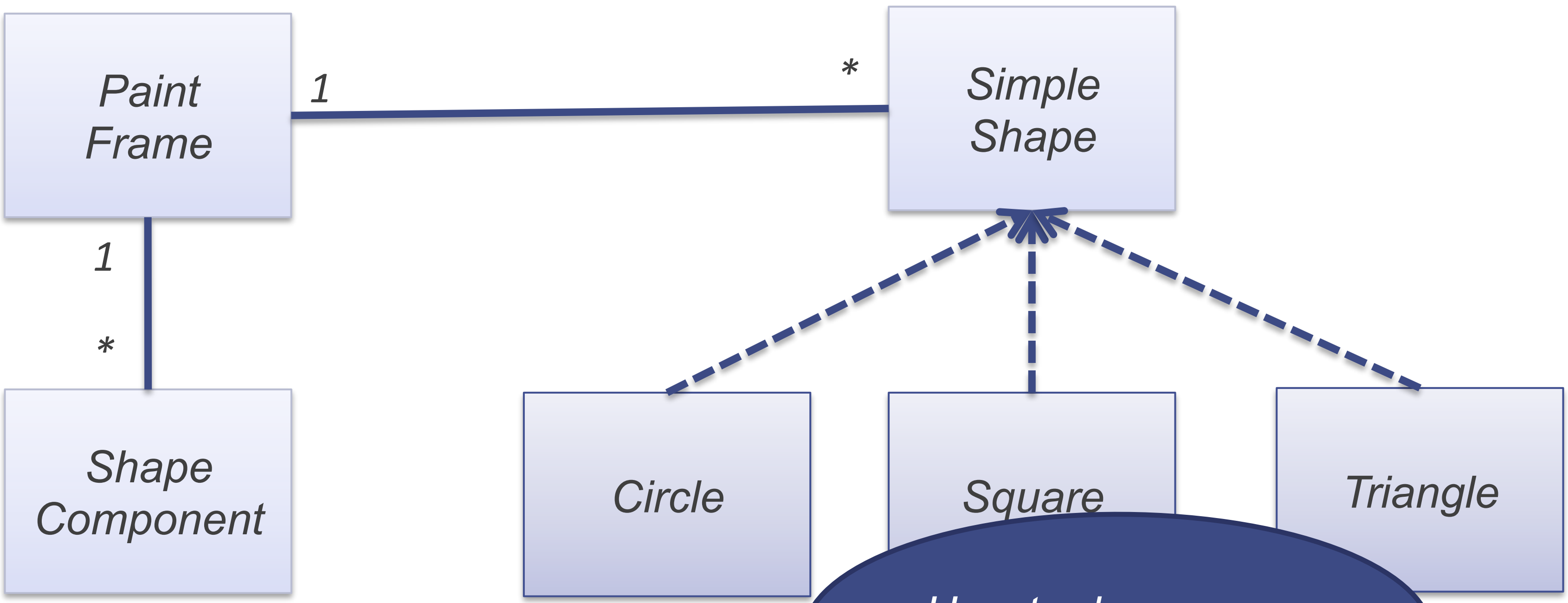
Non-Modular Paint Program (3/3)

Relationship among classes



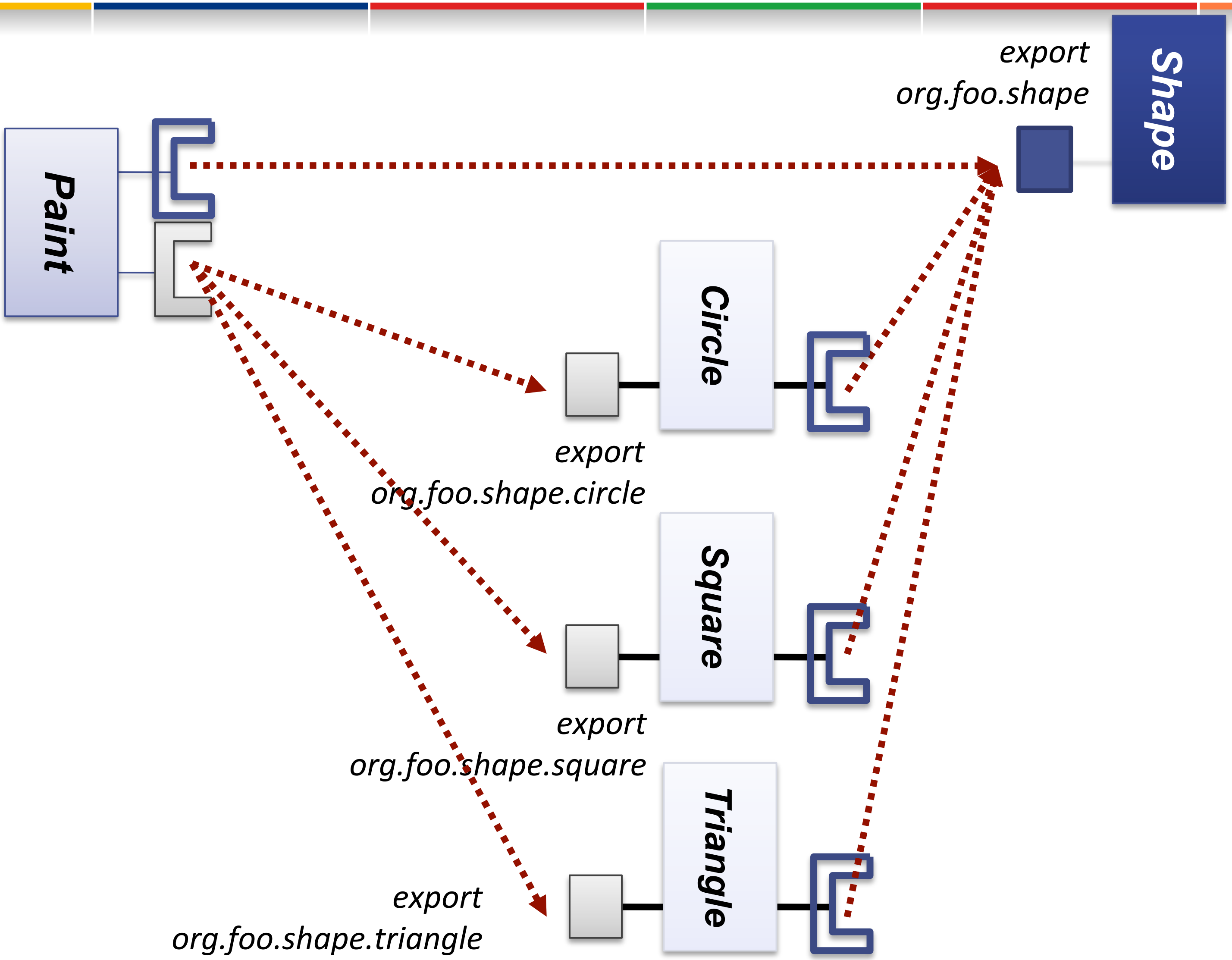
Non-Modular Paint Program (3/3)

Relationship among classes



How to decompose into bundles?

Modular Paint Program Design



export org.foo.shape

export org.foo.shape.circle

export org.foo.shape.square

export org.foo.shape.triangle

Benefits of Modularized Paint Program

Enforced logical boundaries

Automatic dependency resolution

- Ensures proper configuration

Improves reusability of code

Improves ability to create different configurations

The Lifecycle Layer

What & Why of Lifecycle Layer

Once we have a bundle, what do we do with it?

- We need to somehow tell the OSGi framework about it

What & Why of Lifecycle Layer

Once we have a bundle, what do we do with it?

- We need to somehow tell the OSGi framework about it

What if our bundle needs to be initialized somehow?

- We need some sort of hook in the framework

What & Why of Lifecycle Layer

Once we have a bundle, what do we do with it?

- We need to somehow tell the OSGi framework about it

What if our bundle needs to be initialized somehow?

- We need some sort of hook in the framework

What if we want to add and remove bundles at run time?

- We need someway to access the underlying framework

What & Why of Lifecycle Layer

Once we have a bundle, what do we do with it?

- We need to somehow tell the OSGi framework about it

What if our bundle needs to be initialized somehow?

- We need some sort of hook in the framework

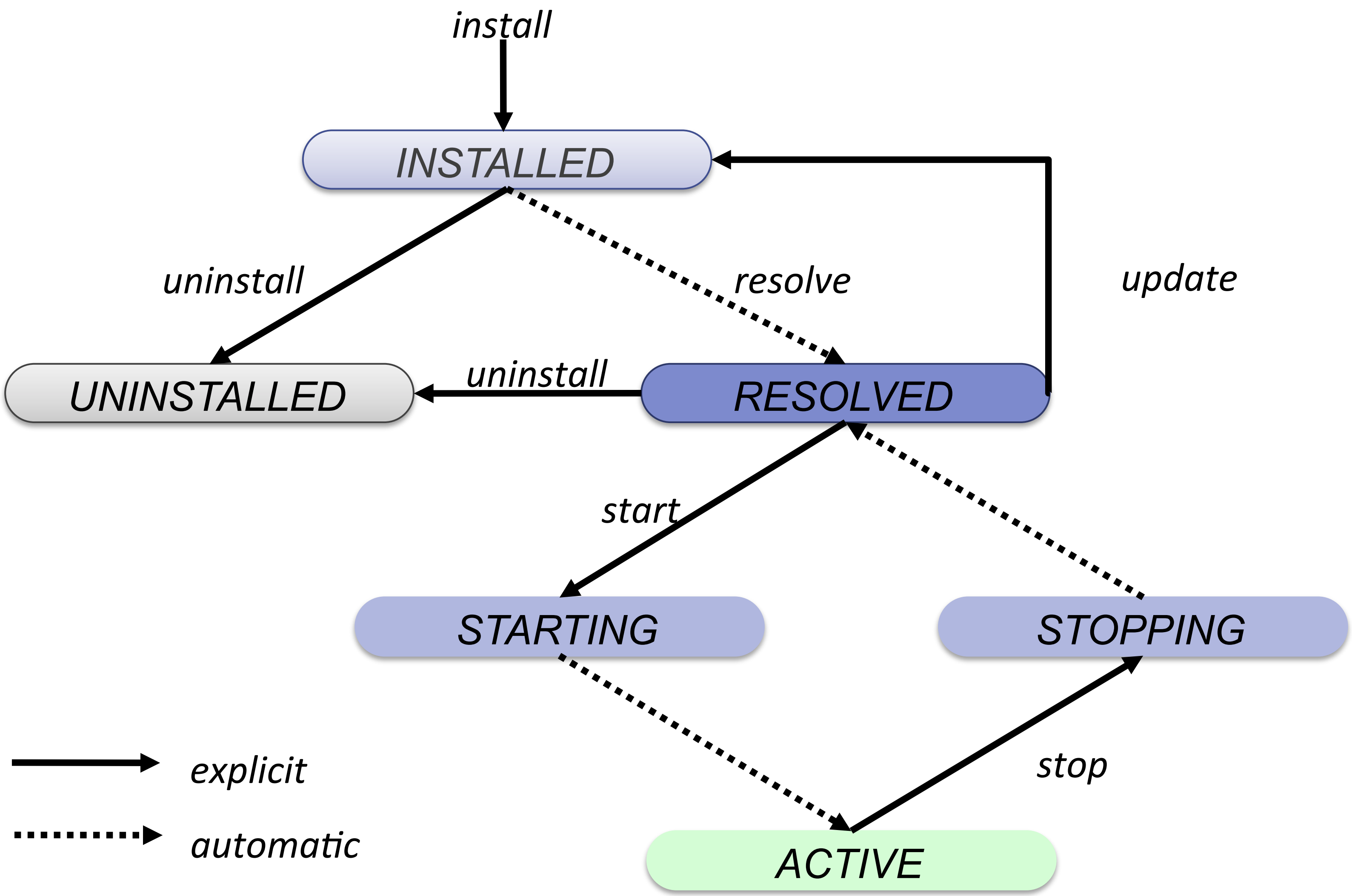
What if we want to add and remove bundles at run time?

- We need someway to access the underlying framework

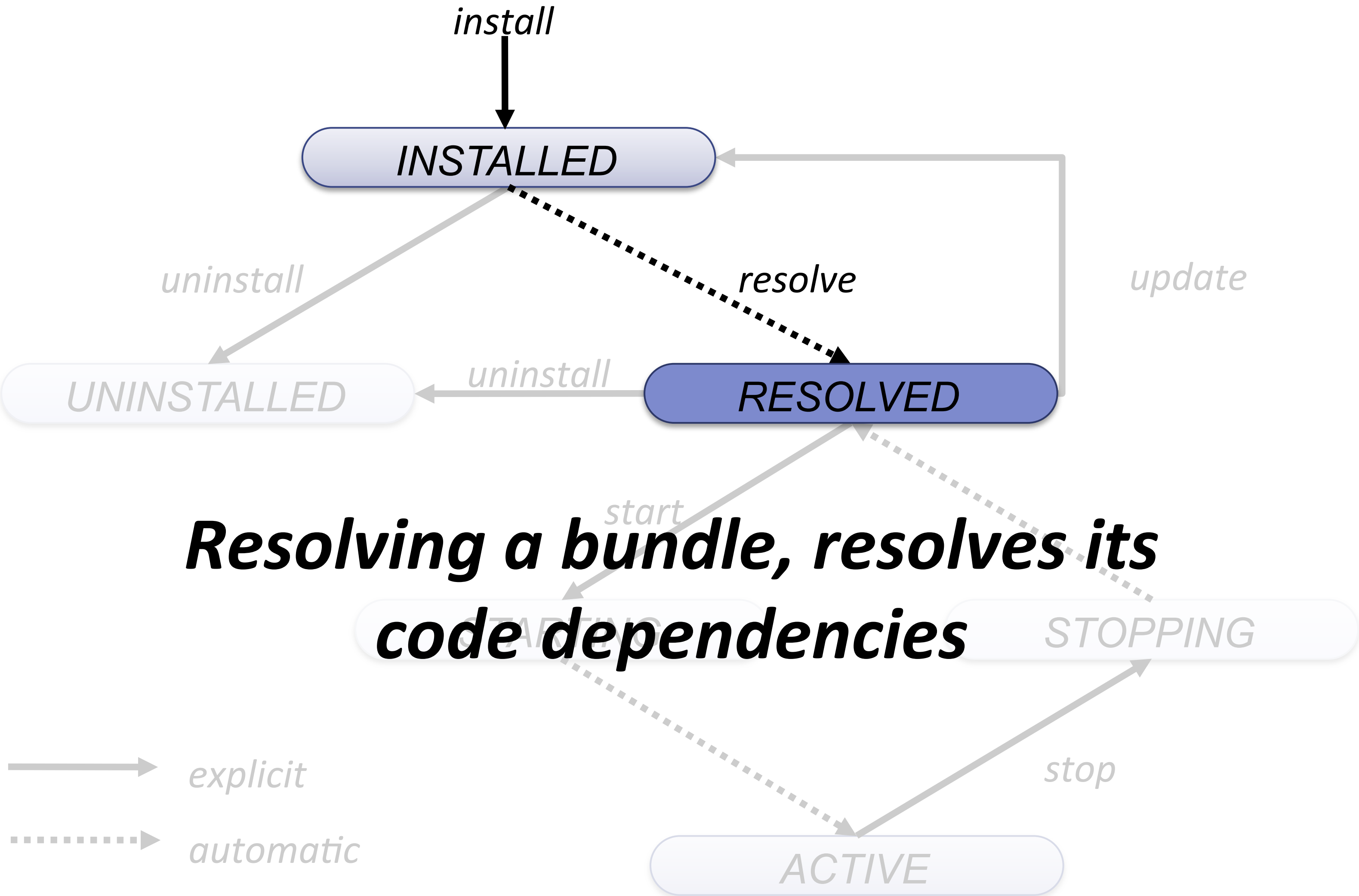
We can do all of these things with a well-defined lifecycle for bundles

- A lifecycle defines the stages of a bundle's lifetime
  - The framework associates a lifecycle state with each bundle

Bundle Life Cycle



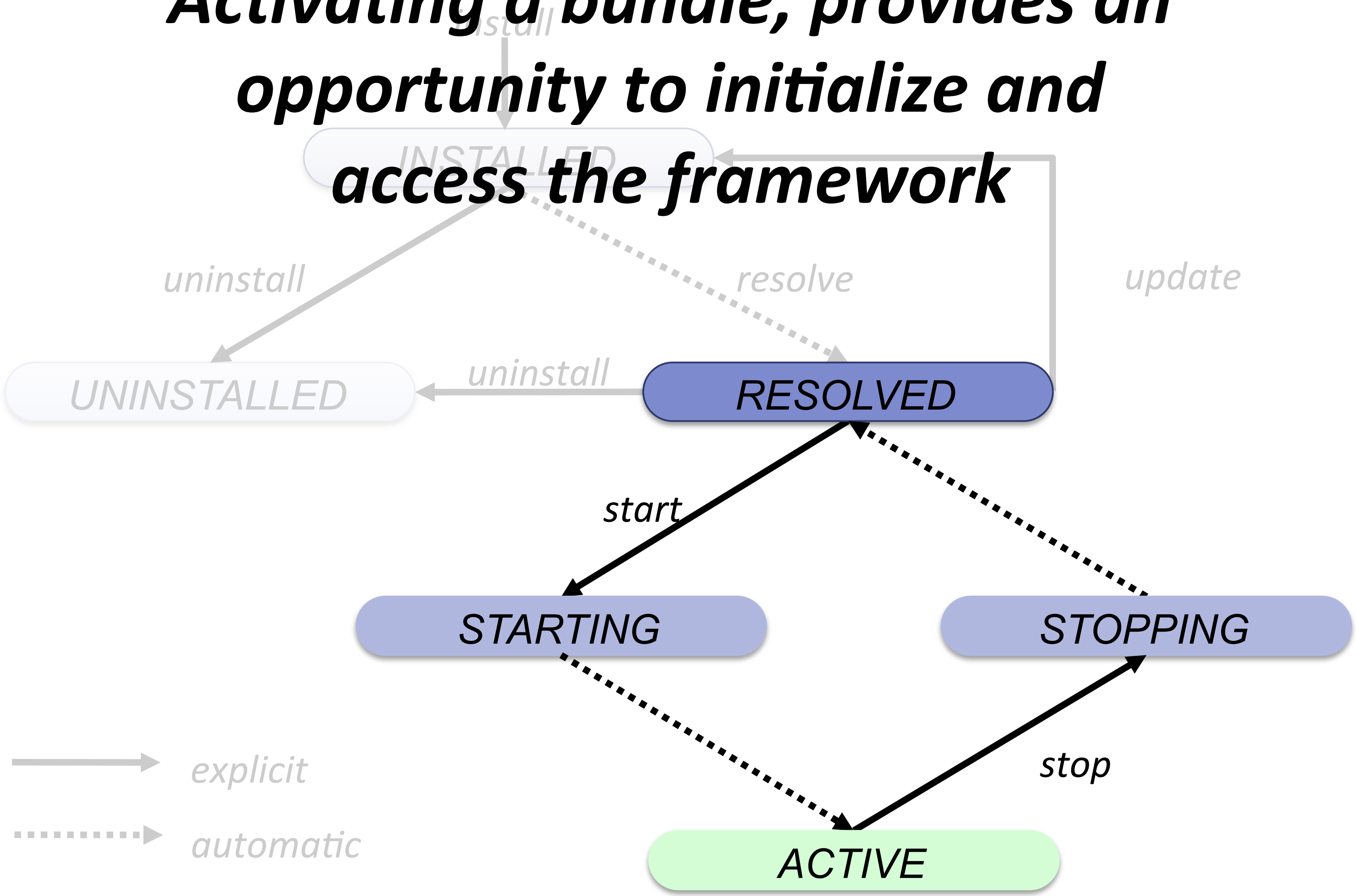
Bundle Life Cycle



Resolving a bundle, resolves its code dependencies

Bundle Life Cycle

Activating a bundle, provides an opportunity to initialize and access the framework



Bundle Activator

The bundle activator is a framework hook to allow bundles to startup and shutdown

Bundle Activator

The bundle activator is a framework hook to allow bundles to startup and shutdown

- The hook is invoked in the STARTING/STOPPING states

Bundle Activator

The bundle activator is a framework hook to allow bundles to startup and shutdown

- The hook is invoked in the STARTING/STOPPING states
- An activator implements a simple interface and is included in the bundle JAR file

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

Bundle Activator

The bundle activator is a framework hook to allow bundles to startup and shutdown

- The hook is invoked in the STARTING/STOPPING states
- An activator implements a simple interface and is included in the bundle JAR file

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

- Additional manifest metadata is needed to declare the activator

```
Bundle-Activator: <fully-qualified-class-name>
```

e.g.:

```
Bundle-Activator: org.foo.MyActivator
```

Bundle Activator

The bundle activator is a framework hook to allow bundles to startup and shutdown

- The hook is invoked in the STARTING/STOPPING states
- An activator implements a simple interface and is included in the bundle JAR file

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

- Additional manifest metadata is nee[illegible] the activator

```
Bundle-Activator: <[illegible]name>
e.g.:
Bundle-Activator: org[illegible]
```

What are these?

Bundle Context

Represents the bundle's execution context

```
public interface BundleContext {
  String getProperty(String key);
  Bundle getBundle();
  Bundle installBundle(String location) throws BundleException;
  Bundle installBundle(String location, InputStream input)
    throws BundleException;
  Bundle getBundle(long id);
  Bundle[] getBundles();
  ...
  void addBundleListener(BundleListener listener);
  void removeBundleListener(BundleListener listener);
  void addFrameworkListener(FrameworkListener listener);
  void removeFrameworkListener(FrameworkListener listener);
  ...
  File getDataFile(String filename);
  ...
}
```

Bundle Context

Represents the bundle's execution context

Lifecycle method to install other bundles

```
public interface BundleContext {
  String getProperty(String key);
  Bundle getBundle();
  Bundle installBundle(String location) throws BundleException;
  Bundle installBundle(String location, InputStream input)
    throws BundleException;
  Bundle getBundle(long id);
  Bundle[] getBundles();
  ...
  void addBundleListener(BundleListener listener);
  void removeBundleListener(BundleListener listener);
  void addFrameworkListener(FrameworkListener listener);
  void removeFrameworkListener(FrameworkListener listener);
  ...
  File getDataFile(String filename);
  ...
}
```

Bundle Context

Represents the bundle's execution context

```
public interface BundleContext {
  String getProperty(String key);
  Bundle getBundle();
  Bundle installBundle(String location) throws BundleException;
  Bundle installBundle(String location, InputStream input)
    throws BundleException;
  Bundle getBundle(long id);
  Bundle[] getBundles();
  ...
  void addBundleListener(BundleListener listener);
  void removeBundleListener(BundleListener listener);
  void addFrameworkListener(FrameworkListener listener);
  void removeFrameworkListener(FrameworkListener listener);
  ...
  File getDataFile(String filename);
  ...
}
```

Access to other installed bundles

Bundle Context

Represents the bundle's execution context

```
public interface BundleContex
  String getProperty(Strin
  Bundle getBundle();
  Bundle installBundle(Stri                              undleException;
  Bundle installBundle(String lo              utStream input)
    throws BundleException;
  Bundle getBundle(long id);
  Bundle[] getBundles();
  ...
  void addBundleListener(BundleListener listener);
  void removeBundleListener(BundleListener listener);
  void addFrameworkListener(FrameworkListener listener);
  void removeFrameworkListener(FrameworkListener listener);
  ...
  File getDataFile(String filename);
  ...
}
```

Access to our own bundle...
what's that?

Bundle

Run-time representation of a bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Bundle

Run-time representation of a bundle

Lifecycle method to start bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Bundle

Run-time representation of a bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Lifecycle method to stop bundle

Bundle

Run-time representation of a bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Lifecycle method to update bundle

Bundle

Run-time representation of a bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Lifecycle method to uninstall bundle

Bundle

Run-time representation of a bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Access to bundle resources

Bundle

Run-time representation of a bundle

```
public interface Bundle {
  ...
  int getState();
  void start(int options) throws BundleException;
  void start() throws BundleException;
  void stop(int options) throws BundleException;
  void stop() throws BundleException;
  void update() throws BundleException;
  void update(InputStream in) throws BundleException;
  void uninstall() throws BundleException;
  Dictionary getHeaders();
  String getSymbolicName();
  long getBundleId();
  String getLocation();
  ...
  URL getResource(String name);
  Enumeration getResources(String name) throws IOException;
  Class loadClass(String name) throws ClassNotFoundException;
  ...
  BundleContext getBundleContext();
}
```

Access to accessible classes

Bundle Dynamism

Bundles can be installed, started, stopped, updated, and uninstalled at run time

- Bundle events signal lifecycle changes

To listen for events

`BundleContext.addBundleListener()`

Bundle Dynamism

Bundles can be installed, started, stopped, updated, and uninstalled at run time

- Bundle events signal lifecycle changes

Implement listener interface

```
public interface BundleListener extends EventListener {
  public void bundleChanged(BundleEvent event);
}
```

Bundle Dynamism

Bundles can be installed, started, stopped, updated, and uninstalled at run time

- Bundle events signal lifecycle changes

Received event

```
public class BundleEvent extends EventObject {
  public final static int    INSTALLED   = 0x00000001;
  public final static int    STARTED     = 0x00000002;
  public final static int    STOPPED     = 0x00000004;
  public final static int    UPDATED     = 0x00000008;
  public final static int    UNINSTALLED = 0x00000010;
  public final static int    RESOLVED    = 0x00000020;
  public final static int    UNRESOLVED  = 0x00000040;
  public final static int    STARTING    = 0x00000080;
  public final static int    STOPPING    = 0x00000100;
  …
  public Bundle getBundle() { … }
  public int getType() { … }
}
```

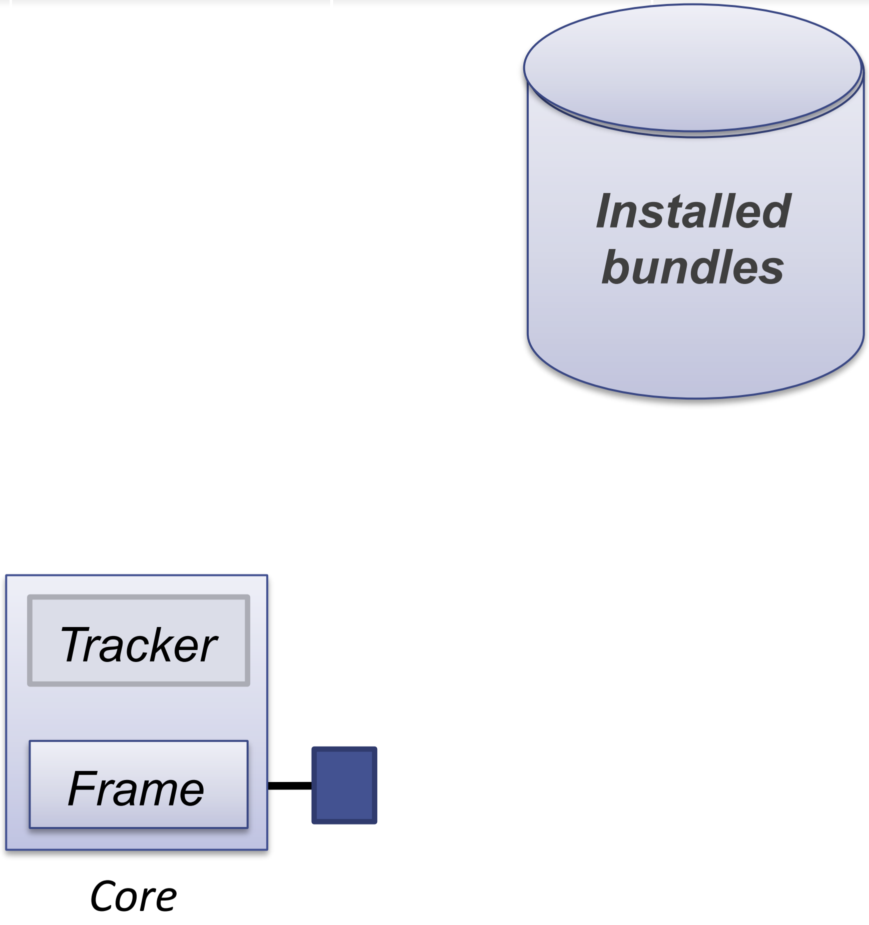
Bundle-Based Dynamic Extensibility

Bundle lifecycle events provide a mechanism for dynamic extensibility

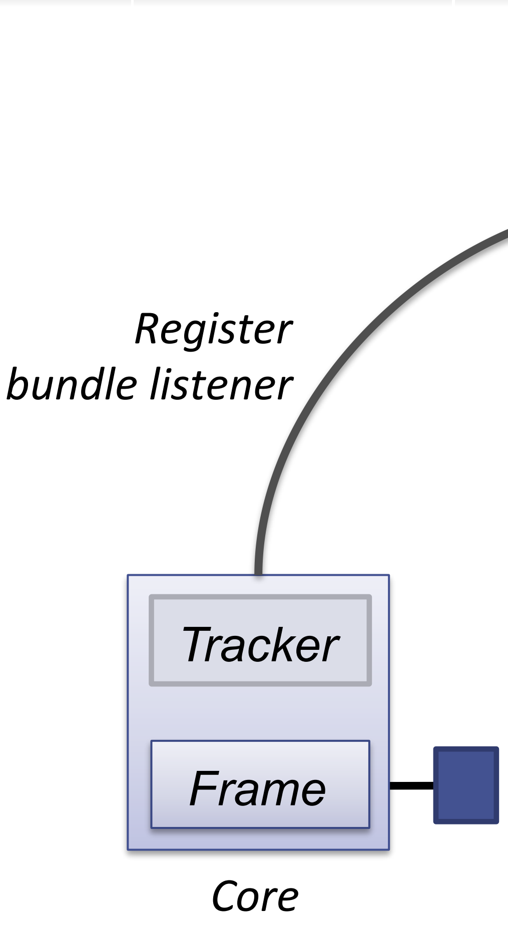
The extender pattern

- An application component, called the extender, listens for bundles to be installed, started, and stopped
- On install, the extender probes bundles to see if they are extensions
  - Typically, extension contain special metadata or resources to indicate they provide an extension
- When started, the extender performs some action to integrate the extension into the application
- When stopped, the extender performs some action to remove the extension from the application

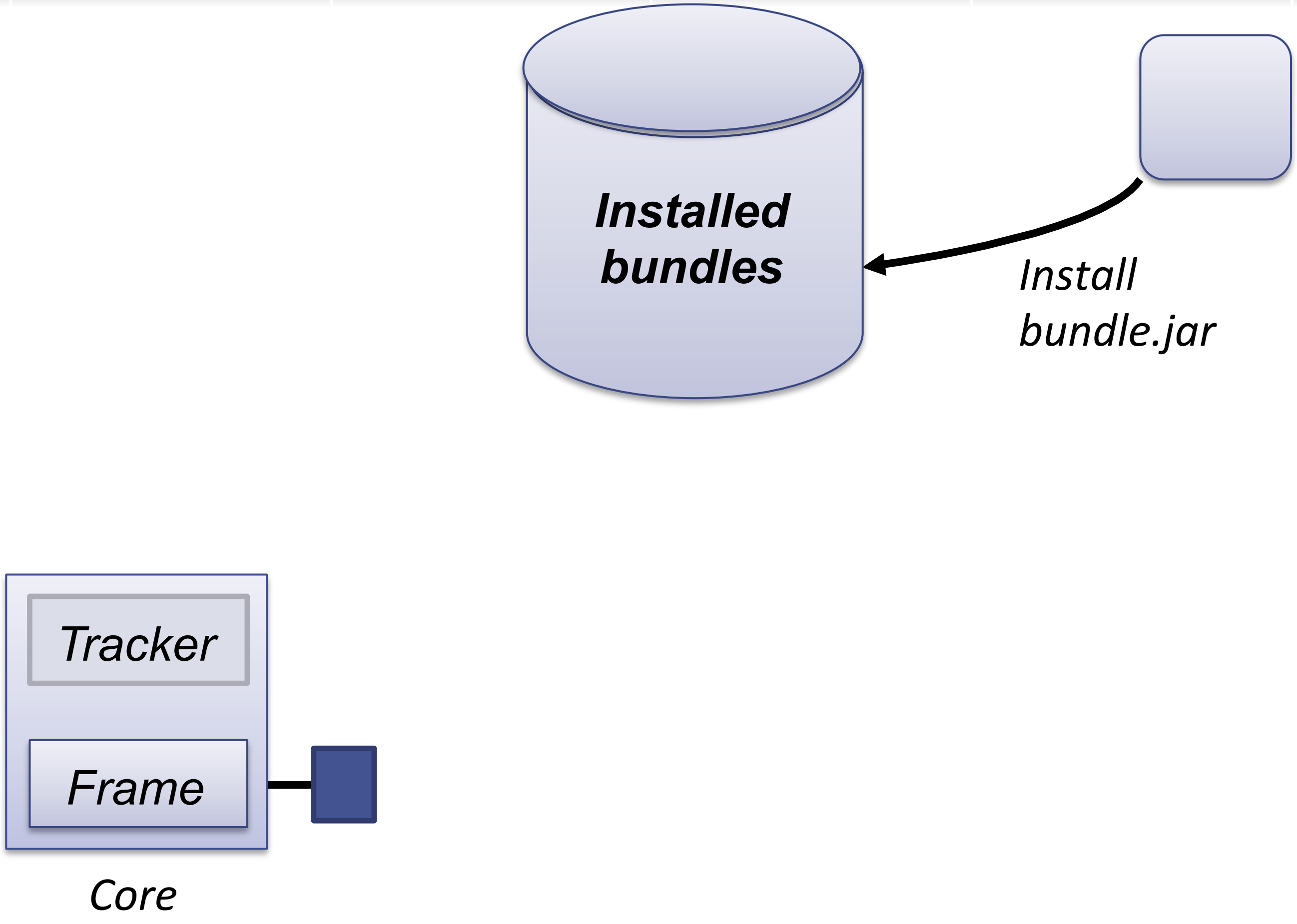
Extender Pattern



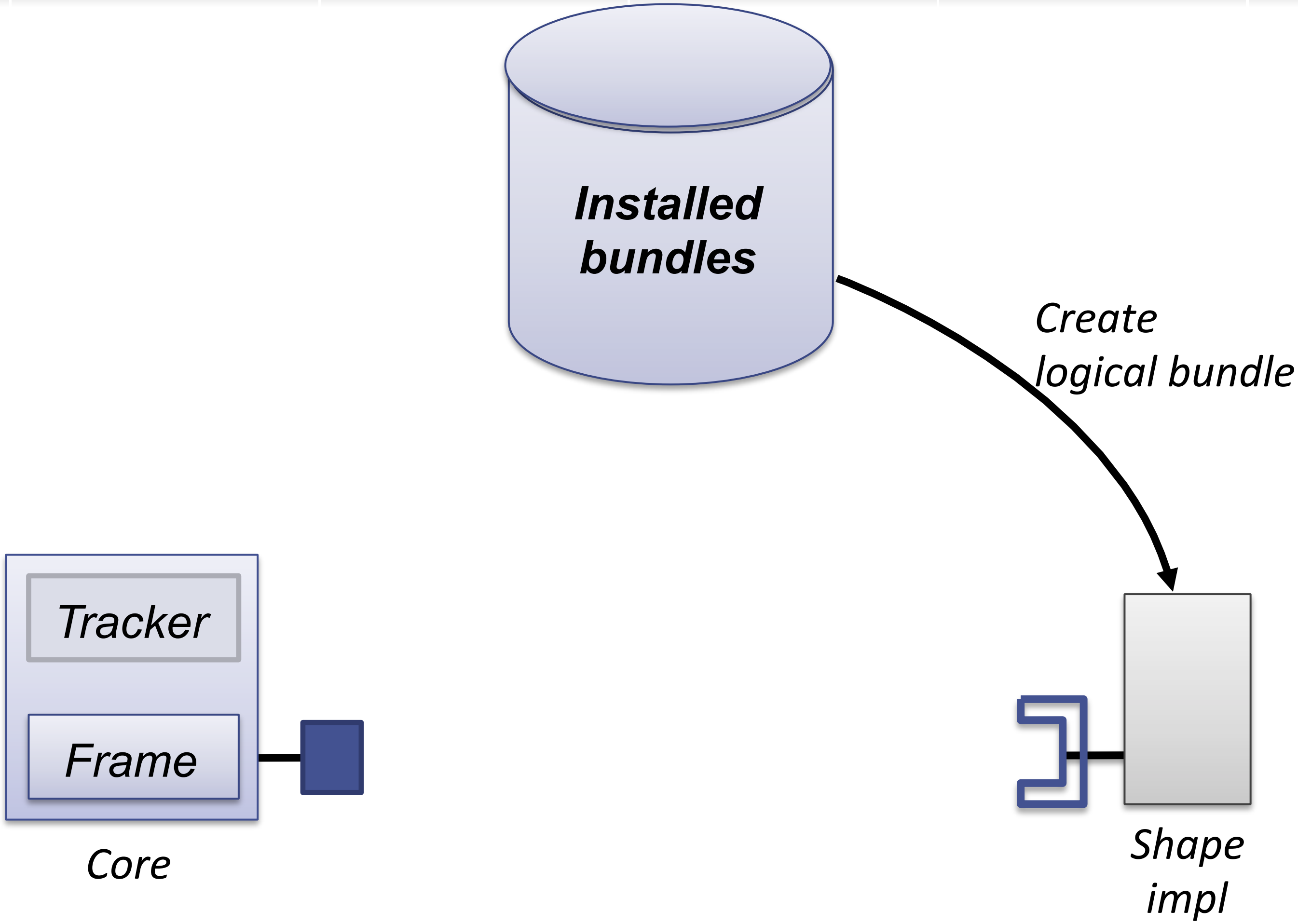
Extender Pattern



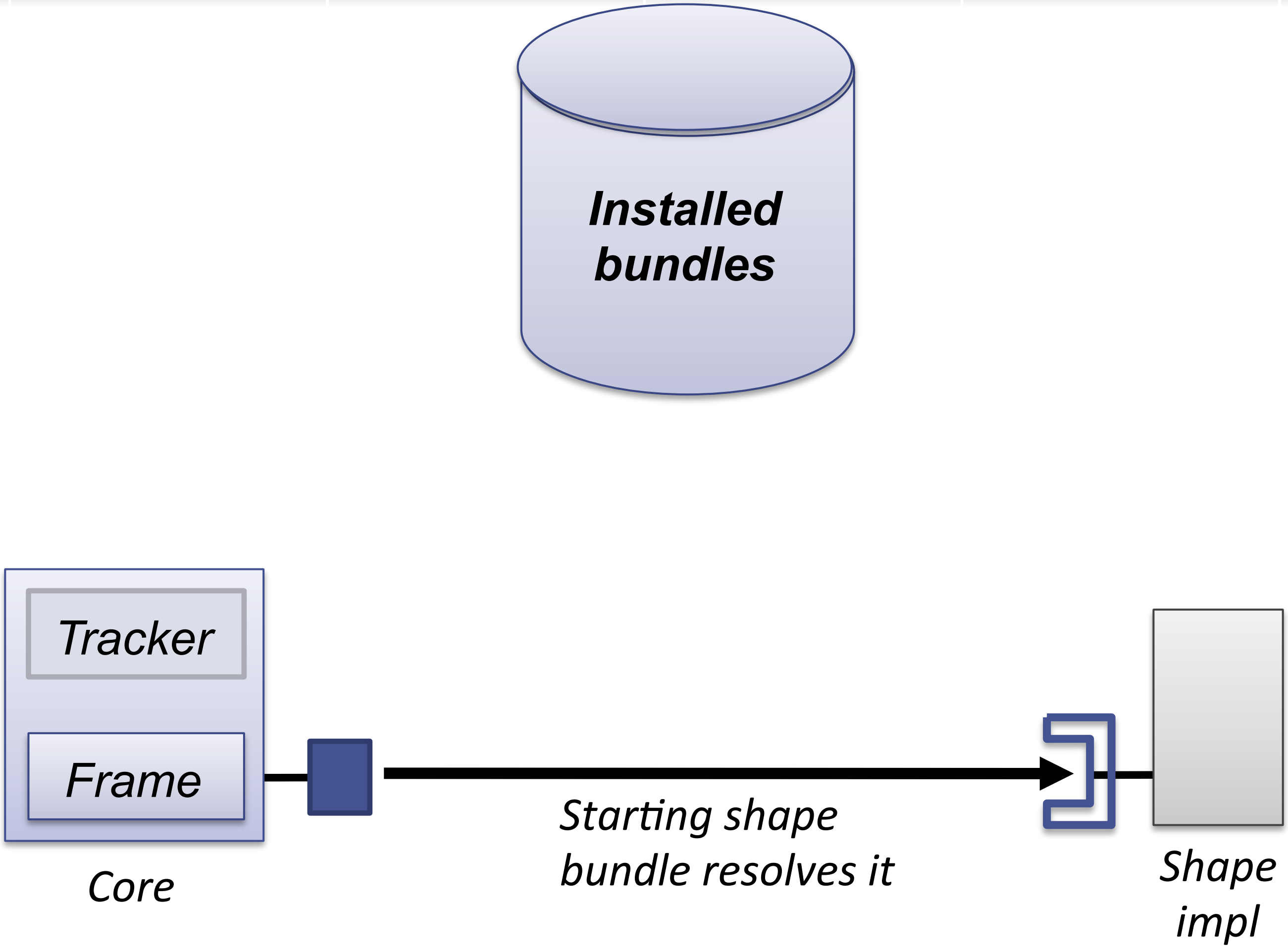
Extender Pattern



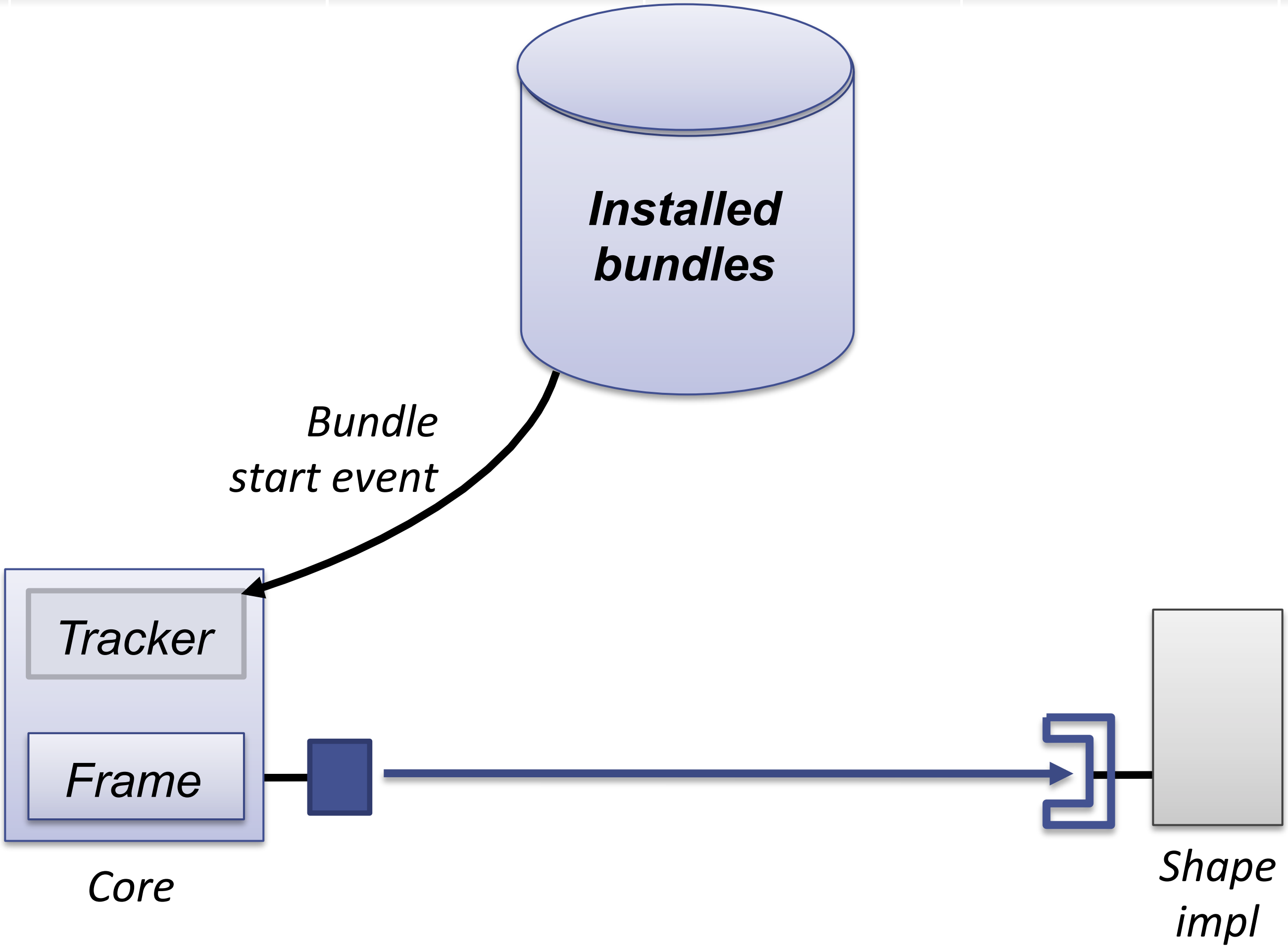
Extender Pattern



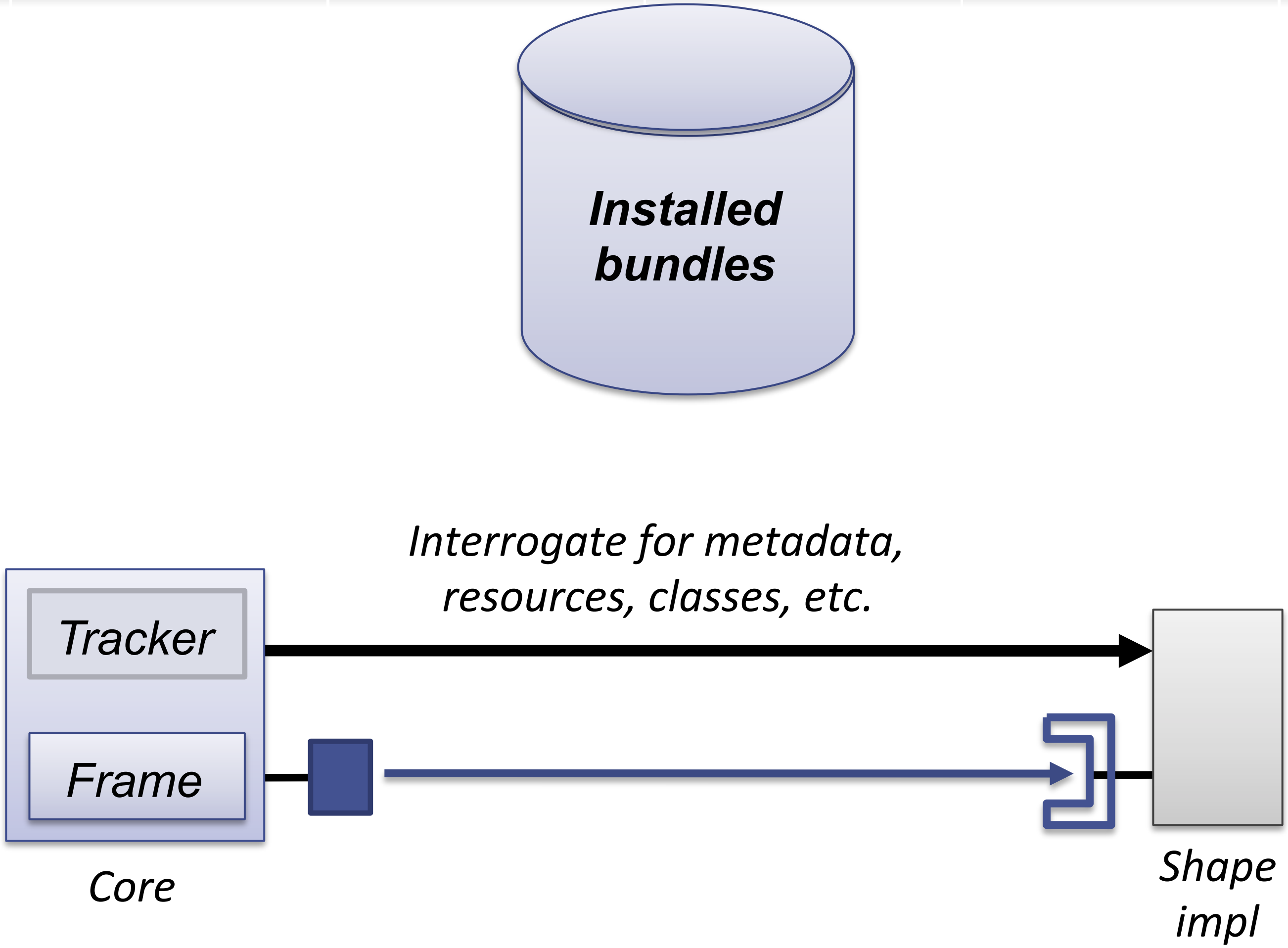
Extender Pattern



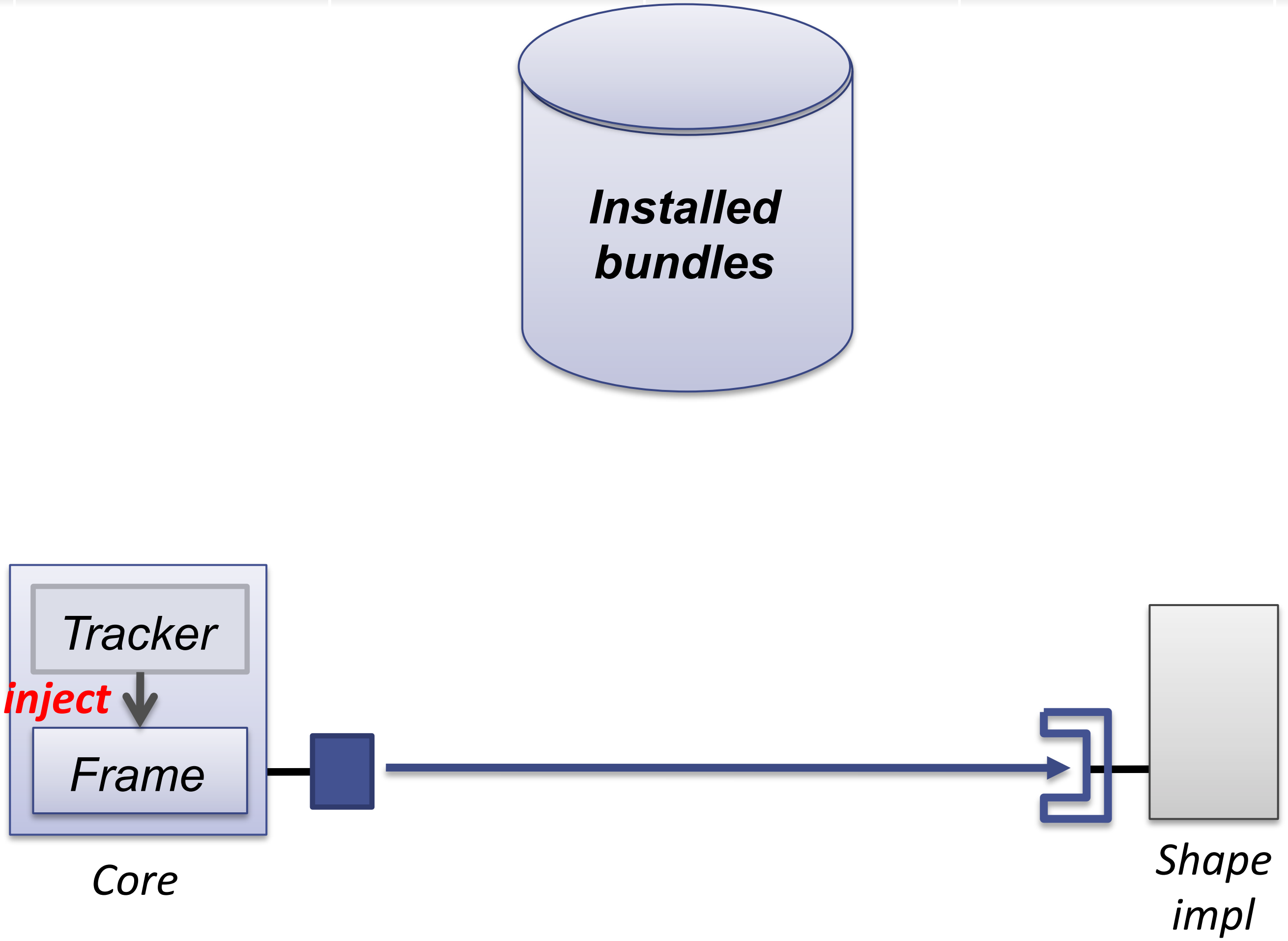
Extender Pattern



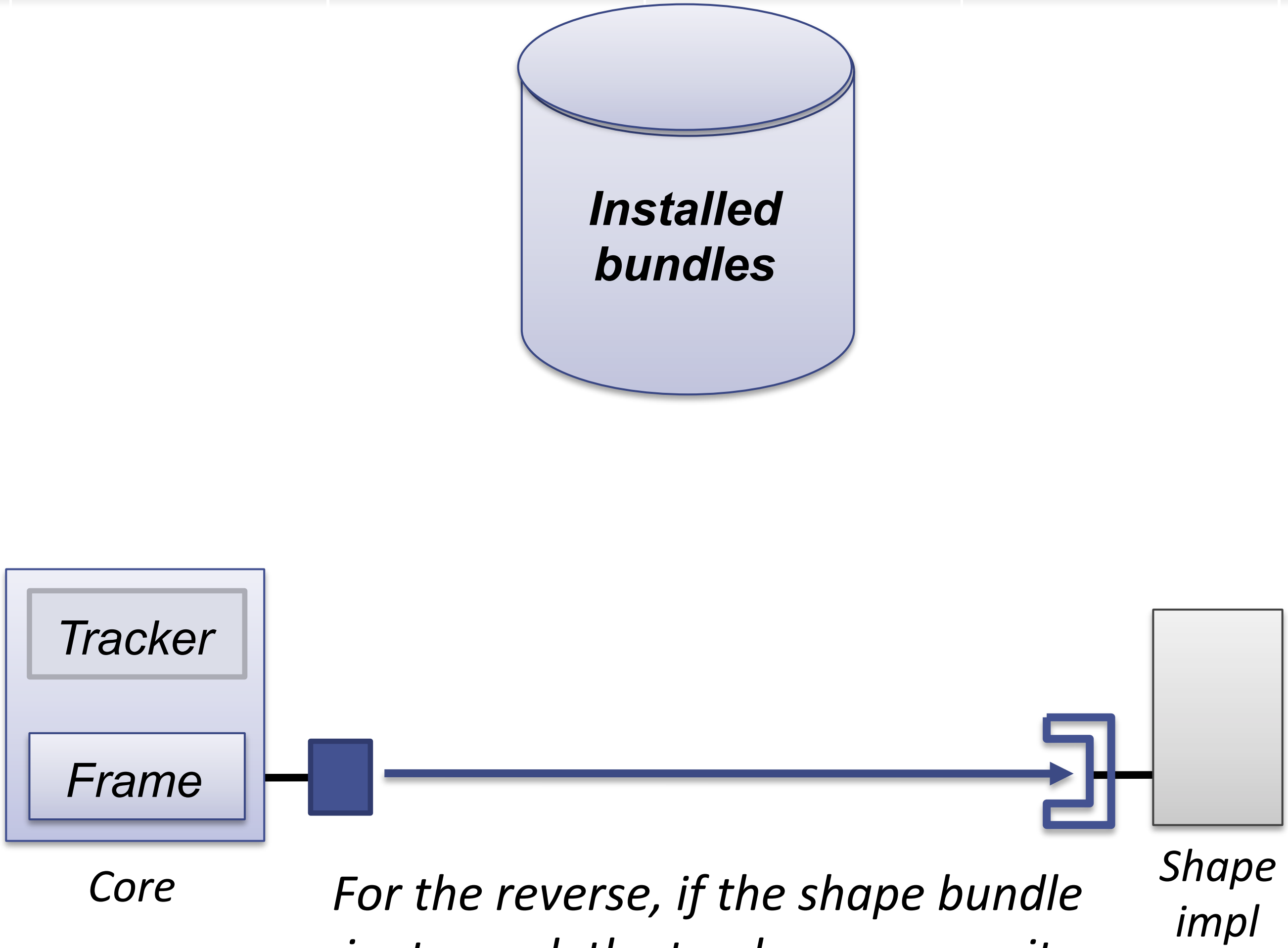
Extender Pattern



Extender Pattern



Extender Pattern



For the reverse, if the shape bundle is stopped, the tracker removes its associated shape.

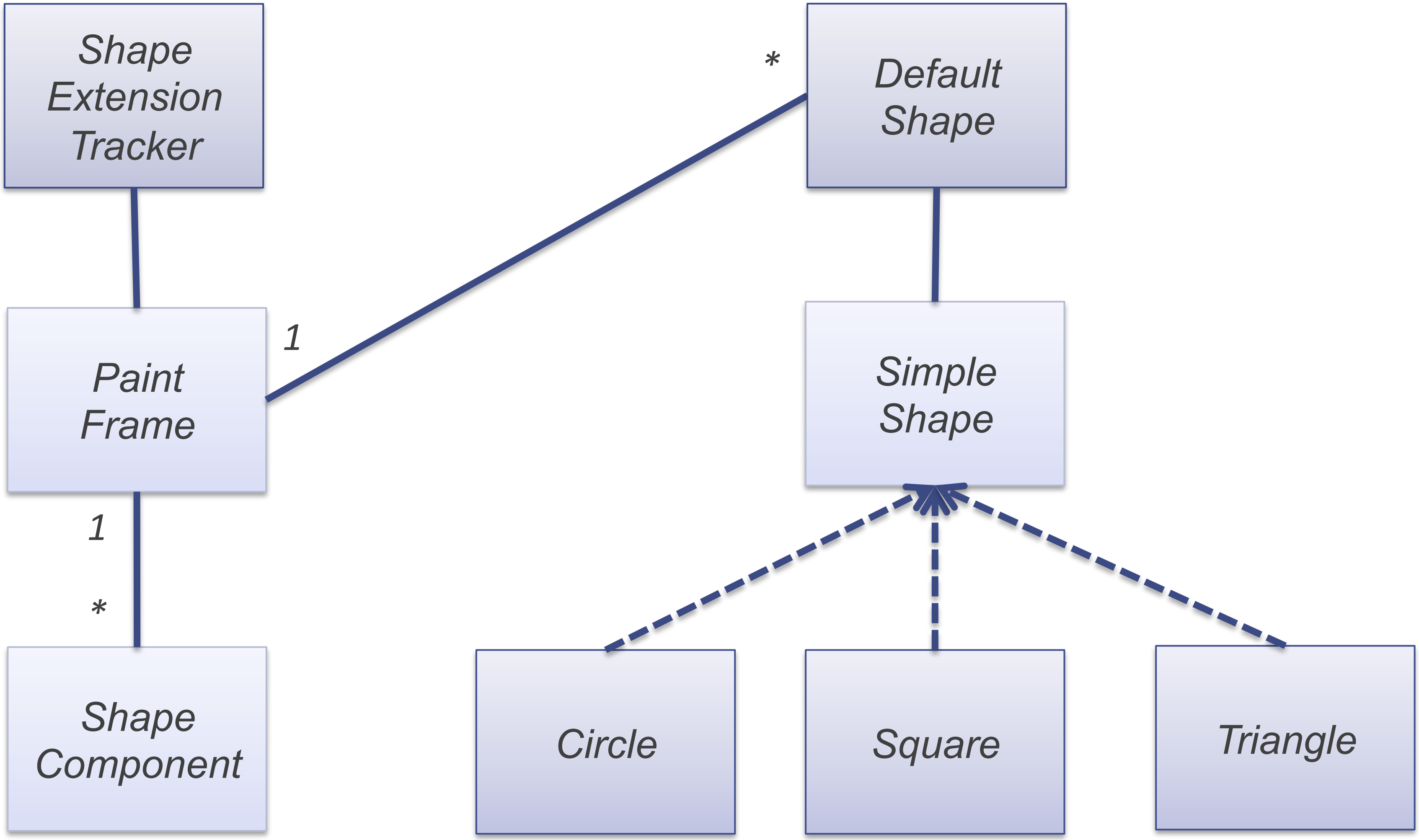
Extender Paint Program Overview (1/2)

Dynamically extensible paint program

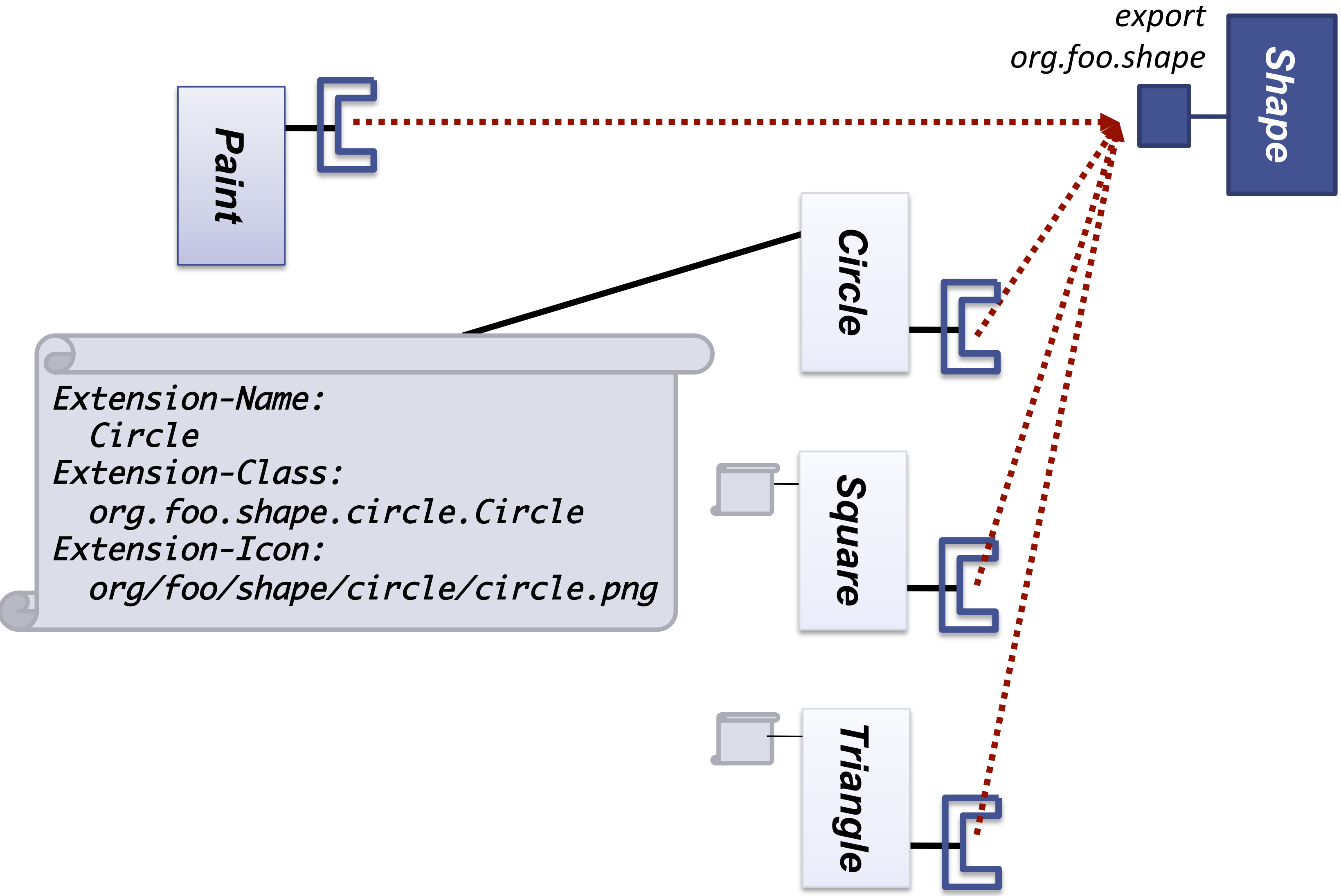
- Uses the extender pattern to deliver shapes
- The paint bundle is the extender, i.e., it listens for bundles containing shapes
- On install, the extender probes bundles to see if they are extensions
  - Special metadata in the manifest denotes the name, class, and icon of the shape

Uses placeholder when shape has been used, but currently unavailable because the bundle is not active

Extender Paint Program Design (1/2)



Extender Paint Program Design (2/2)



```
Extension-Name:
  Circle
Extension-Class:
  org.foo.shape.circle.Circle
Extension-Icon:
  org/foo/shape/circle/circle.png
```

Refreshing the Framework

Update and uninstall lifecycle operations are a little complicated

- Why?

Refreshing the Framework

Update and uninstall lifecycle operations are a little complicated

- Why?
  - Existing bundle maybe be using classes from the bundle being updated or uninstalled
  - Cannot pull the rug out from under dependent bundles

Refreshing the Framework

Update and uninstall lifecycle operations are a little complicated

- Why?
  - Existing bundle maybe be using classes from the bundle being updated or uninstalled
  - Cannot pull the rug out from under dependent bundles

To deal with this, the framework treats update and uninstall as a two-step process

- Updates and uninstalls do not happen immediately
- Framework must be “refreshed” to put them into effect
  - Actually, for updates it is a little more complicated than this, but we can accept this view for now...

Refreshing the Framework

Update and uninstall lifecycle operations are a little complicated

- Why?
  - Existing bundle maybe be using classes from the bundle being updated or uninstalled
  - Cannot pull the rug out from under dependent bundles

To deal with this, the framework treats update and uninstall as a two-step process

- Updates and uninstalls do not happen immediately
- Framework must be “refreshed” to put them into effect
  - Actually, for updates it is a little more complicated than this, but we can accept this view for now...

How do we refresh the framework?

Package Admin

Framework provides special API to deal with bundles interactions

```
public interface PackageAdmin {
  static final int BUNDLE_TYPE_FRAGMENT = 0x00000001;
  Bundle getBundle(Class clazz);
  Bundle[] getBundles(String symbolicName, String
versionRange);
  int getBundleType(Bundle bundle);
  ExportedPackage getExportedPackage(String name);
  ExportedPackage[] getExportedPackages(Bundle bundle);
  ExportedPackage[] getExportedPackages(String name);
  Bundle[] getFragments(Bundle bundle);
  RequiredBundle[] getRequiredBundles(String symbolicName);
  Bundle[] getHosts(Bundle bundle);
  void refreshPackages(Bundle[] bundles);
  boolean resolveBundles(Bundle[] bundles);
}
```

Package Admin

Framework provides special API to deal with bundles interactions

```
public interface PackageAdmin {
  static final int BUNDLE_TYPE_FRAGMENT = 0x00000001;
  Bundle getBundle(Class clazz);
  Bundle[] getBundles(String symbolicName, String
versionRange);
  int getBundleType(Bundle bundle);
  ExportedPackage getExportedPackage(String name);
  ExportedPackage[] getExportedPackages(Bundle bundle);
  ExportedPackage[] getExportedPackages(String name);
  Bundle[] getFragments(Bundle bundle);
  RequiredBundle[] getRequiredBundles(String symbolicName);
  Bundle[] getHosts(Bundle bundle);
  void refreshPackages(Bundle[] bundles
  boolean resolveBundles(Bundle[] bu
}
```

Provides various methods to introspect bundle dependencies

Package Admin

Framework provides special API to deal with bundles interactions

```
public interface PackageAdmin {
  static final int BUNDLE_TYPE_FRAGMENT = 0x00000001;
  Bundle getBundle(Class clazz);
  Bundle[] getBundles(String symbolicName, String 
versionRange);
  int getBundleType(Bundle bundle);
  ExportedPackage getExportedPackage(String name);
  ExportedPackage[] getExportedPackages(
  ExportedPackage[] getExportedPacka
  Bundle[] getFragments(Bundle bund
  RequiredBundle[] getRequiredBundle
  Bundle[] getHosts(Bundle bundle);
  void refreshPackages(Bundle[] bundles);
  boolean resolveBundles(Bundle[] bundles);
}
```

So, how do we gain access to this API?

The Service Layer

Service Orientation

The OSGi framework promotes a service-oriented interaction pattern among bundles

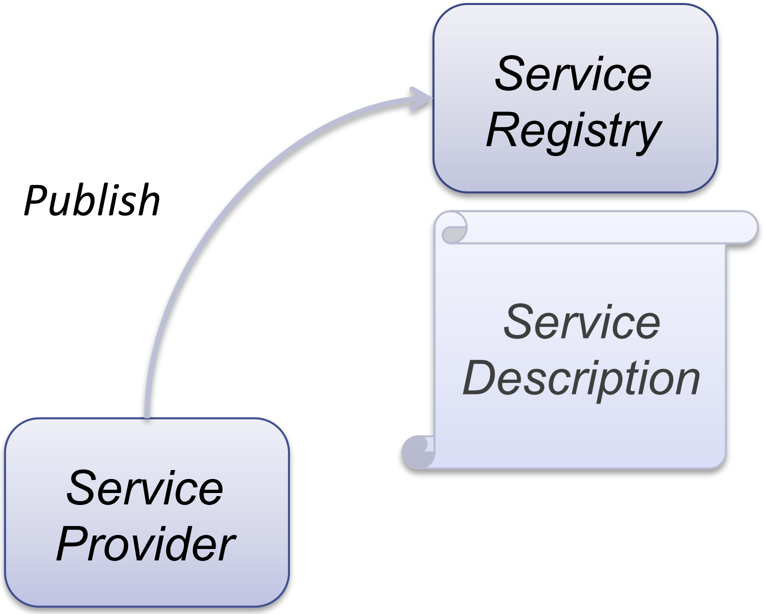
Service Orientation

The OSGi framework promotes a service-oriented interaction pattern among bundles

Service Registry

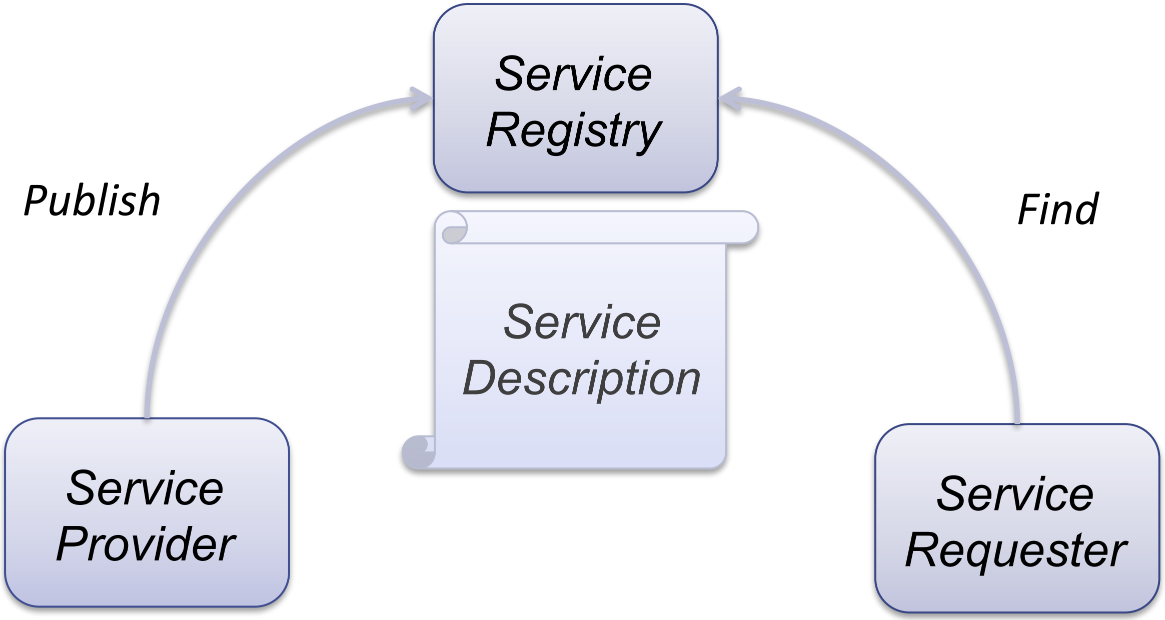
Service Orientation

The OSGi framework promotes a service-oriented interaction pattern among bundles



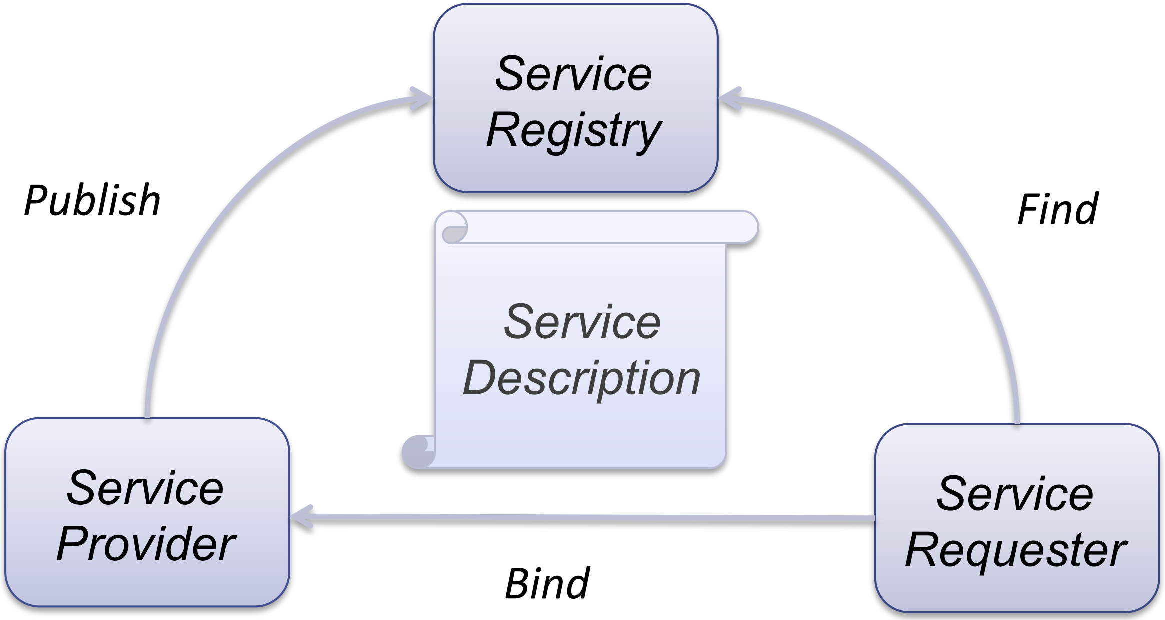
Service Orientation

The OSGi framework promotes a service-oriented interaction pattern among bundles



Service Orientation

The OSGi framework promotes a service-oriented interaction pattern among bundles



OSGi Service Approach Advantages

Lightweight services

- Direct method invocation

Structured code

- Promotes separation of interface from implementation
- Enables reuse, substitutability, loose coupling, and late binding

Dynamics

- Loose coupling and late binding make it possible to support run-time management of module

OSGi Applications

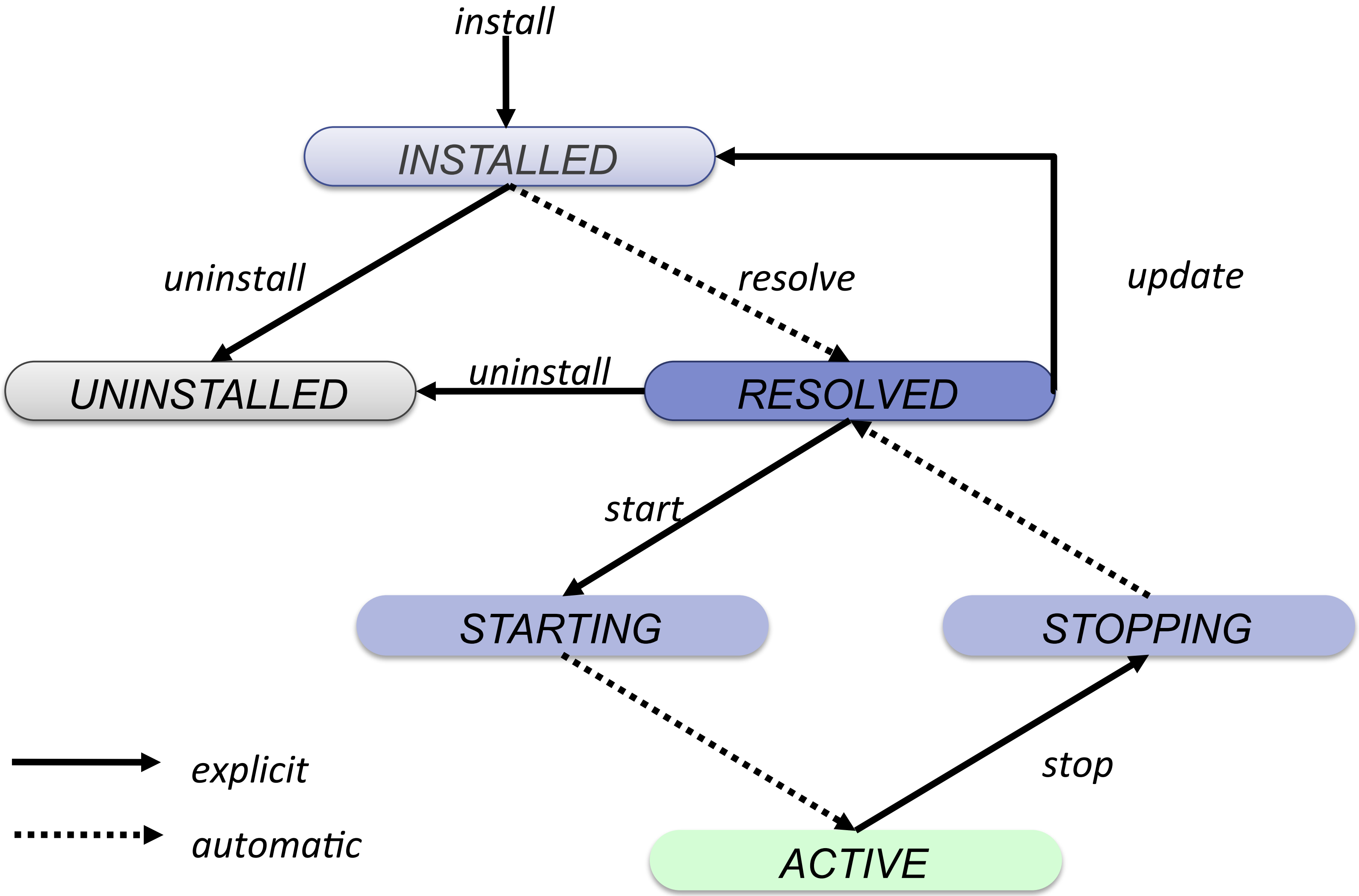
A collection of bundles that interact via service interfaces

- Bundles may be independently developed and deployed
- Bundles and their associated services may appear or disappear at any time

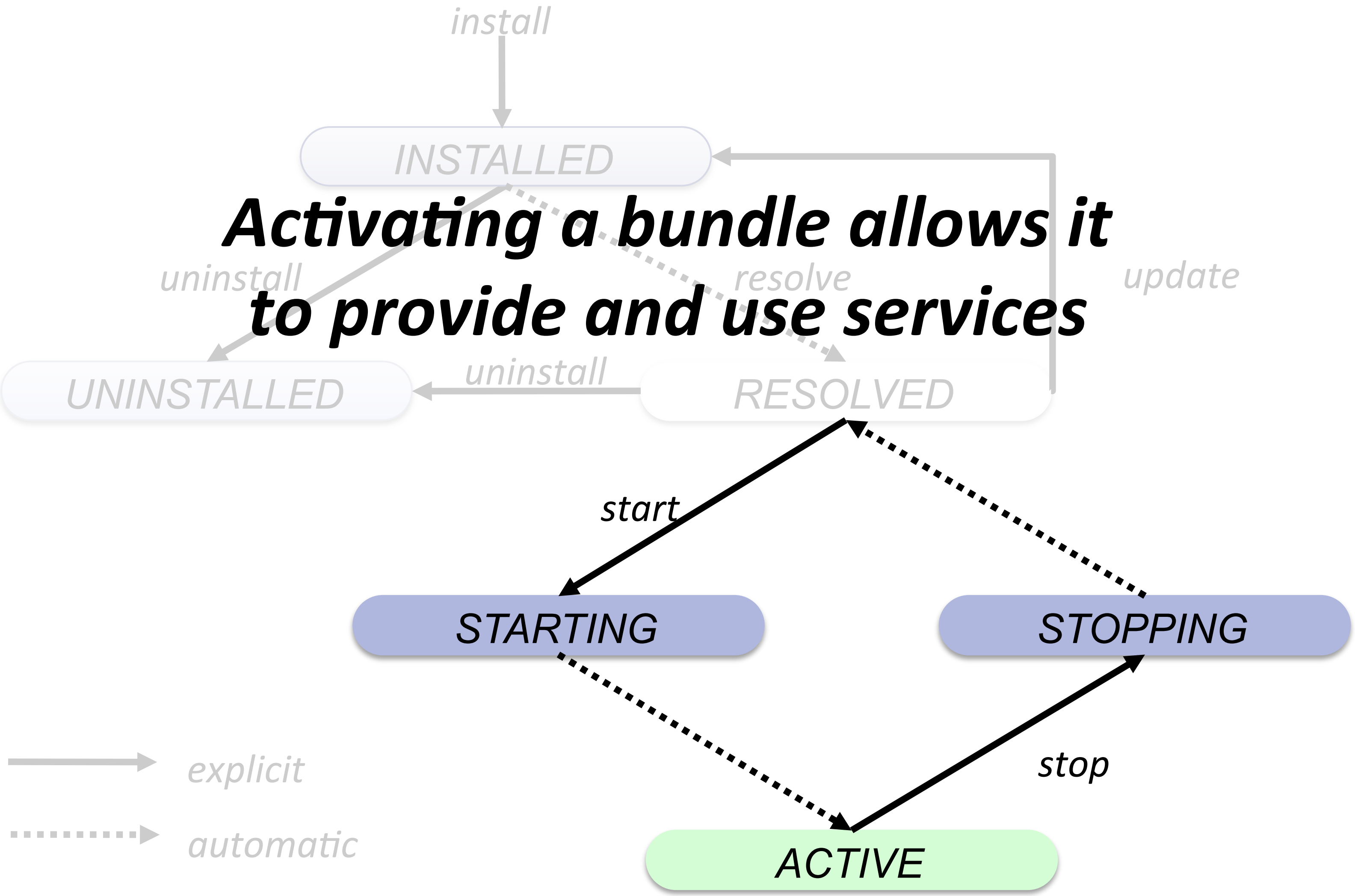
Resulting application follows a Service-Oriented Component Model approach

- Combines ideas from both component and service orientation

Bundle Life Cycle (Revisited)



Bundle Life Cycle (Revisited)



Activating a bundle allows it to provide and use services

What's a Service?

Just a simple Java object

Typically described by a Java interface

- Allows for multiple providers

Using a service is just like using any object

Hello World Service Example

Let's assume we have this service interface

```
package com.foo.hello;
public interface Hello {
  void sayHello(String name);
}
```

Hello World Service Example

Let's assume we have this service interface

```
package com.foo.hello;
public interface Hello {
  void sayHello(String name);
}
```

And this implementation

```
package com.foo.hello.impl;
import com.foo.hello;
public class HelloImpl implements Hello {
  public void sayHello(String name) {
    System.out.println("Hello " + name + "!");
  }
}
```

Publishing a Service (1/2)

BundleContext allows bundles to publish services

```
public interface BundleContext {
  …
  void addServiceListener(ServiceListener listener, String
filter)
  throws InvalidSyntaxException;
  void addServiceListener(ServiceListener listener);
  void removeServiceListener(ServiceListener listener);
  ServiceRegistration registerService(
    String[] clazzes, Object service, Dictionary props);
  ServiceRegistration registerService(
    String clazz, Object service, Dictionary props);
  ServiceReference[] getServiceReferences(String clazz, String
filter)
    throws InvalidSyntaxException;
  ServiceReference getServiceReference(String clazz);
  Object getService(ServiceReference reference);
  boolean ungetService(ServiceReference reference);
}
```

Publishing a Service (1/2)

BundleContext allows bundles to publish services

```
public interface BundleContext {
  …
  void addServiceListener(ServiceListener listener, String 
filter)
  throws InvalidSyntaxException;
  void addServiceListener(ServiceListener listener);
  void removeServiceListener(ServiceListener listener);
  ServiceRegistration registerService(
    String[] clazzes, Object service, Dictionary props);
  ServiceRegistration registerService(
    String clazz, Object service, Dictionary props);
  ServiceReference[] getServiceReferences(String clazz, String 
filter)
    throws InvalidSyntaxException;
  ServiceReference getServiceReference
  Object getService(ServiceReference re
  boolean ungetService(ServiceReference re
}
```

We have two methods for publishing services

Publishing a Service (2/2)

Bundles often publish services in their activator

```
package com.foo.hello.impl;
import org.osgi.framework.*;
public class Activator implements BundleActivator {
  private ServiceRegistration m_reg = null;
  public void start(BundleContext context) {
    m_reg = context.registerService(
      com.foo.hello.Hello.class.getName(), new HelloImpl(), 
null);
  }

  public void stop(BundleContext context) {
    m_reg.unregister();
  }
}
```

Publishing a Service (2/2)

Bundles often publish services in their activator

```
package com.foo.hello.impl;
import org.osgi.framework.*;
public class Activator implements BundleActivator {
  private ServiceRegistration m_reg = null;
  public void start(BundleContext context) {
    m_reg = context.registerService(
      com.foo.hello.Hello.class.getName(), new HelloImpl(),
null);
  }

  public void stop(BundleContext context)
    m_reg.unregister();
  }
}
```

We register the service when starting, which makes it available to other bundles

Publishing a Service (2/2)

Bundles often publish services in their activator

```
package com.foo.hello.impl;
import org.osgi.framework.*;
public class Activator implements BundleActivator {
  private ServiceRegistration m_reg = null;
  public void start(BundleContext context) {
    m_reg = context.registerService(
      com.foo.hello.Hello.class.getName(), new HelloImpl(),
null);
  }

  public void stop(BundleContext context) {
    m_reg.unregister();
  }
}
```

We unregister it when stopping

Packaging the Hello World Service

Our service implementation bundle contains these packages

- com.foo.hello
- com.foo.hello.impl

Packaging the Hello World Service

Our service implementation bundle contains these packages

- com.foo.hello
- com.foo.hello.impl

And the following manifest metadata

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: com.foo.hello.impl
Export-Package: com.foo.hello
Import-Package: org.osgi.framework,
com.foo.hello
Bundle-Activator: com.foo.hello.impl.Activator
```

Using a Service (1/2)

BundleContext allows bundles to find services

```
public interface BundleContext {
  …
  void addServiceListener(ServiceListener listener, String
filter)
  throws InvalidSyntaxException;
  void addServiceListener(ServiceListener listener);
  void removeServiceListener(ServiceListener listener);
  ServiceRegistration registerService(
    String[] clazzes, Object service, Dictionary props);
  ServiceRegistration registerService(
    String clazz, Object service, Dictionary props);
  ServiceReference[] getServiceReferences(String clazz, String
filter)
    throws InvalidSyntaxException;
  ServiceReference getServiceReference(String clazz);
  Object getService(ServiceReference reference);
  boolean ungetService(ServiceReference reference);
}
```

Using a Service (1/2)

BundleContext allows bundles to find services

```
public interface BundleContext {
  …
  void addServiceListener(ServiceListener listener, String
filter)
  throws InvalidSyntaxException;
  void addServiceListener(ServiceListener li
  void removeServiceListener(ServiceListen
  ServiceRegistration registerService(
    String[] clazzes, Object service, Diction
  ServiceRegistration registerService(
    String clazz, Object service, Dictionary props);
  ServiceReference[] getServiceReferences(String clazz, String
filter)
    throws InvalidSyntaxException;
  ServiceReference getServiceReference(String clazz);
  Object getService(ServiceReference reference);
  boolean ungetService(ServiceReference reference);
}
```

We have methods to find service references and get service objects

Using a Service (2/2)

Bundles retrieve service references

- Indirect references to service object

```
package com.foo.hello.client;
import org.osgi.framework.*;
import com.foo.hello.Hello;
public class HelloClient implements BundleActivator {
  public void start(BundleContext context) {
    ServiceReference ref = context.getServiceReference(
      com.foo.hello.Hello.class.getName());
    if (ref != null) {
      Hello h = (Hello) context.getService(ref);
      if (h != null) {
        h.sayHello("World");
        context.ungetService(h);
      }
    }
  }
…
}
```

Using a Service (2/2)

Bundles retrieve service references

- Indirect references to service object

```
package com.foo.hello.client;
import org.osgi.framework.*;
import com.foo.hello.Hello;
public class HelloClient implements BundleActivator {
  public void start(BundleContext context) {
    ServiceReference ref = context.getServiceReference(
      com.foo.hello.Hello.class.getName());
    if (ref != null) {
      Hello h = (Hello) context.getService(ref);
      if (h != null) {
        h.sayHello("World");
        context.ungetService(h);
      }
    }
  }
…
}
```

We first get the service reference

Using a Service (2/2)

Bundles retrieve service references

- Indirect references to service object

```
package com.foo.hello.client;
import org.osgi.framework.*;
import com.foo.hello.Hello;
public class HelloClient implements BundleActivator {
  public void start(BundleContext context) {
    ServiceReference ref = context.getServiceReference(
      com.foo.hello.Hello.class.getName());
    if (ref != null) {
      Hello h = (Hello) context.getService(ref);
      if (h != null) {
        h.sayHello("World");
        context.ungetService(h);
      }
    }
  }
…
}
```

From the service reference we get the service object

Using a Service (2/2)

Bundles retrieve service references

- Indirect references to service object

```
package com.foo.hello.client;
import org.osgi.framework.*;
import com.foo.hello.Hello;
public class HelloClient implements BundleActivator {
  public void start(BundleContext context) {
    ServiceReference ref = context.getServiceReference(
      com.foo.hello.Hello.class.getName());
    if (ref != null) {
      Hello h = (Hello) context.getService(ref);
      if (h != null) {
        h.sayHello("World");
        context.ungetService(ref);
      }
    }
  }
…
}
```

We use the service like a normal object

Using a Service (2/2)

Bundles retrieve service references

- Indirect references to service object

```
package com.foo.hello.client;
import org.osgi.framework.*;
import com.foo.hello.Hello;
public class HelloClient implements BundleActivator {
  public void start(BundleContext context) {
    ServiceReference ref = context.getServiceReference(
      com.foo.hello.Hello.class.getName());
    if (ref != null) {
      Hello h = (Hello) context.getService(ref);
      if (h != null) {
        h.sayHello("World");
        context.ungetService(h);
      }
    }
  }
…
}
```

And release the service object when we are done with it

Packaging the Hello World Service

Our client implementation bundle contains this package

- com.foo.hello.client

And the following manifest metadata

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: com.foo.hello.client
Import-Package: com.foo.hello,
org.osgi.framework
Bundle-Activator: com.foo.hello.client.Activator
```

Service Dynamism

Services can be published and revoked at run time

- Service events signal service changes
  - Must track events for any services being used

To listen for events

```
BundleContext.addServiceListener()
```

Services Dynamism

Services can be published and revoked at run time

- Service events signal service changes
  - Must track events for any services being used

<u>Implement listener interface</u>

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

Services Dynamism

Services can be published and revoked at run time

- Service events signal service changes
  - Must track events for any services being used

Received event

```
public class ServiceEvent extends EventObject {
  public final static int REGISTERED    = 0x00000001;
  public final static int MODIFIED      = 0x00000002;
  public final static int UNREGISTERING = 0x00000004;
  …
  public ServiceReference getServiceReference() { … }
  public int getType() { … }
}
```

Services Dynamism

Services can be published and revoked at run time

- Service events signal service changes
  - Must track events for any services being used

Received event

```
public class ServiceEvent extends EventObject {
  public final static int REGISTERED    = 0x00000001;
  public final static int MODIFIED      = 0x00000002;
  public final static int UNREGISTERING = 0x00000004;
  …
  public ServiceReference getServiceReference() { … }
  public int getType() { … }
}
```

Even though service are just normal objects, they are potentially much more volatile, so service events are very important

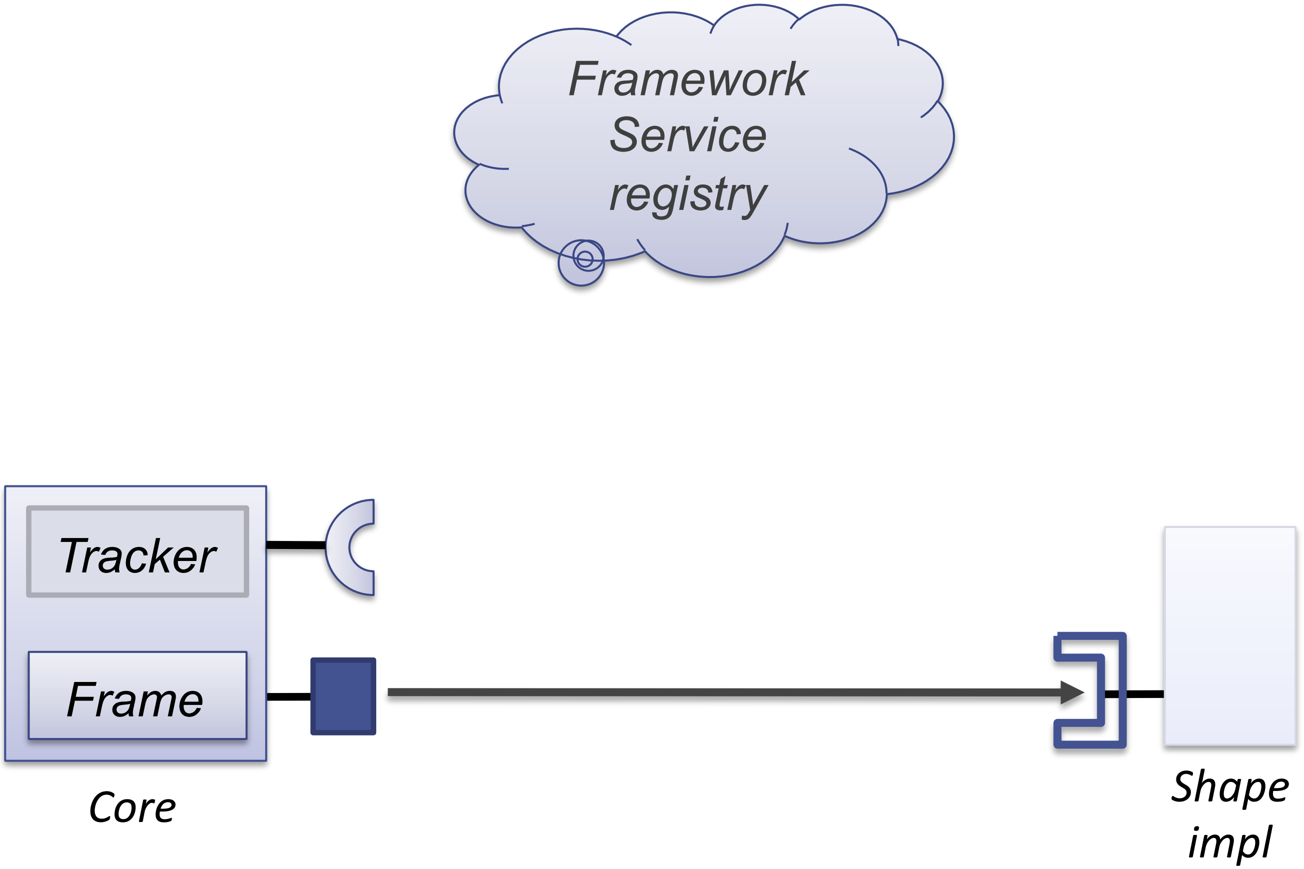
Service-Based Dynamic Extensibility

Service events provide a mechanism for dynamic extensibility

The whiteboard pattern

- Treats the service registry as a whiteboard
  - A reverse way to create a service
- An application component listens for services of a particular type to be added and removed
- On addition, the service is integrated into the application
- On removal, the service is removed from the application

Whiteboard Pattern



Framework Service registry

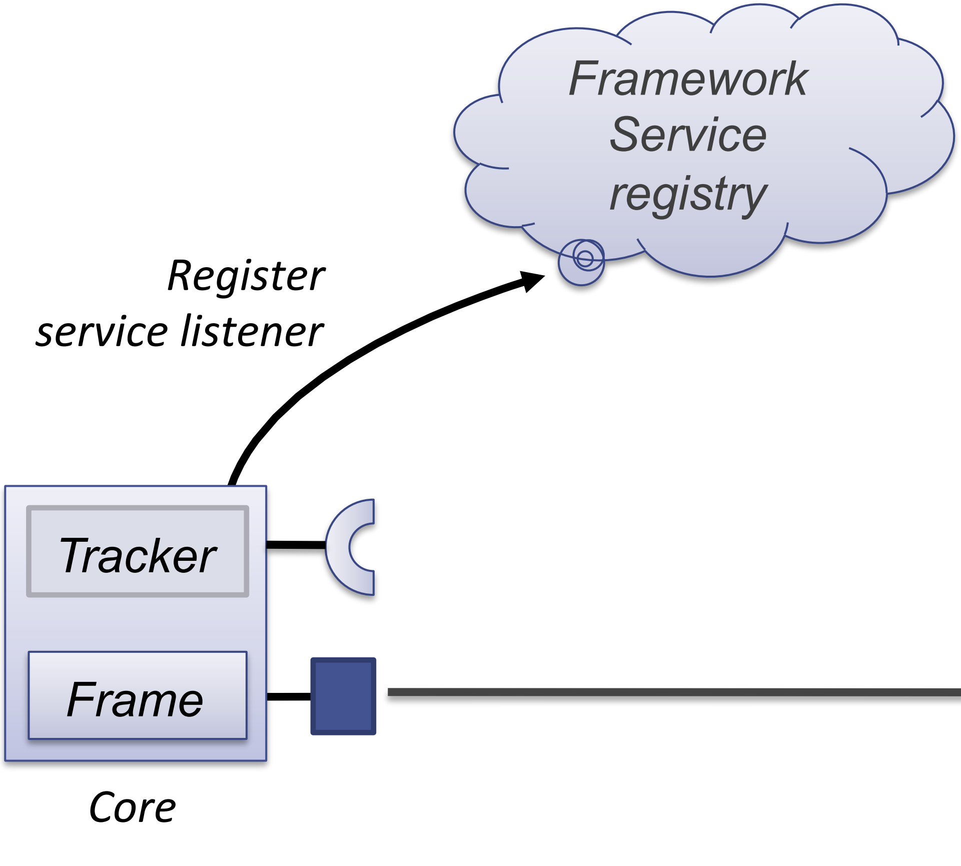
Tracker

Frame

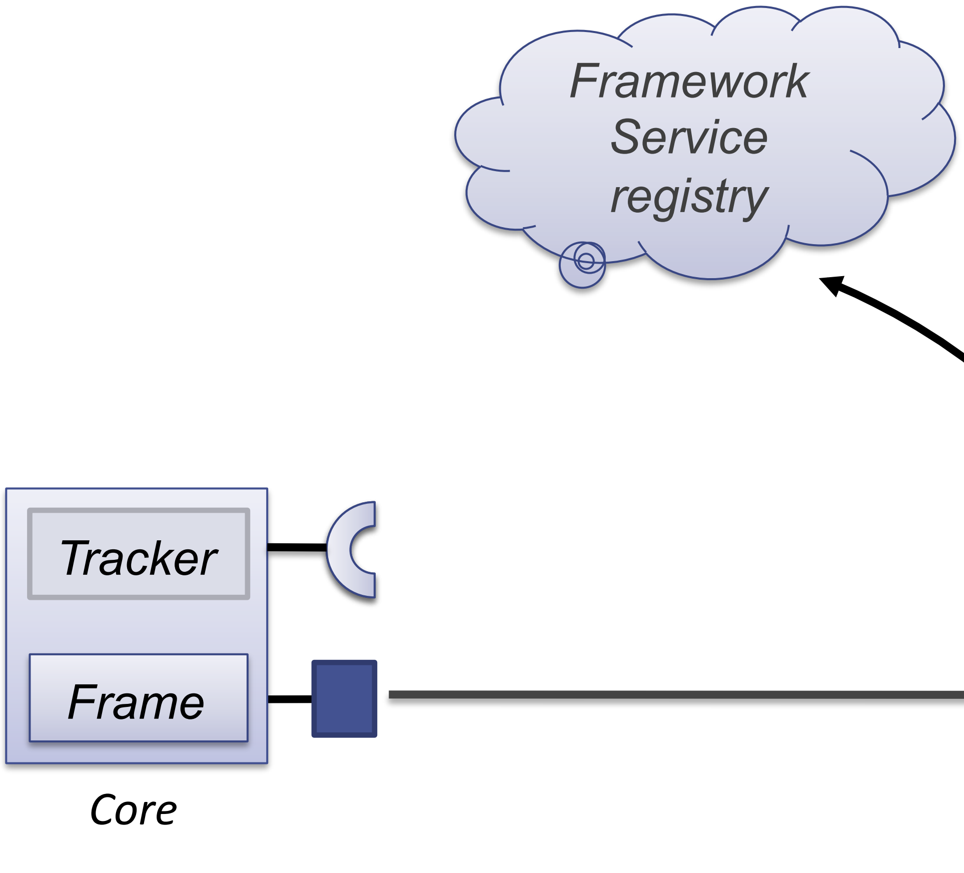
Core

Shape impl

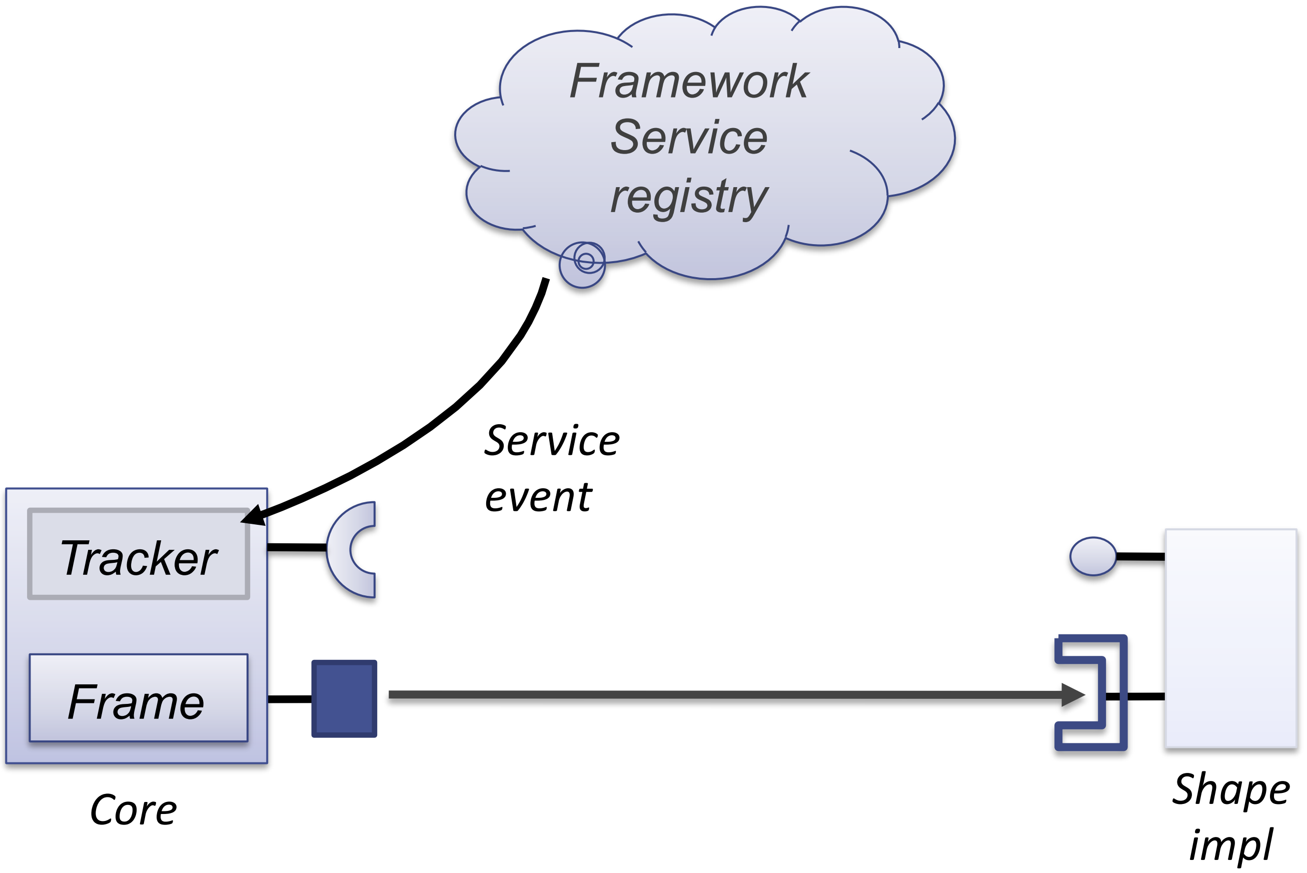
Whiteboard Pattern



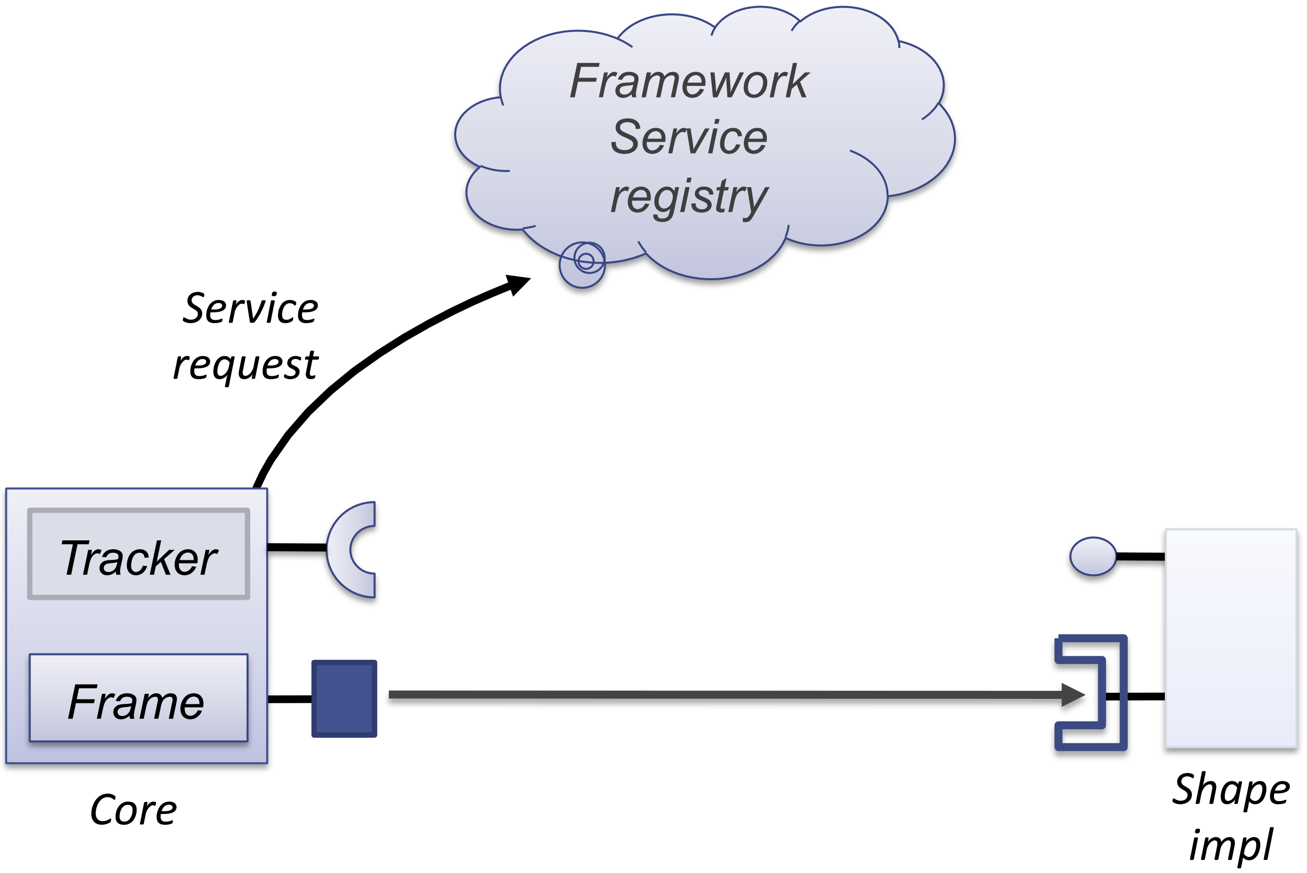
Whiteboard Pattern



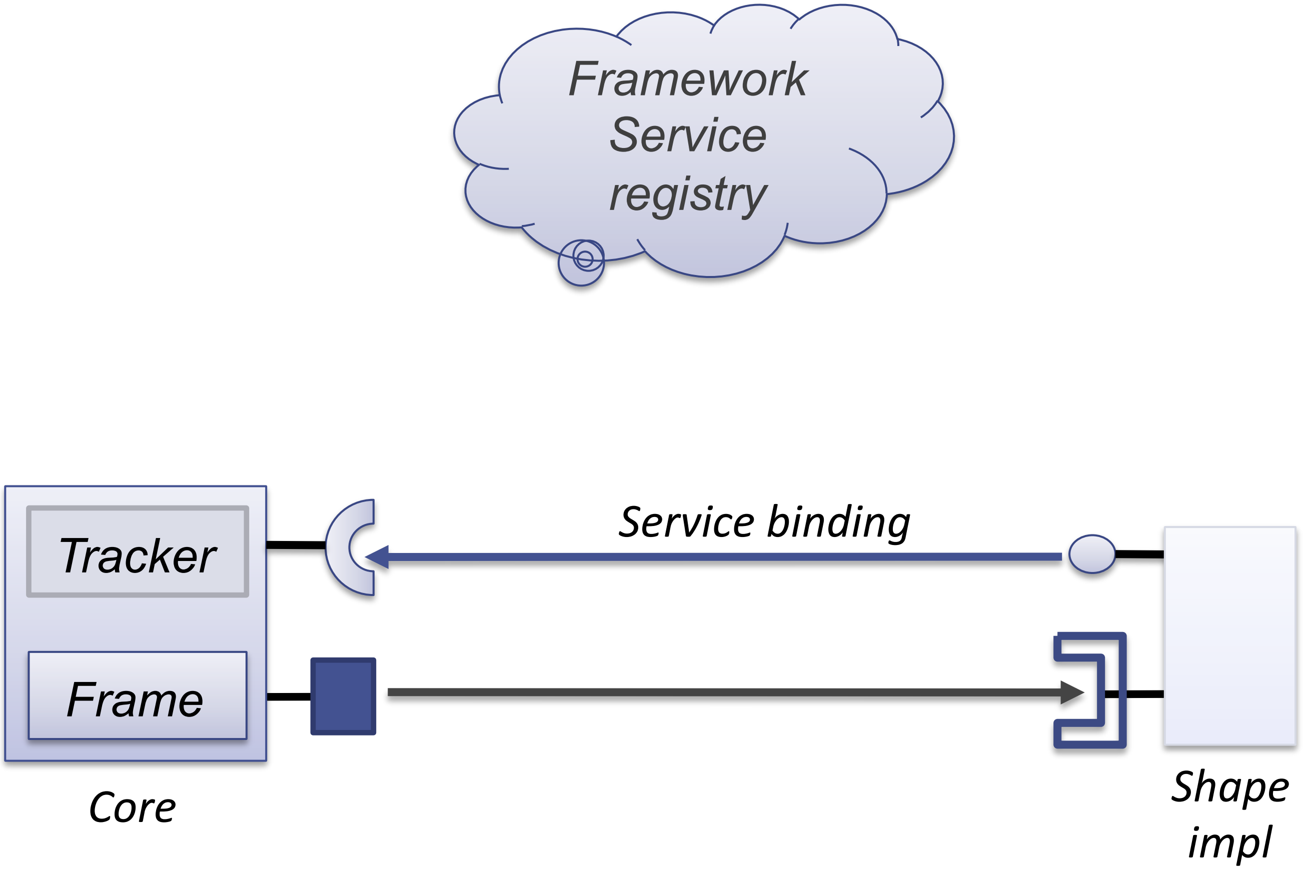
Whiteboard Pattern



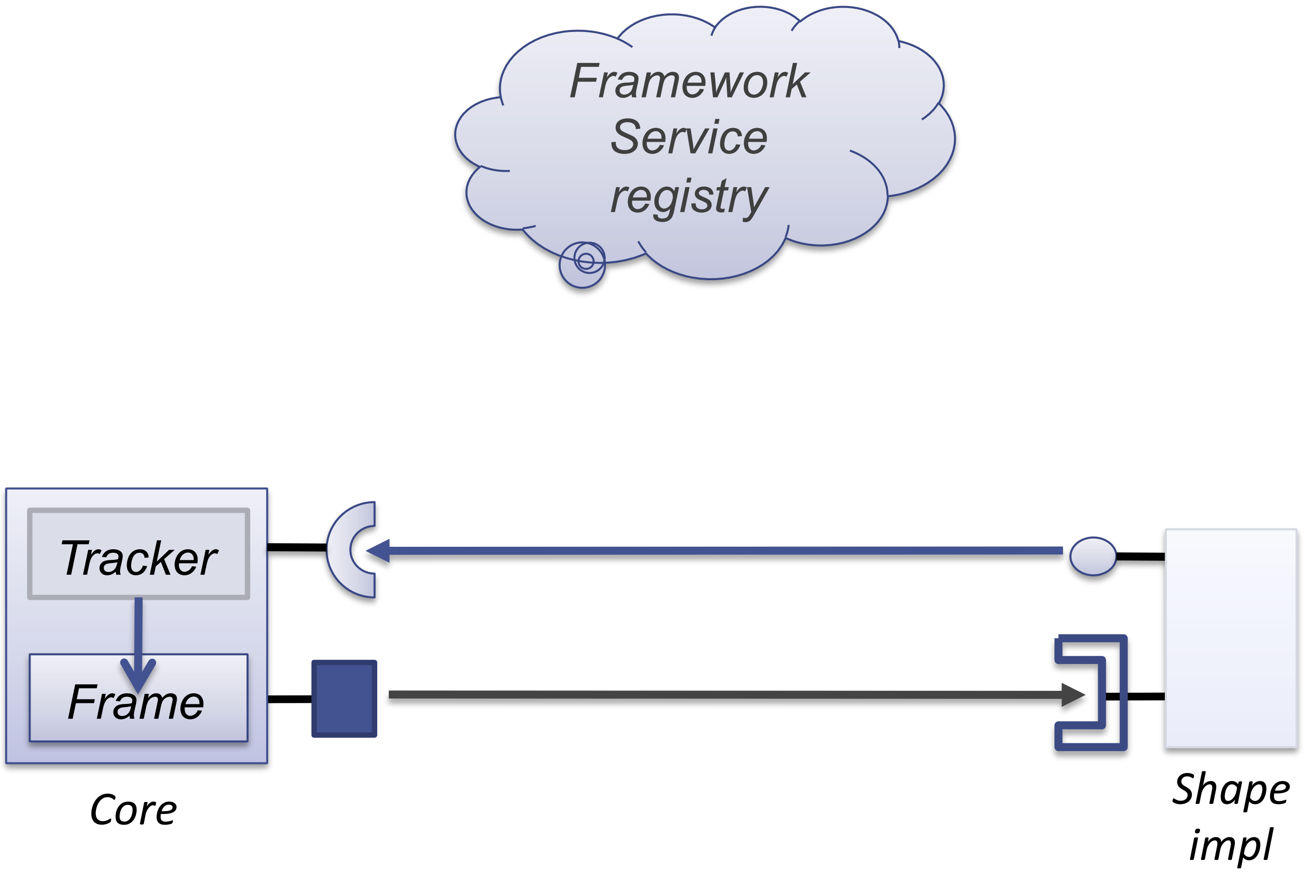
Whiteboard Pattern



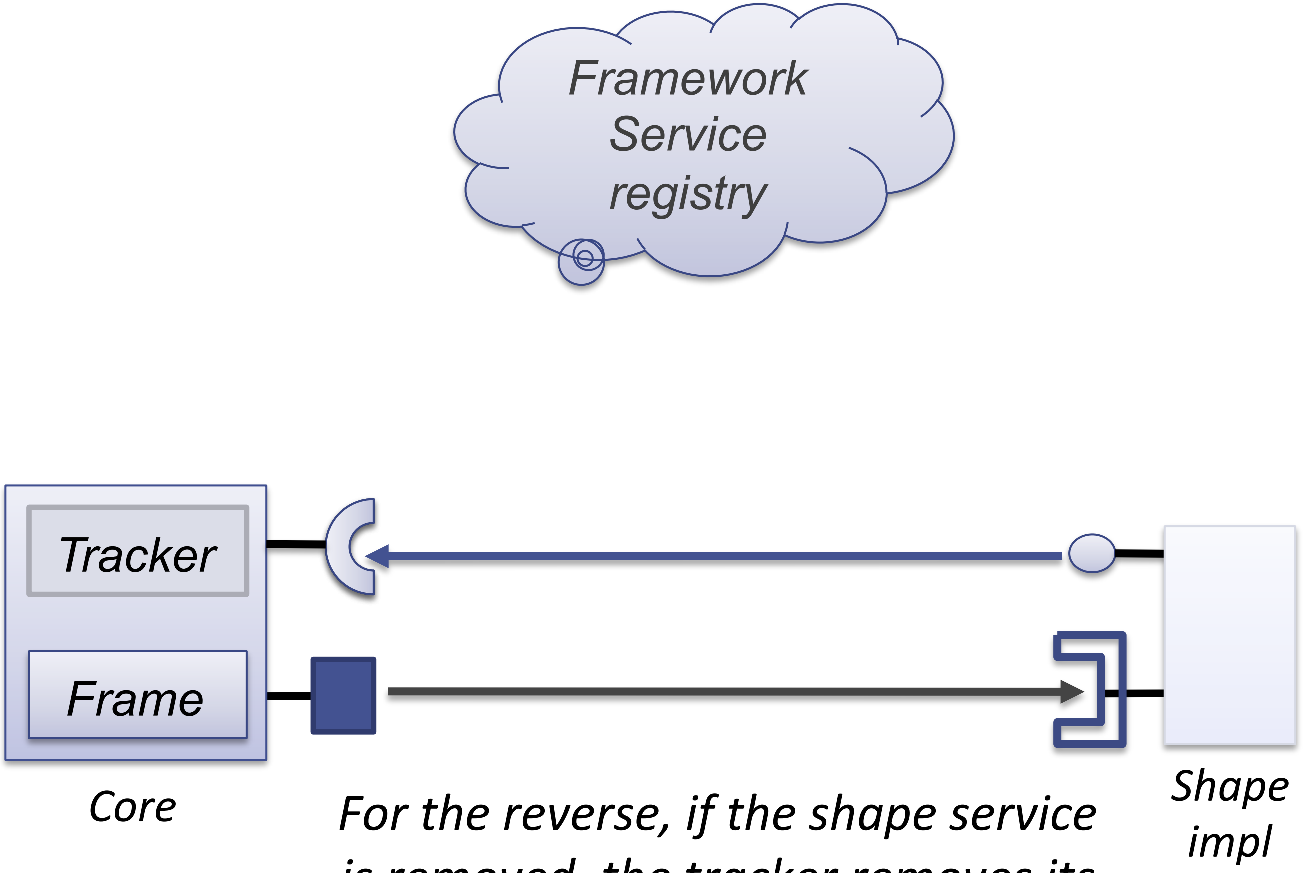
Whiteboard Pattern



Whiteboard Pattern



Whiteboard Pattern



For the reverse, if the shape service is removed, the tracker removes its associated shape.

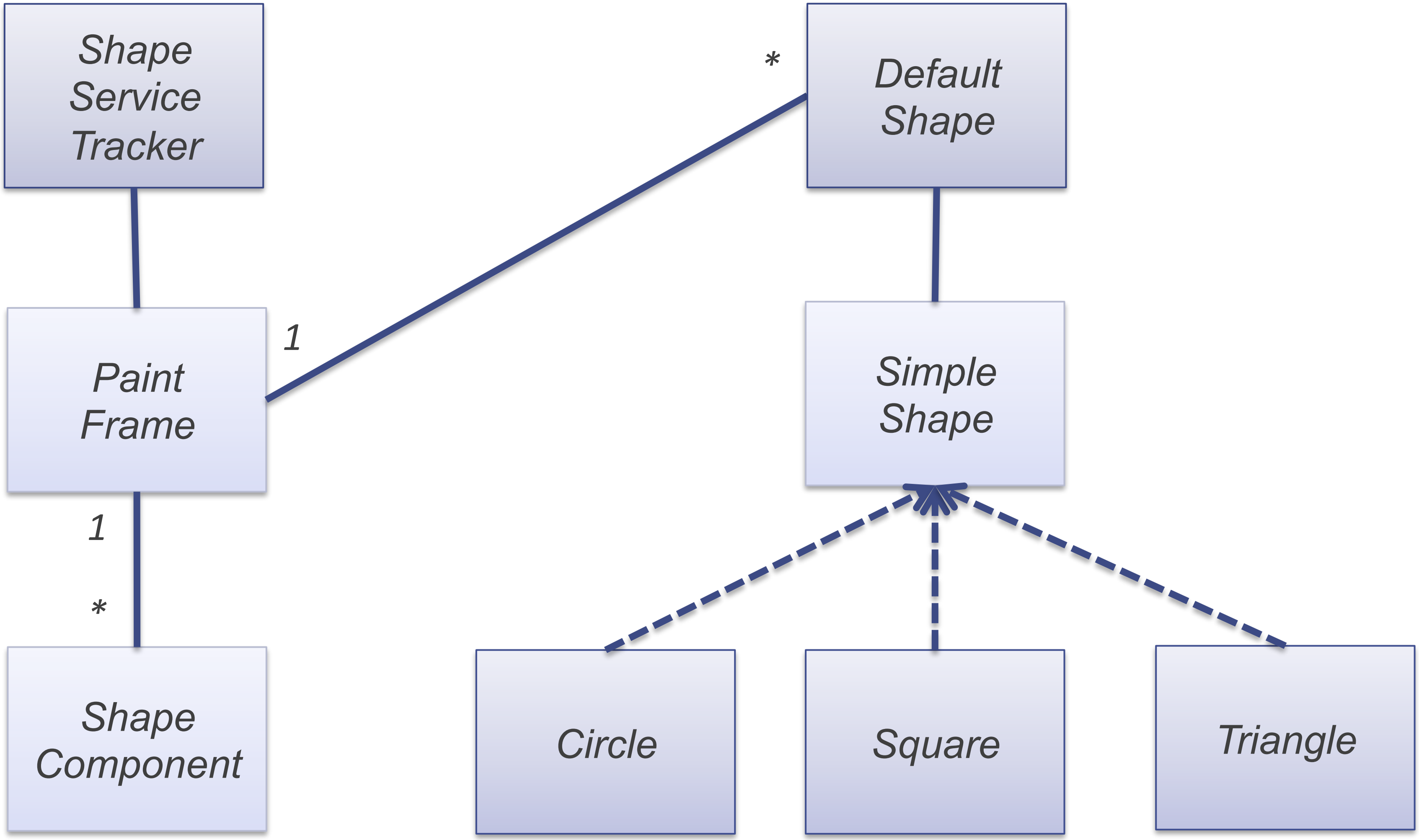
Service Paint Program Overview

Dynamically extensible paint program

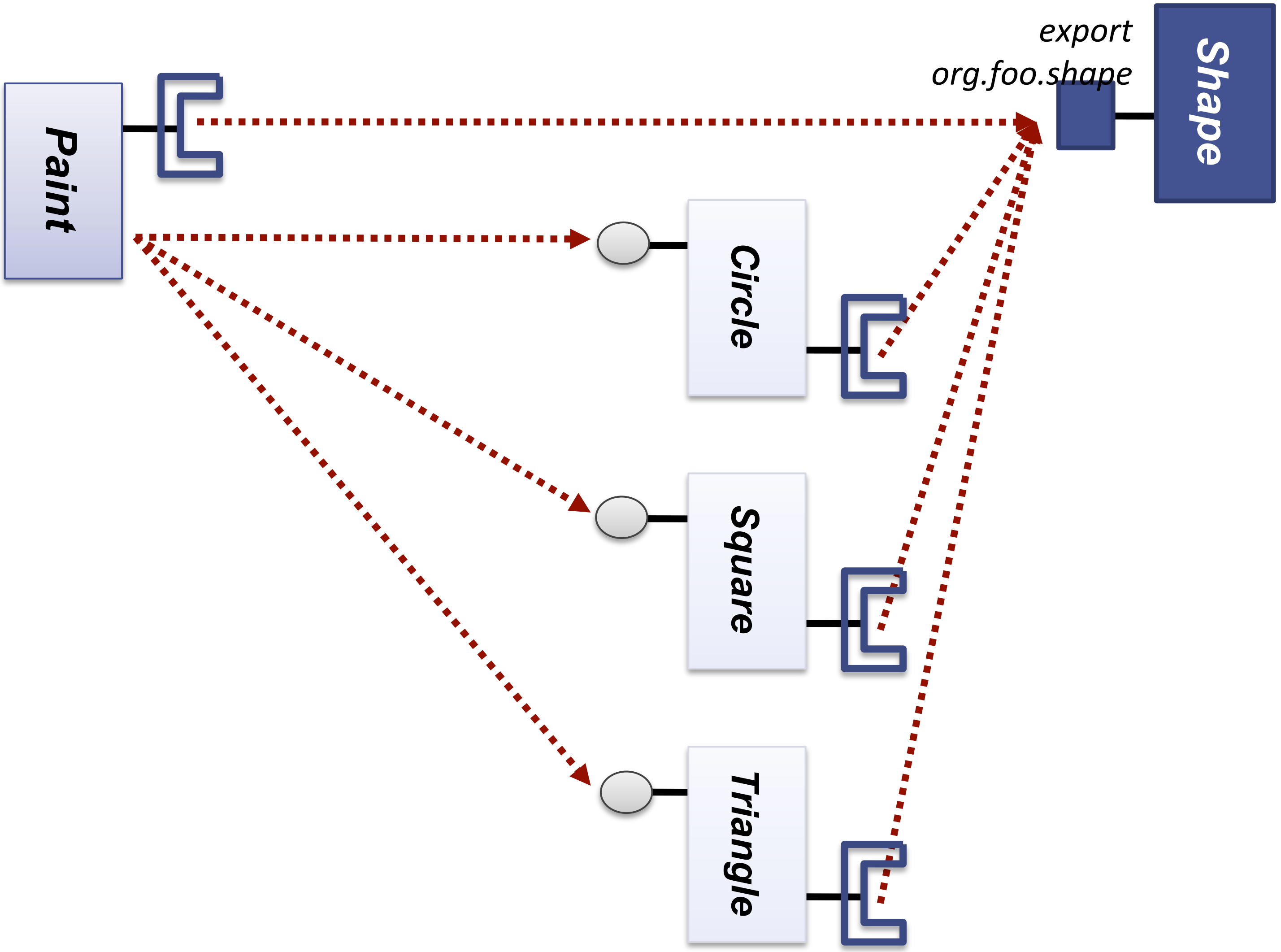
- Uses the whiteboard pattern to deliver shapes
- The paint bundle listens for shape services that come and go
- Uses service properties for the name and icon of the shape

Uses placeholder when shape has been used, but is currently unavailable because the service is not available

Service Paint Program Design (1/2)



Service Paint Program Design (2/2)



Challenges of Dynamism

Both bundles and services are dynamic

OSGi is inherently multi-threaded

This means you have to deal with the fact that

- Your application will likely see multiple threads
- Application components can appear or disappear at any time

There is help

- Service Tracker
- Service Component Model : Declarative Services, iPOJO, Blueprint

What a pain !

Modularity and Dynamism are two really interesting properties

- Using it looks a nightmare!
- Migrating to OSGi, looks terrible

All aspects are important in OSGI, especially hardest ones:

- Packaging
- Multithreading and synchronization
- Classloading

Pain Relievers

Packaging

- BND, BNDTools
- SpringSource Bundlor (wrapping)
- Apache Felix Sigil, Maven-Tycho…

Service Component Runtime (dependency injection)

- Declarative Services, Blueprint
- Apache Felix iPOJO

Enterprise OSGi

- Apache Aries
- Eclipse Gemini, Virgo

Administration tools and Deployment

- Web Console
- OBR

Packaging

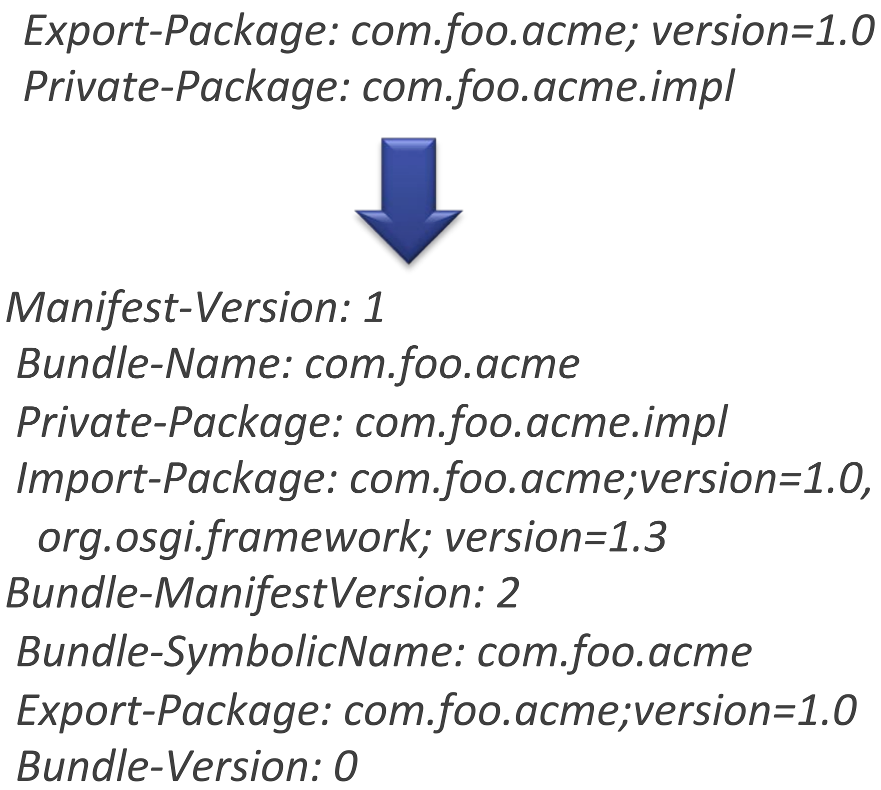
BND

BND

- How to make bundles easily
- Description of the bundle content in term of
  - Imported/Exported/Private packages
  - Resources
  - Embedded Jar
- Compute the correct metadata

Frontends

- Command Line
- Ant
- Maven (maven-bundle-plugin)
- BndTool (Eclipse Plugin)



```
Export-Package: com.foo.acme; version=1.0
Private-Package: com.foo.acme.impl
```

```
Manifest-Version: 1
 Bundle-Name: com.foo.acme
 Private-Package: com.foo.acme.impl
 Import-Package: com.foo.acme;version=1.0,
   org.osgi.framework; version=1.3
Bundle-ManifestVersion: 2
 Bundle-SymbolicName: com.foo.acme
 Export-Package: com.foo.acme;version=1.0
 Bundle-Version: 0
```

maven-bundle-plugin

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <instructions>
      <Export-Package>
       com.foo.acme; version=1.0.0
      </Export-Package>
      <Private-Package>
       com.foo.acme.impl
      </Private-Package>
    </instructions>
  </configuration>
</plugin>
```

The ‘how to find bundle’ dilemma

How to transform a plain Jar into a Bundle

- BND and Bundlor support this use case out of the box
- Common Strategy
  - Export all packages (except *.impl, *.internals)
  - Compute imports (as optional)

Is it good ?

- Do not manage the visibility, optionality, reflection
- Do not use services

Existing repositories

- Spring Source : https://ebr.springsource.com/repository/app/
- Service Mix : http://servicemix.apache.org/SMX4/bundles-repository.html
- Chameleon Common : http://wiki.chameleon.ow2.org/xwiki/bin/view/Main/WebHome
- A lot of projects are already OSGi-aware !

Advanced Service Handling

Service-Component Model

Why ?

- Simplification of the development model
  - Dynamism
  - Management
  - Reconfiguration
- Architectural view
- Allow to easily create sophisticated applications

Service-Component Model

Why ?

- Simplification of the development model
  - Dynamism
  - Management
  - Reconfiguration
- Architectural view
- Allow to easily create sophisticated applications

Service-Component models

- Infuse service-oriented mechanisms in a component model
- Provide
  - Simple development model
  - Architectural views, composition mechanisms

Existing Service Component Models

Declarative Services

- Specified in OSGi R4
- Define a declarative component model to deal with the service dynamism

Blueprint

- Specified in the OSGi Enterprise Profile
- Spring on the top of OSGi
- Beans can use services and be exposed as services

Apache Felix iPOJO

- POJO-based component model
- Extensible
  - Is not limited to dynamism
- Supports annotations
- The most advanced today
- http://ipojo.org

Providing the Shape service with iPOJO

```
@Component
@Provides
public class Circle implements SimpleShape {
        @ServiceProperty(name=SimpleShape.NAME_PROPERTY)
        private String name;

        @ServiceProperty(name=SimpleShape.ICON_PROPERTY)
        private ImageIcon icon;

        @Validate
        public void start() {
                icon = new ImageIcon(this.getClass().getResource
("circle.png")));
                name = "Circle";
        }

        public void draw(Graphics2D g2, Point p) {
            // Draw a circle
        }
}
```

Requiring shapes with iPOJO

```
@Component
@Instantiate
public class Host {

        public Host() {
         // Create the frame...
        }

        @Bind
        public synchronized void bindShape(SimpleShape shape) {
         // Update the frame
        }

        @Unbind
        public synchronized void unbindShape(SimpleShape shape) {
         // Update the frame
        }

        //...
}
```

What iPOJO manages for you ?

Services

- Dependencies: dynamism, synchronization
- Service Providing: publication, serving, service properties, updates

Lifecycle

- Instance lifecycle
- Callbacks
- Controllable!

Others

- Asynchronous communication
- Extender pattern, Whiteboard pattern
- JMX
- Transaction, JPA…
- **Extensible!**

What iPOJO manages for you ?

Factory / Instance distinction

- @Component => Component Type
- You can create several instance from the same type with different configurations
  - 3 shapes (instances), 1 component type

Management

- Interaction with the OSGi Config Admin
- WebConsole Plugin
- Introspectable

Injection

- Based on bytecode enhancement
  - Offline or Install-time
  - Tested on a lot of JVMs
- Field injection
- Method callback
- Constructor injection

Where iPOJO is used?

Applications Servers

- Home Gateway
- RFID Suite
- JEE Application Server (OW2 JOnAS)

System

- Insurance softwares
- Embedded devices

Others

- Desktop applications (Swing, SWT, QT…)
- Android
- Mobile Games (uGASP)

Enterprise OSGi

Enterprise OSGi

Integrate JEE Technologies into OSGi: One goal, Two trends:

Enterprise OSGi : the specification

- First specification released in March 2010
- Defines
  - Web Applications (WABs)
  - Remote Services, SCA Definitions
  - J* : JDBC, JNDI, JTA, JPA, JMX

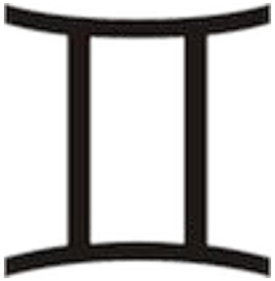
Hybrid application servers

- JEE application servers relying on OSGi and exposing OSGi
  - OW2 JoNAS, Oracle Glassfish, IBM Websphere
  - Redhat Jboss
- OSGi Applications using JEE services & JEE components using OSGi services
- http://blog.akquinet.de/2009/07/27/jonas-showcase-having-the-best-of-jee-and-osgi/

Eclipse Gemini

Software Suite containing Enterprise-technologies support

- Reference Implementation for many of the Enterprise OSGi Specifications
- Contains
  - Blueprint
  - Web Container
  - JPA
  - JDBC
  - JMX
  - JNDI
  - …



Apache Aries

Implementations and extensions of the Enterprise OSGi Specifications

- Contains
  - Blueprint
  - JPA
  - JTA
  - JDBC
  - JMX
  - SPI
  - JNDI
  - …
- Assembly / Application format: EBA

Eclipse Virgo

Originally SpringSource dmServer (dynamic modules)

- *OSGi* web container
  - Application description, deployment and management : Plans, isolated, atomics
  - Provisioning: PARs
  - Legacy libraries
  - Administration
  - Toolings

Others

Remote Services

- CXF Distributed OSGi
  - Web Services
- OW2 Chameleon Rose
  - Technology agnostics
  - JSONRPC, Web Service, REST (Jersey)…

Distributed Events

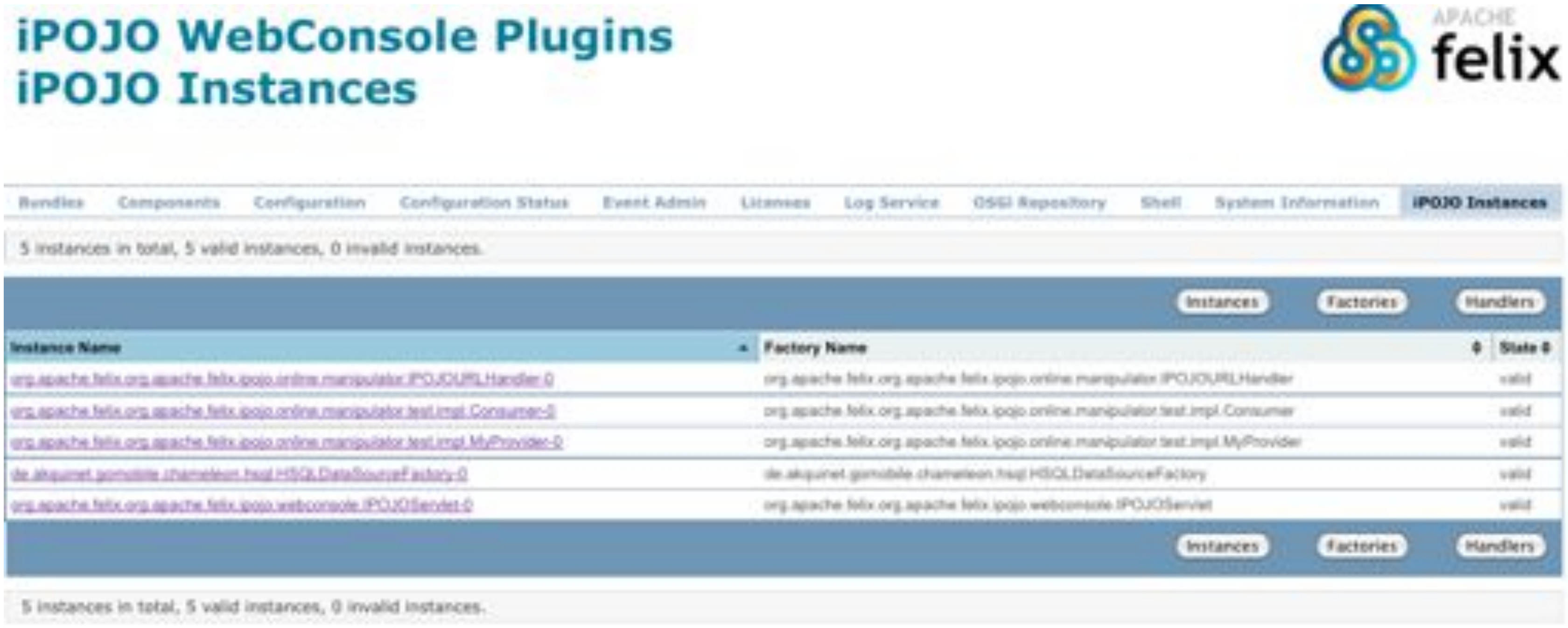
- Event Admin bridges
- OW2 Rose JMS Bridges (activeMQ, HornetQ, Joram)

ESBs

- Service Mix
- Camel

Administration

Apache Felix Web Console



iPOJO WebConsole Plugins
iPOJO Instances

5 instances in total, 5 valid instances, 0 invalid instances.

Instance Name ▲	Factory Name ⇕	State ⇕
org.apache.felix.org.apache.felix.ipojo.online.manipulator.IPOJOURLHandler-0	org.apache.felix.org.apache.felix.ipojo.online.manipulator.IPOJOURLHandler	valid
org.apache.felix.org.apache.felix.ipojo.online.manipulator.test.impl.Consumer-0	org.apache.felix.org.apache.felix.ipojo.online.manipulator.test.impl.Consumer	valid
org.apache.felix.org.apache.felix.ipojo.online.manipulator.test.impl.MyProvider-0	org.apache.felix.org.apache.felix.ipojo.online.manipulator.test.impl.MyProvider	valid
de.akquinet.gomobile.chameleon.hsql.HSQLDataSourceFactory-0	de.akquinet.gomobile.chameleon.hsql.HSQLDataSourceFactory	valid
org.apache.felix.org.apache.felix.ipojo.webconsole.IPOJOServlet-0	org.apache.felix.org.apache.felix.ipojo.webconsole.IPOJOServlet	valid

5 instances in total, 5 valid instances, 0 invalid instances.

Apache Ace

Software Distribution framework

- Based on OSGi
  - For OSGi but not only

Features

- Creation of deployment package
  - Allow to push installation to a set of gateway
  - Support installations / updates / uninstallations
  - Pull and Push
- Dependency Management
  - Smart deployment
- Scalability
  - Not limited in terms of administered gateways
- http://incubator.apache.org/ace/

Others

There are a lot of tools

Provisioning

- OSGi Bundle Repository: Deployment solution to resolve dependencies
- Eclipse P2

Remote administration

- JMX
- SSH Remote Shell
- VisualVM OSGi Plugin (developed on OW2 Chameleon)

Conclusion

OSGi can be a really good technology

Mind the Gap

Modularity and Dynamism are hard

Don’t except something easy

- Modularity is **HARD** !
  - Writing modular code is hard
  - Modularizing existing code is a lot harder

- Bigger is your codebase, harder it will be
  - Complex code (reflection, dynamic loading) can be really a nightmare

- Think about what your are doing !
  - What’s the point ?
  - Why are you doing such kind of fancy mechanism
    - Forget the “Just For Fun” answer
    - **Keep it simple**

OSGi

We've seen all OSGi has to offer

- Module layer
- Lifecycle layer
- Service layer

While there are plenty of more details to these layers, you should now be familiar with the most important parts

- The most commonly used/needed features
- The most commonly used patterns

A lot of tools are available, use them !

- Existing services
- Component Models

Writing Modular Code

Carefully mange your packages

- Avoid split package
- Separate specification / implementation
- Package specifications in their own bundle

Avoid Class.forName

- No global visibility in OSGi
- If really need be, give the correct classloader

Use Services

- Use Services !!!

Use component models

- Don't use the OSGi API Directly

Recommendations

OSGi in Action

Richard S. Hall, Karl Pauls, Stuart McCulloch, David Savage

Questions ?

Karl Pauls
karl.pauls@akquinet.de
Bülowstraße 66, 10783 Berlin
+49 151 226 49 845

Dr. Clement Escoffier
clement.escoffier@akquinet.de
Bülowstraße 66, 10783 Berlin
+49 175 2467717