# Just stay loose with OSGi and Apache Felix

**Carsten Ziegeler**
cziegeler@apache.org

Apache Con NA Presentation – November 2010 - Atlanta

Leading the Wave
of Open Source

ApacheCon

# About

- Member of the ASF
  - Sling, Felix, Portals, Sanselan, Excalibur, Incubator (Cocoon)
  - PMC: Felix, Portals, Sling, Incubator, Excalibur (Chair)
- RnD Team at Adobe(Day Software)
- Article/Book Author, Technical Reviewer
- JSR 286 Spec Group (Portlet API 2.0)

## Agenda

1 Motivation

2 And Action...

3 Why OSGi?

4 Apache Felix

5-7 Bundles, Services, Dynamics

8 Famous Final Words

## 1 Motivation

## Motivation

- Modularity is key
  - Manage growing complexity
  - Support dynamic extensibility
- No solution in standard Java
  - OSGi: tried and trusted
- Embrace change – Embrace OSGi
  - Only a few concepts – easy to get started
  - Minor “overhead”
- Loose Coupling
  - Modules and Services

## 2 And Action…

## Paint Program

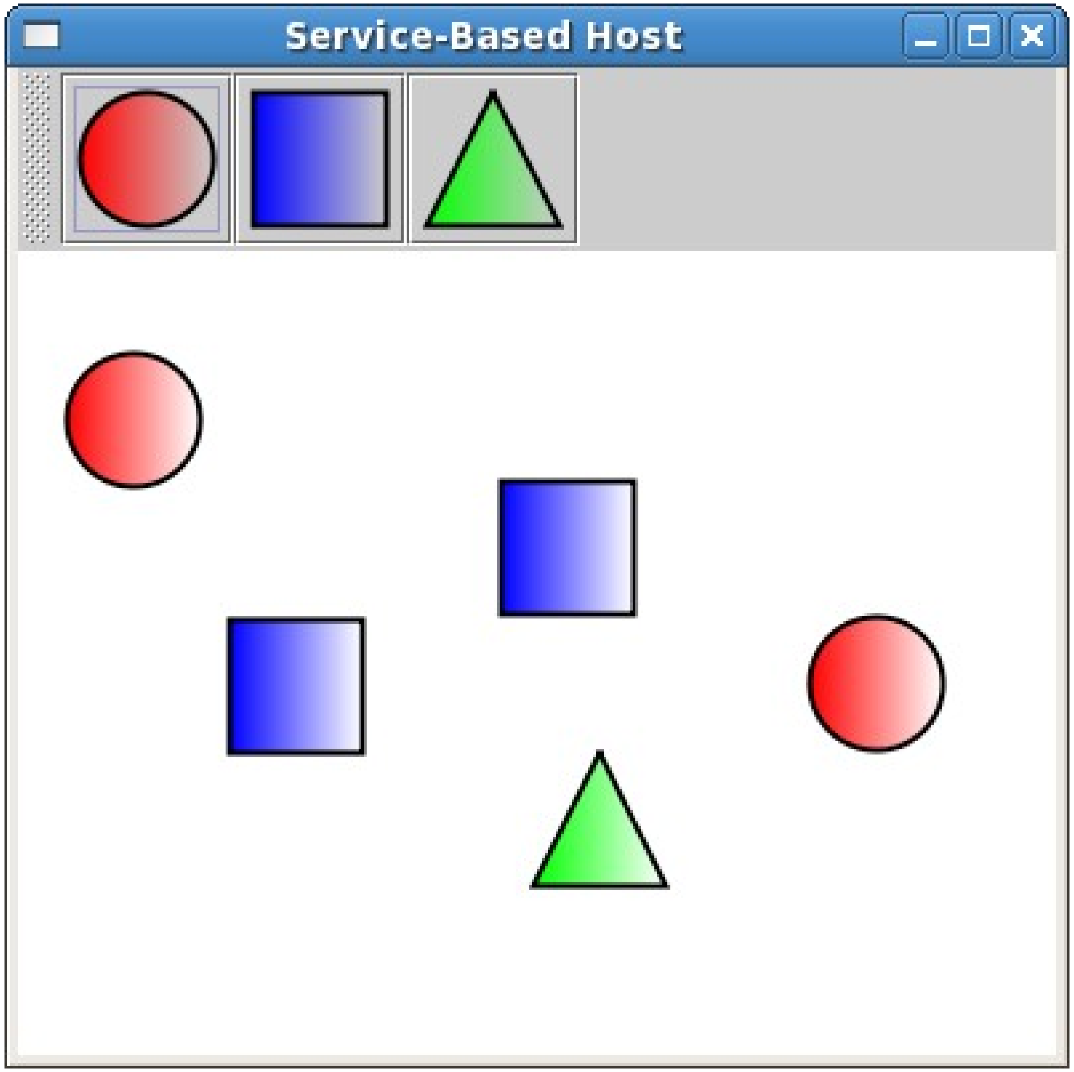
- Swing-based paint program
- Interface `SimpleShape` for drawing
  - Different implementations
  - Each shape has name and icon properties
  - Available shapes are displayed in tool bar
- Select shape and then select location
  - Shapes can be dragged, but not resized
- Support dynamic deployment of shapes

## Shape Abstraction

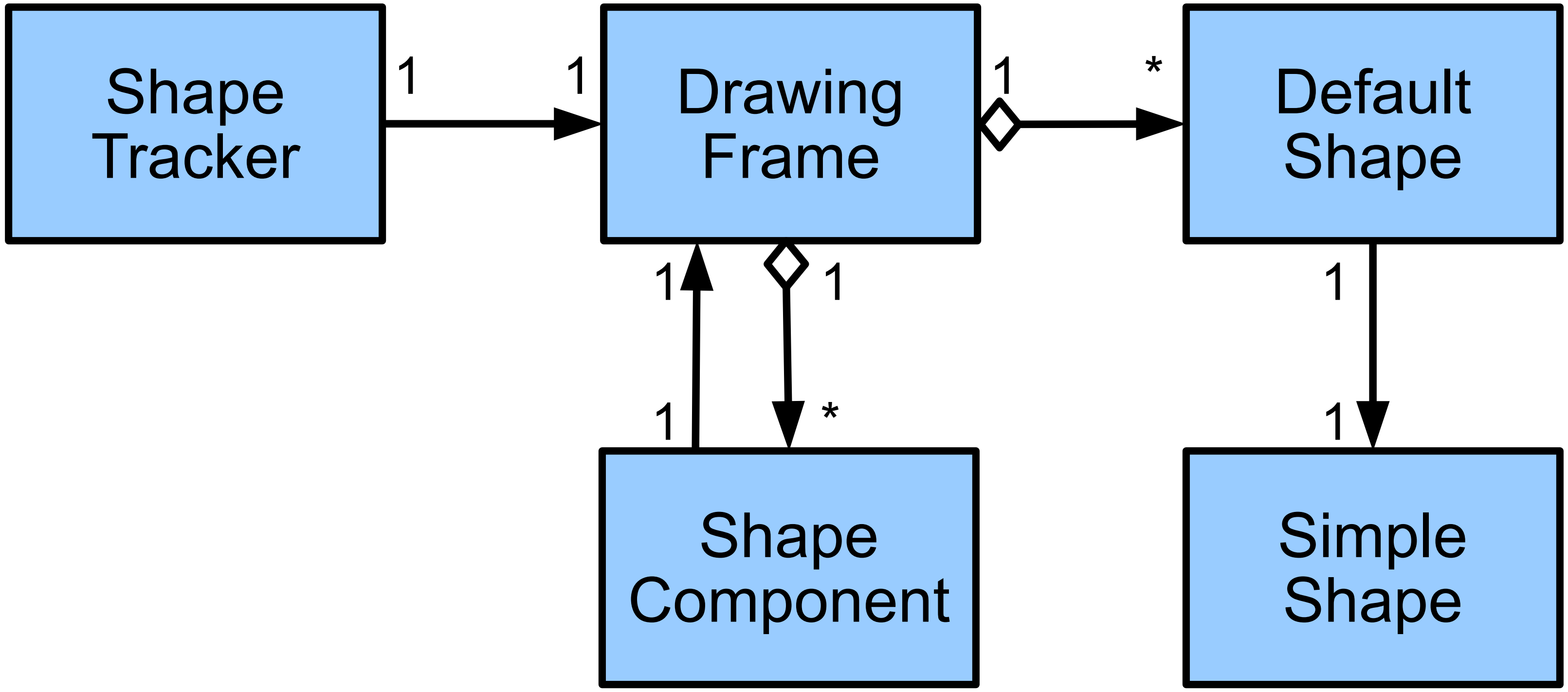
- Conceptual `SimpleShape` interface

```
public interface SimpleShape
{
    /**
     * Method to draw the shape of the service.
     * @param g2 The graphics object used for
     *           painting.
     * @param p The position to paint the shape.
    **/
    public void draw(Graphics2D g2, Point p);
}
```

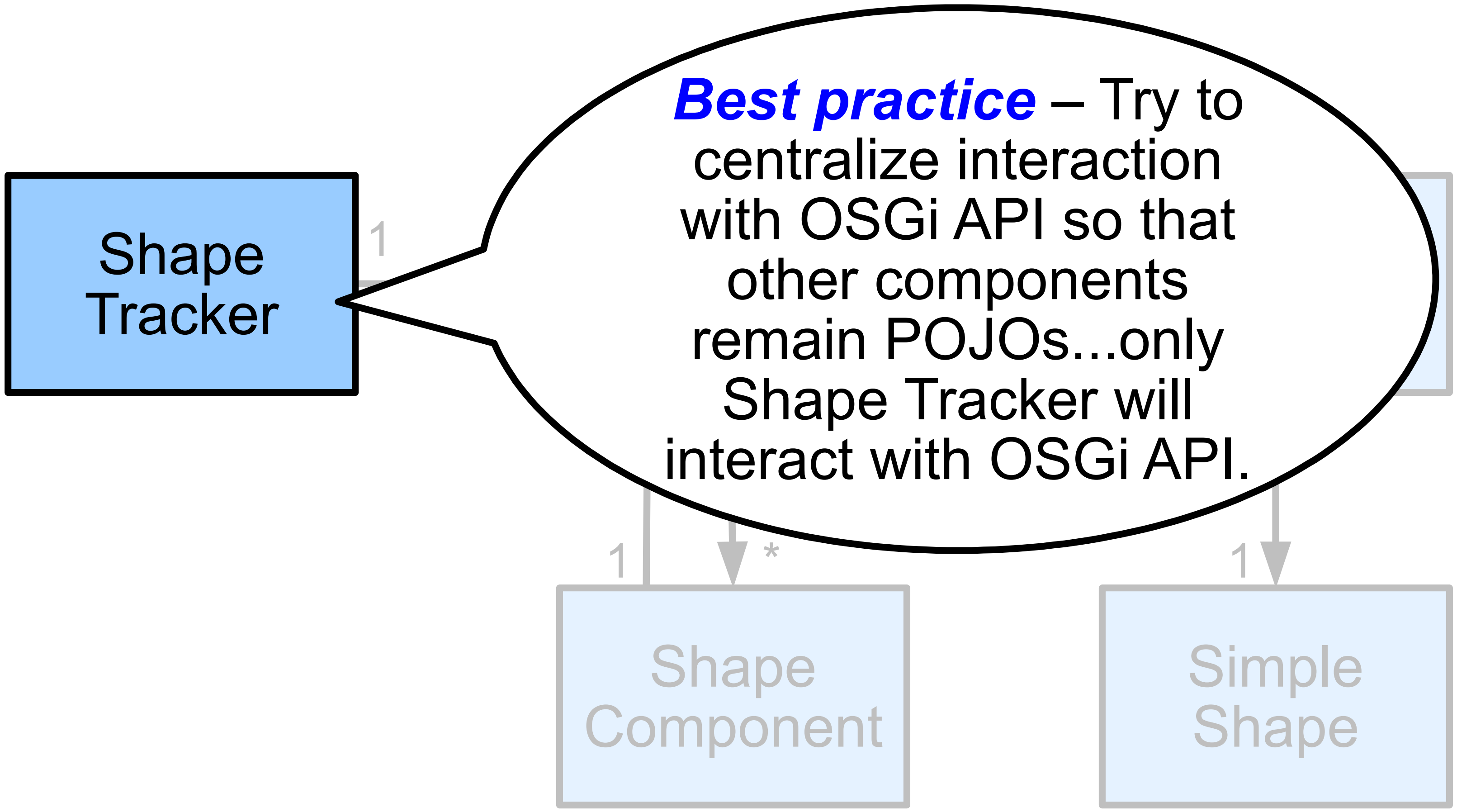
# Paint Program Mock Up



# High-Level Architecture

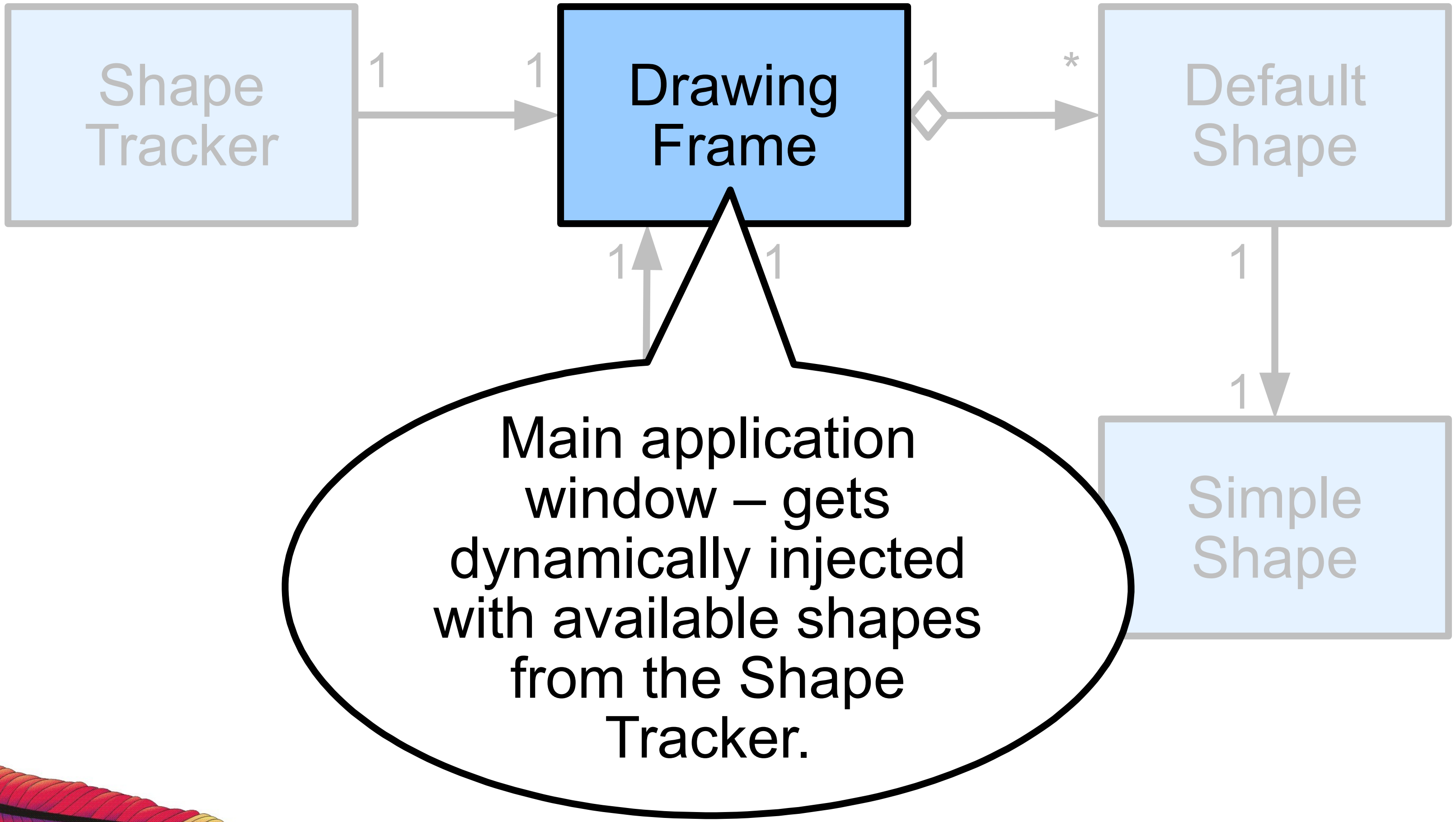


# High-Level Architecture

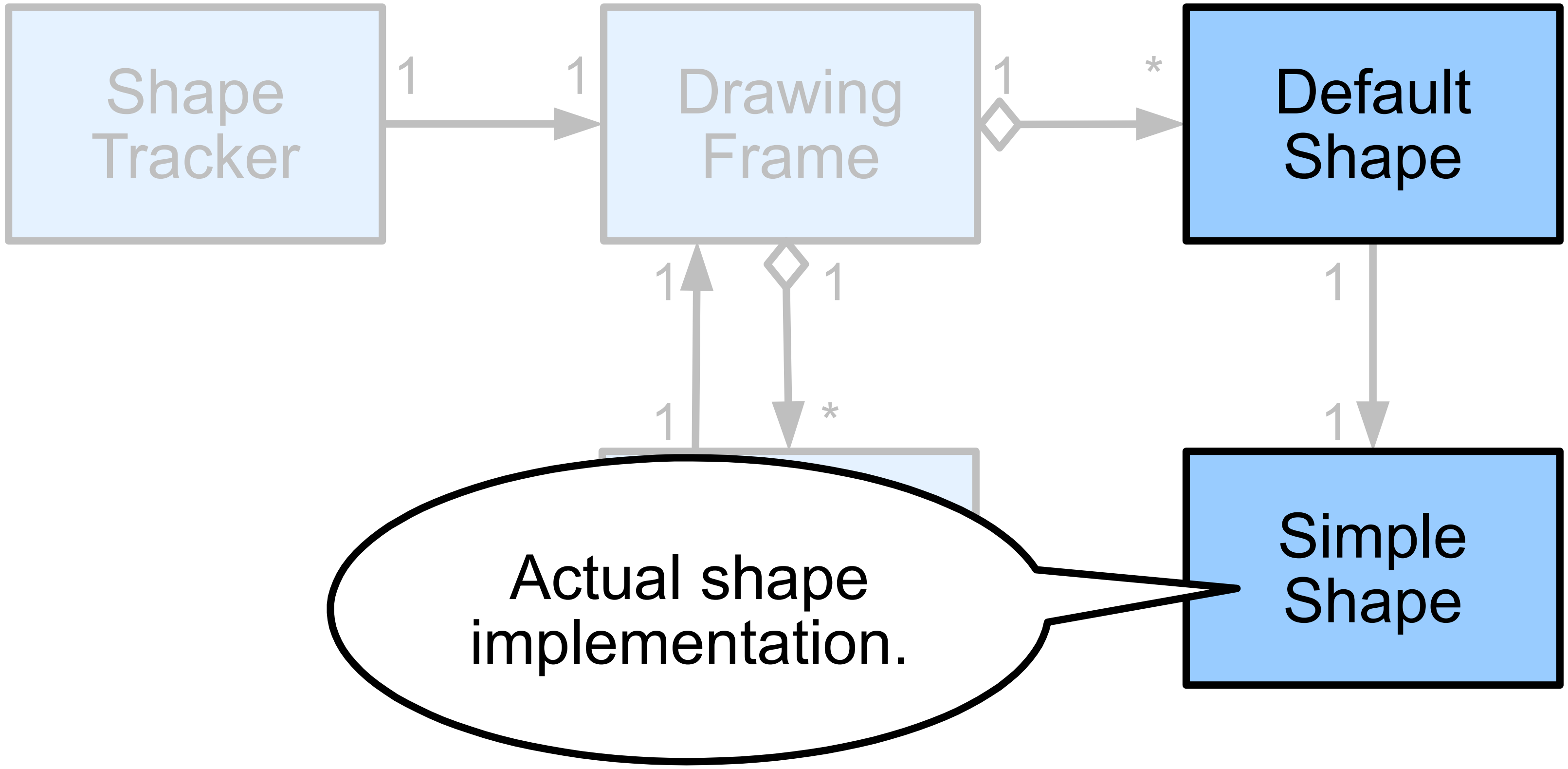


***Best practice*** – Try to centralize interaction with OSGi API so that other components remain POJOs...only Shape Tracker will interact with OSGi API.

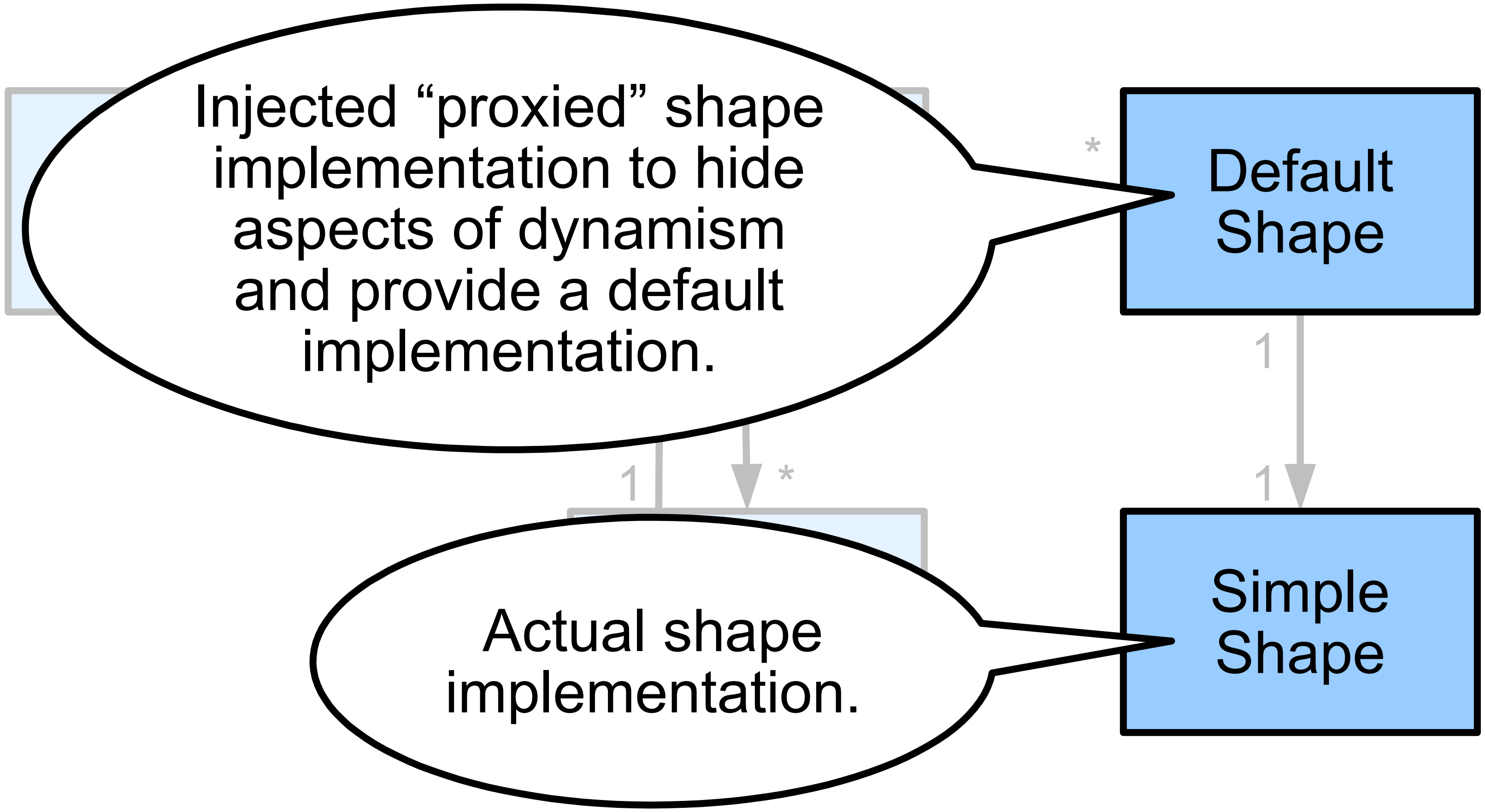
# High-Level Architecture


Shape Tracker
1
1
Drawing Frame
1
*
Default Shape
1
1
1
1
Simple Shape
Main application window – gets dynamically injected with available shapes from the Shape Tracker.

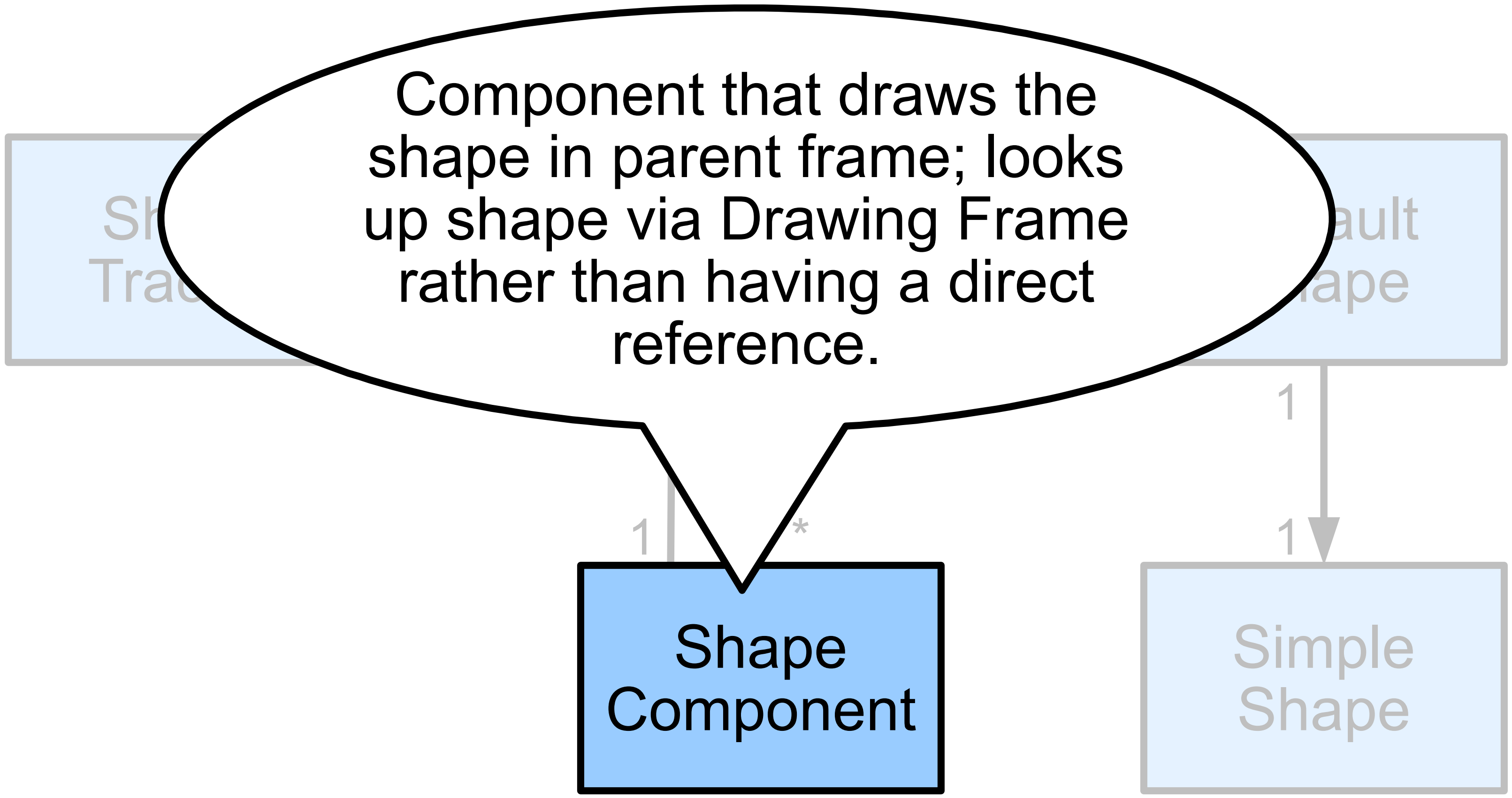
# High-Level Architecture



# High-Level Architecture


Injected “proxied” shape implementation to hide aspects of dynamism and provide a default implementation.
Default Shape
*
1
1
1
*
Actual shape implementation.
Simple Shape

# High-Level Architecture

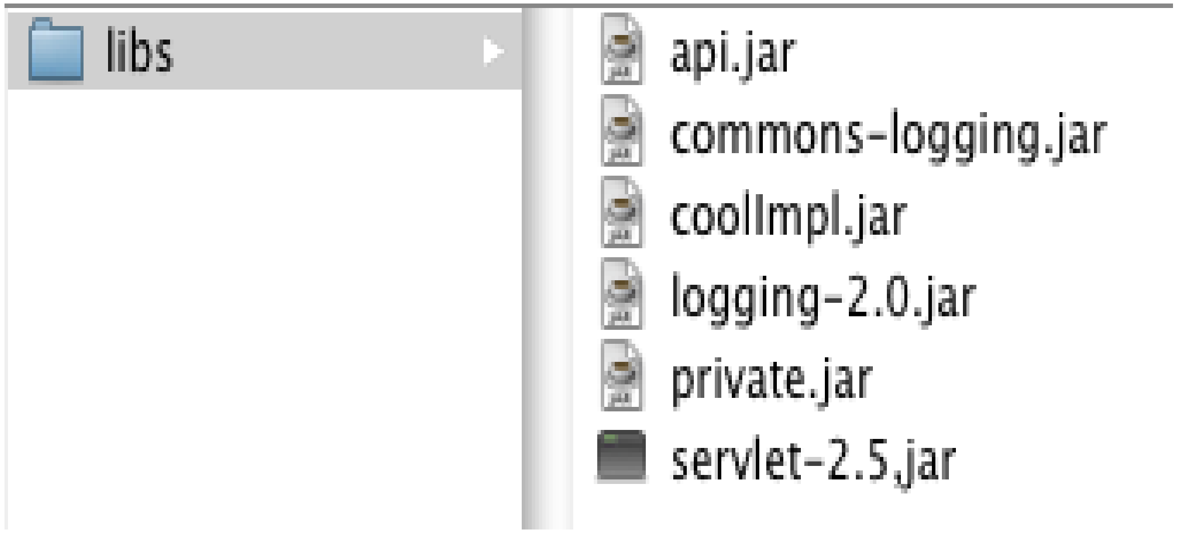


Component that draws the shape in parent frame; looks up shape via Drawing Frame rather than having a direct reference.

## LIVE DEMO

## 3 Why OSGi?

# Class Path Hell

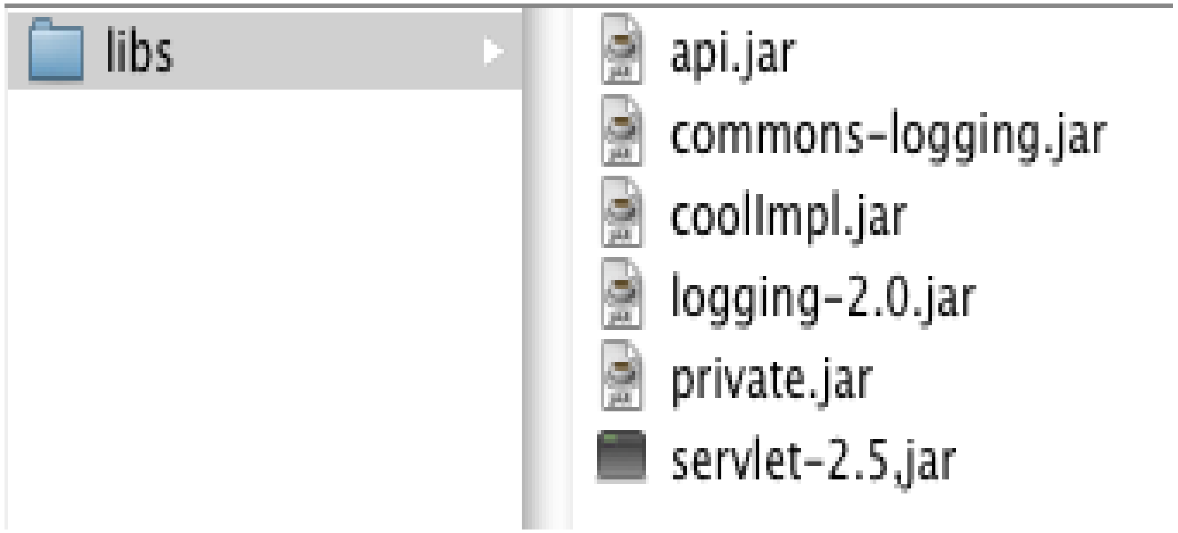


libs

api.jar
commons-logging.jar
coolImpl.jar
logging-2.0.jar
private.jar
servlet-2.5.jar

- Can you spot some potential problems?

# Class Path Hell


libs

api.jar
commons-logging.jar
coolImpl.jar
logging-2.0.jar
private.jar
servlet-2.5.jar

- What libs are used? Versions?
- Which jar is used? Version?
- No difference between private and public classes

## Java's Shortcomings

- Simplistic version handling
  - “First” class(!) from class path
- Split packages by default
  - The complete class path is searched
  - Leads to shadowing or version mixing
- Implicit dependencies
  - Dependencies are implicit in class path ordering
  - JAR files add improvements for extensions, but cannot control visibility

## Java's Shortcomings

- Missing module concept
  - Classes are too fine grained
  - Packages are too simplistic
- Limited scoping mechanisms
  - No module access modifier
  - Impossible to declare all private stuff as private
- No deployment/lifecycle support

## OSGi Technology

- Adds modularity and dynamics
  - Module concept
    - Explicit sharing (importing and exporting)
  - Automatic management of code dependencies
    - Enforces sophisticated consistency rules for class loading
  - Life-cycle management
    - Manages dynamic deployment and configuration
- Service Registry
  - Publish/find/bind

## 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)

## 4 Apache Felix

## Apache Felix

- Top-level project (March 2007)
- OSGi R4 (R4.2) implementation
  - Framework (frequent releases)
  - Services (continued development)
    - Log, Package Admin, Event Admin, Configuration Admin, Declarative Services, Meta Type, Deployment Admin (and more)
  - Certified Platform!
- Tools
  - Maven Plugins, Web Console, iPojo
  - Karaf (own TLP now), Sigil

## Apache Felix

- Growing community
  - Diverse and healthy!
  - Several code grants and contributions
  - Various (Apache) projects use Felix / have expressed interest in Felix and/or OSGi
    - e.g., ServiceMix, Directory, **Sling**, Tuscany
- "Roadmap"
  - Implementing upcoming R4.3
  - Tooling (Web Console, Karaf, Sigil)

## 5 OSGi – Part 1 Bundles

# OSGi Framework

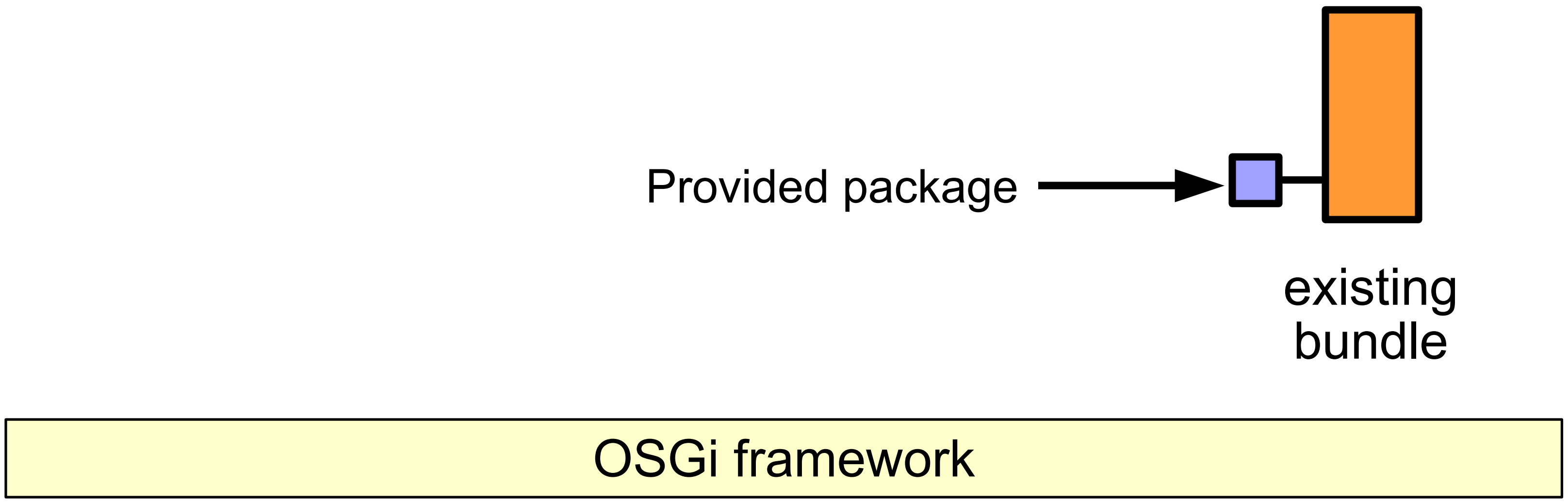
- Component-oriented framework
- Module concept: Bundles
  - Separate class loader -> graph
  - Package sharing and version management
  - Life-cycle management and notification
- Dynamic!
  - Install, update, and uninstall at runtime
- Runs multiple applications and services in a single VM

## OSGi Modularity

- Explicit code boundaries and dependencies
  - Package imports and exports
- Multi-version support
  - Version ranges for dependencies
- Class space is managed by OSGi
- Managed life cycle
  - Dynamic install, update, uninstall

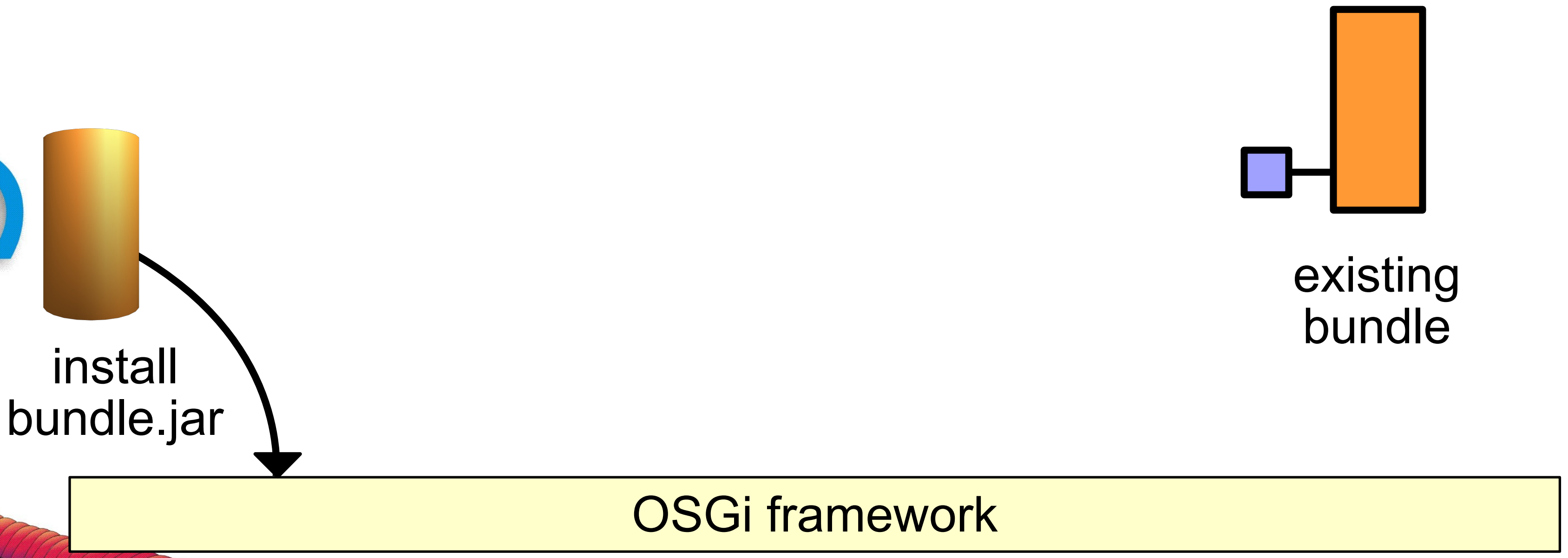
# OSGi Modularity - Example

- Dynamic module deployment and dependency resolution



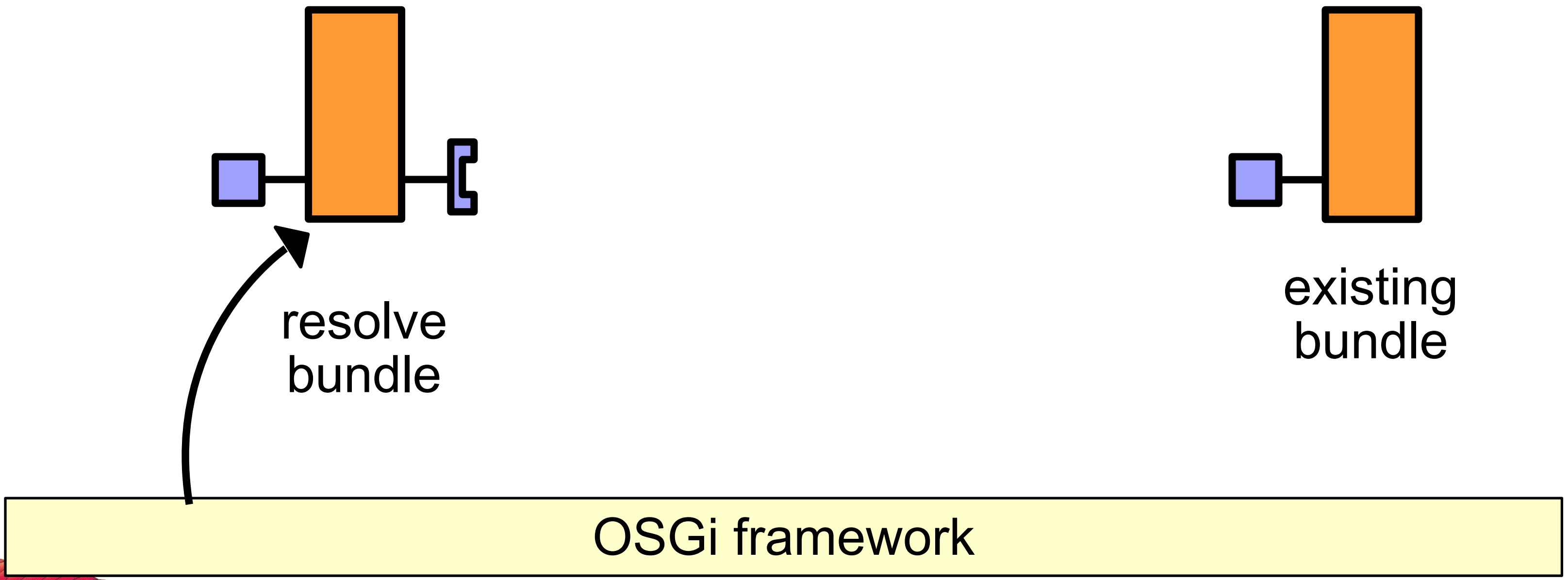
## OSGi Modularity - Example

- Dynamic module deployment and dependency resolution



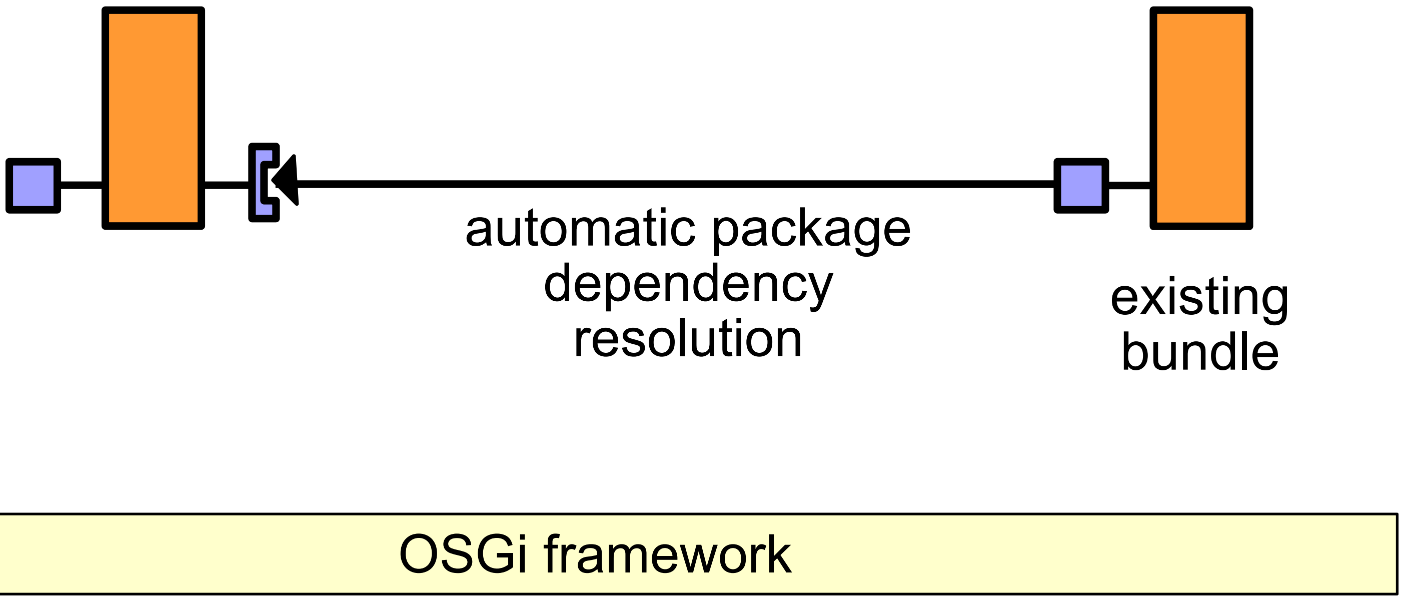
## OSGi Modularity - Example

- Dynamic module deployment and dependency resolution



# OSGi Modularity - Example

- Dynamic module deployment and dependency resolution



# Loose Coupling I

- Loose Coupling of bundles
  - Package imports and exports with versions
- Independent from bundle organization
  - "Someone" provides the package
  - Rebundling
- Error management for unresolved bundles
- Requires
  - Thinking modular!
  - Proper meta data
  - Consistent version management

Not tied to OSGi

## Think Modular

- This is not a new concept or latest hype!
- Think about modularity!
  - What is your API?
  - Create a clean package space
- Make things only public if necessary/used
  - Starting with private stuff going public is easy
- Use proper versioning contracts
  - Eclipse
  - OSGi Alliance: Semantik Versioning

# Creating a Bundle

- Plain old JAR with additional metadata in the manifest
  - Bundle identifier, version, exports, imports
- Tools
  - Text editor (Manifest)
  - Eclipse (PDE)
  - Bundle packaging tools
    - **BND** from Peter Kriens
    - Apache Felix ***maven-bundle-plugin*** based on BND

## Maven is Your Friend

- Apache Felix Maven Bundle Plugin
- Creates metadata based on POM
  - Automatically: import packages
  - Manually: export and private packages
- Analyses classes for consistency
- Allows to include dependencies
- Creates final bundle JAR file

## Maven Bundle Plugin Sample

```
<artifactId>org.apache.sling.engine</artifactId>
<packaging>bundle</packaging>
<version>2.0.7-SNAPSHOT</version>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <extensions>true</extensions>
      <configuration>
        <instructions>
          <Export-Package>
            org.apache.sling.engine;version=${pom.version}
          </Export-Package>
          <Private-Package>
            org.apache.sling.engine.impl
          </Private-Package>
          <Embed-Dependency>
            commons-fileupload
          </Embed-Dependency>
        </instructions>
```

# Maven Bundle Plugin Sample

```
<artifactId>org.apache.sling.engine</artifactId>
<packaging>bundle</packaging>
<version>2.0.7-SNAPSHOT</version>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <extensions>true</extensions>
      <configuration>
        <instructions>
          <Export-Package>
            org.apache.sling.engine;version=2.0
          </Export-Package>
          <Private-Package>
            org.apache.sling.engine.impl
          </Private-Package>
          <Embed-Dependency>
            commons-fileupload
          </Embed-Dependency>
        </instructions>
```

# Maven Bundle Plugin Sample

```
<artifactId>org.apache.sling.engine</artifactId>
<packaging>bundle</packaging>
<version>2.0.7-SNAPSHOT</version>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <extensions>true</extensions>
      <configuration>
        <instructions>
          <Export-Package>
            org.apache.sling.engine;version=2.0
          </Export-Package>
          <Private-Package>
            org.apache.sling.engine.impl
          </Private-Package>
          <Embed-Dependency>
            commons-fileupload
          </Embed-Dependency>
        </instructions>
```

# Maven Bundle Plugin Sample

```
<artifactId>org.apache.sling.engine</artifactId>
<packaging>bundle</packaging>
<version>2.0.7-SNAPSHOT</version>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <extensions>true</extensions>
      <configuration>
        <instructions>
          <Export-Package>
            org.apache.sling.engine;version=2.0
          </Export-Package>
          <Private-Package>
            org.apache.sling.engine.impl
          </Private-Package>
          <Embed-Dependency>
            commons-fileupload
          </Embed-Dependency>
        </instructions>
```

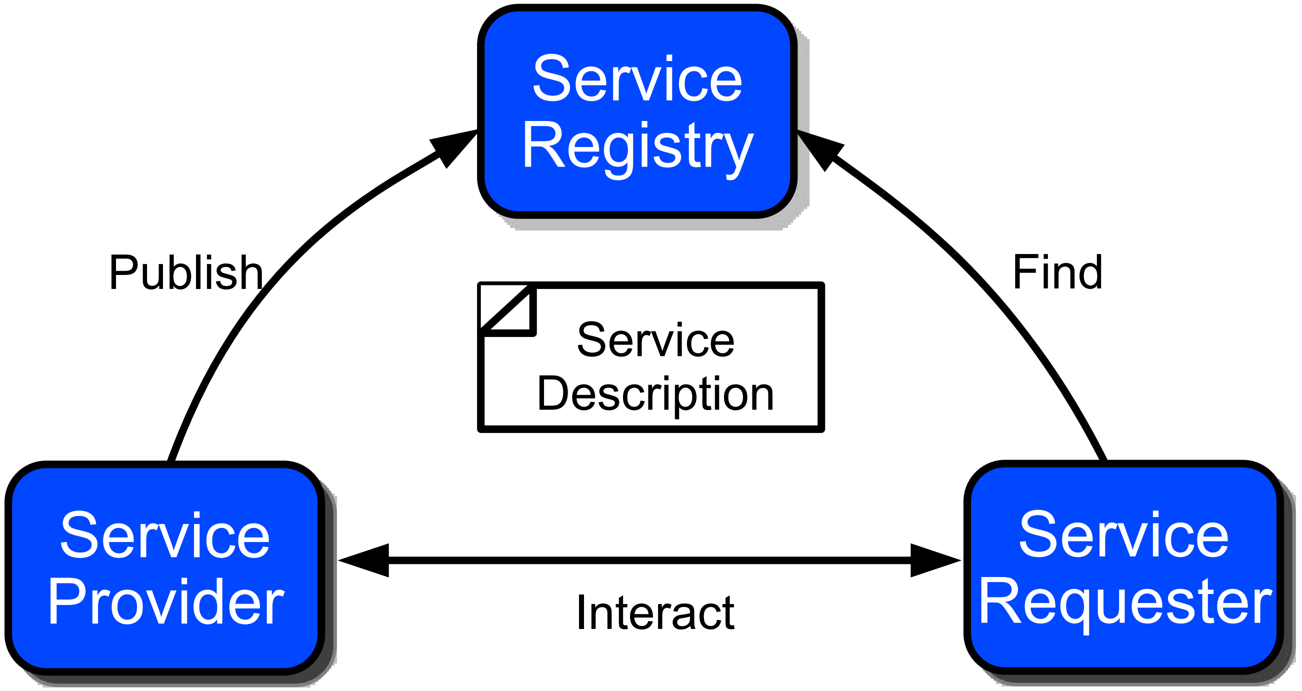
## Be Modular!

- Create clean package spaces
  - public vs private
- Provide Bundles
  - Add manifest information
- Think about dependencies
  - Additional bundle vs include
  - Optional
  - Version ranges
- Manage releases and versions properly
- Benefits even without OSGi

## 6 OSGi – Part 2 Services

## OSGi Services (1/3)

- Service-oriented architecture
  - Publish/find/bind
  - Possible to use modules without services



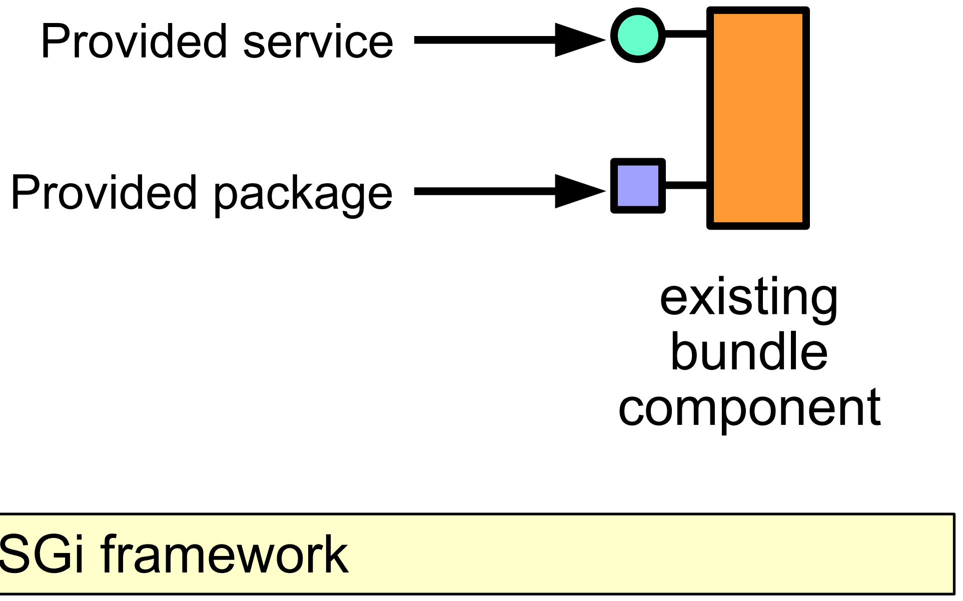
# OSGi Services (2/3)

- An OSGi application is...
  - 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

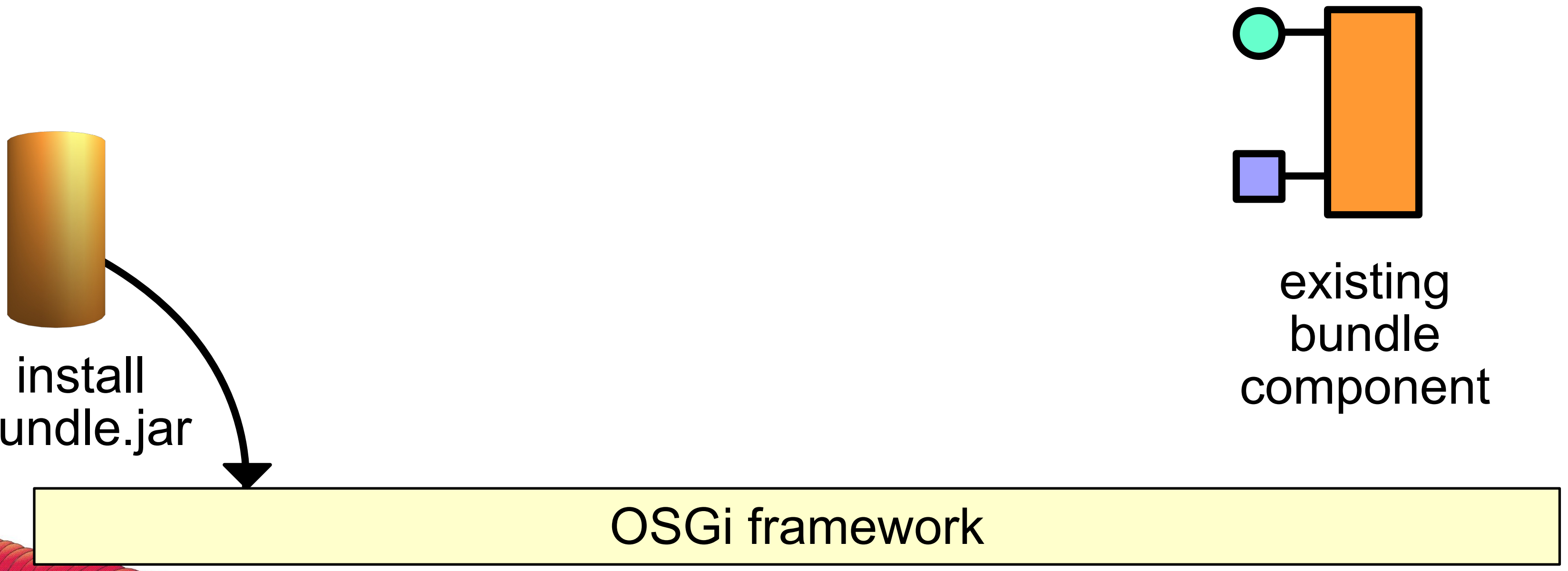
## OSGi Services (3/3)

- Dynamic service lookup



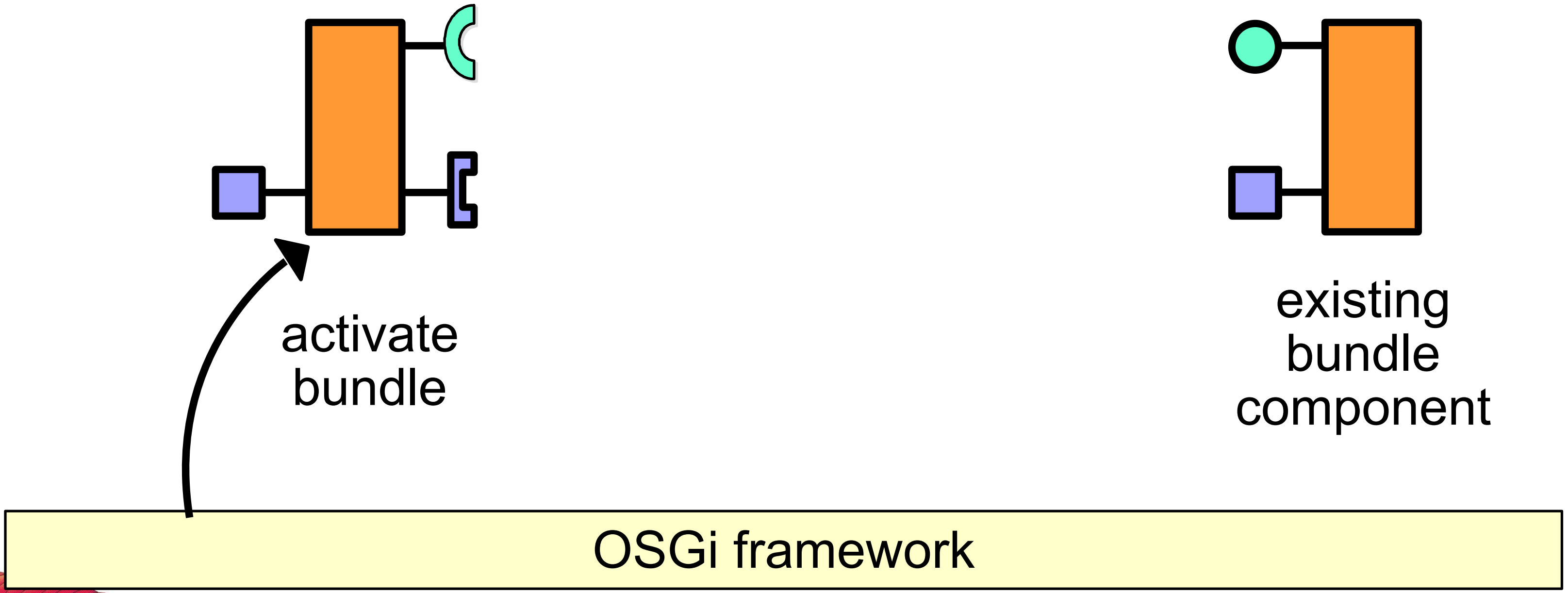
# OSGi Services (3/3)

- Dynamic service lookup



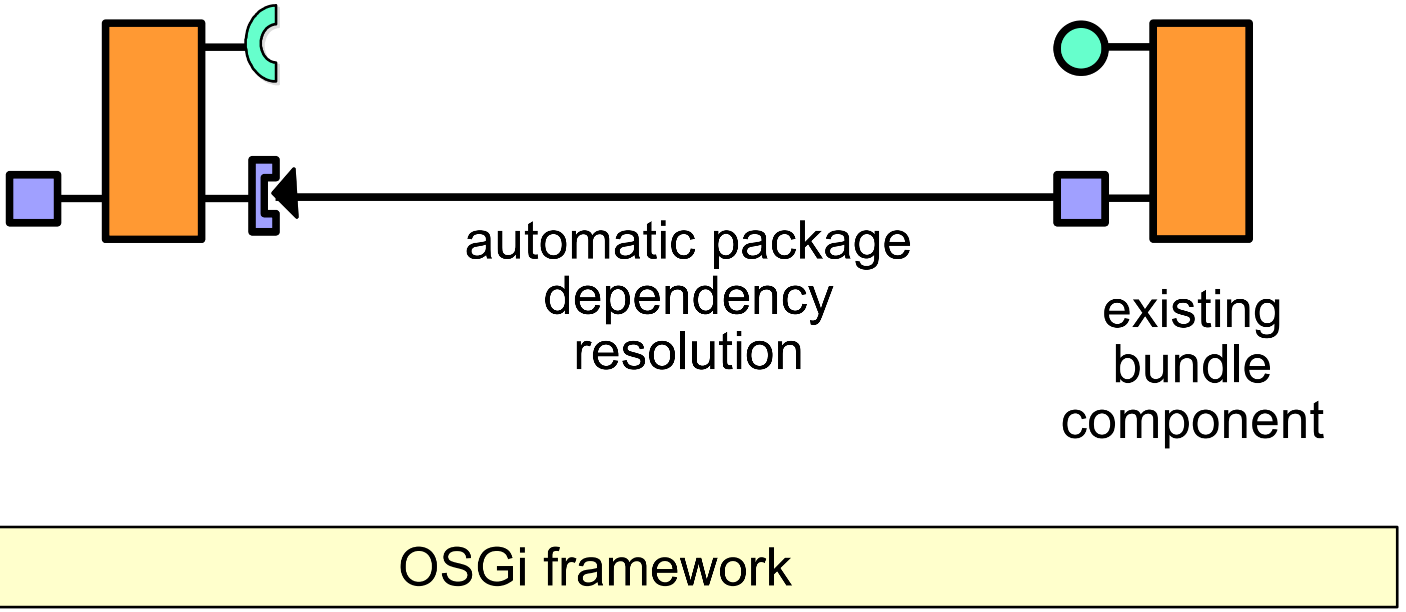
## OSGi Services (3/3)

- Dynamic service lookup



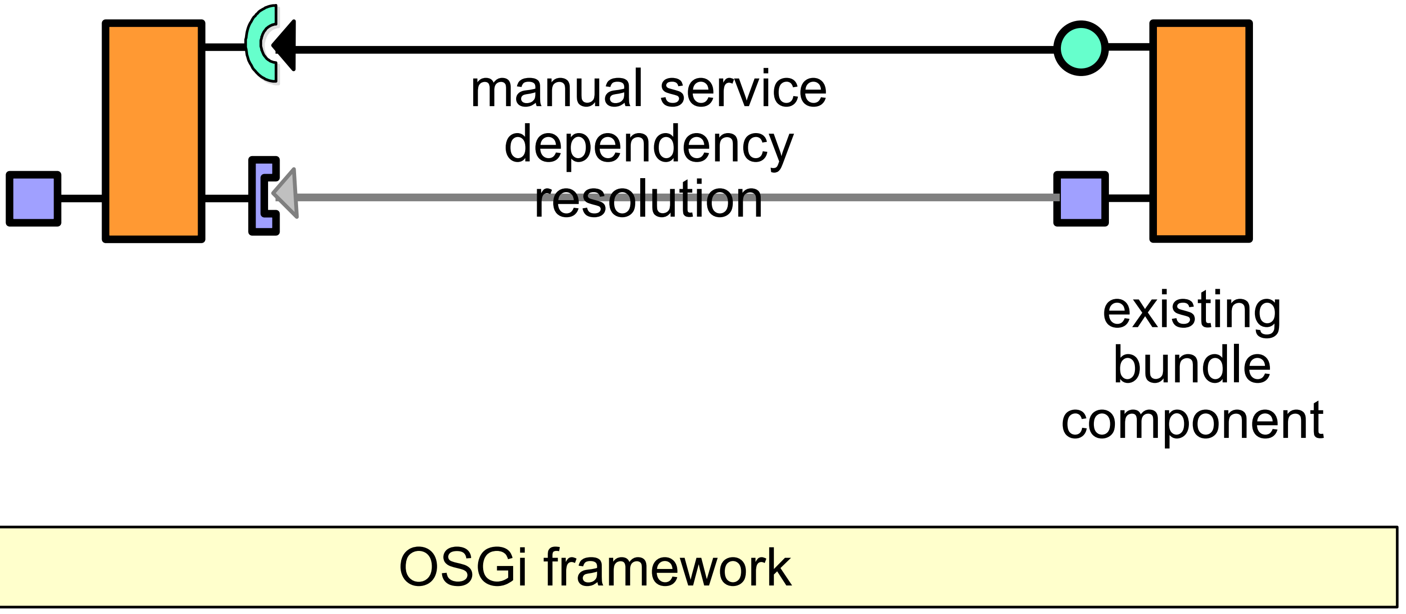
## OSGi Services (3/3)

- Dynamic service lookup


automatic package dependency resolution
existing bundle component
OSGi framework

## OSGi Services (3/3)

- Dynamic service lookup


manual service dependency resolution
existing bundle component
OSGi framework

## (OSGi) Services Advantages

- Lightweight services
  - Lookup is based on interface name
  - Direct method invocation
- Good design practice
  - Separates interface from implementation
  - Enables reuse, substitutability, loose coupling, and late binding

## OSGi Services Advantages

- Dynamic
  - Loose coupling and late binding
- Application's configuration is simply the set of deployed bundles
  - Deploy only the bundles that you need

## OSGi Service Registry

- Powerful but complicate to use directly
- Requires a different way of thinking
- Dynamic
  - Packages might come and go
  - Services might appear/disappear
- Manually resolve and track services
  - Doable, but complicated

## OSGi Service Registry

- More advanced solutions available
- Service Tracker
  - Still somewhat of a manual approach
- Declarative Services, Blueprint, *iPOJO*
  - Sophisticated service-oriented component frameworks
  - Automated dependency injection and more
  - More modern, POJO-oriented approaches
- Straight forward with Declarative Services, Annotations, Maven/Ant...

## Sample Service Development

- Developing a component
- Implementing a service (EventHandler)
- Uses another service (ThreadPool)
- Configurable (cleanup.period)
- Service properties (not user configurable)
- Only available if references are available

# Developing with Annotations

```
@Component
@Service(value=EventHandler.class)
@Properties({
    @Property(name="event.topics", value="*", propertyPrivate=true),
    @Property(name="event.filter", value="(event.distribute=*)",
              propertyPrivate=true)
})
public class DistributingEventHandler
    implements EventHandler {

    protected static final int DEFAULT_CLEANUP_PERIOD = 15;

    @Property(intValue=DEFAULT_CLEANUP_PERIOD)
    private static final String PROPERTY_CLEANUP_PERIOD ="cleanup.period";

    @Reference
    protected ThreadPool threadPool;

    @Activate
    protected void activate(final Map<String, Object> props) {
        this.cleanupPeriod = (Integer)props.get(PROP_CLEANUP_PERIOD);
    }
```

Leading the Wave
of Open Source

## Apache Felix SCR Tooling

- Maven Plugin or Ant Task
- Processes annotations
- Generates configuration files for
  - Declarative Services
  - Metatype
- Central configuration of services through ConfigAdmin
- Annotations or JavaDoc Tags (JDK 1.4)

## OSGi Declarative Services

- Component Framework Specification
  - XML Configuration
    - Contained in bundle
    - Manifest entry pointing to config(s)
  - Publishing services (through OSGi registry)
  - Consuming services
    - Policy (static,dynamic), cardinality (0..1, 1..1, 0..n)
  - Default configuration
  - Service lifecycle management
- Various Implementations
  - Apache Felix SCR

# Dynamic Services Configuration

```
<scr:component enabled="true"
         name="org.apache.sling.event.impl.DistributingEventHandler">

  <implementation
       class="org.apache.sling.event.impl.DistributingEventHandler"/>

  <service servicefactory="false">
    <provide interface="org.osgi.service.event.EventHandler"/>
  </service>

  <property name="repository.path" value="/var/eventing/distribution"/>
  <property name="cleanup.period" type="Integer" value="15"/>

  <reference name="threadPool"
             interface="org.apache.sling.event.ThreadPool"
             cardinality="1..1" policy="static"
             bind="bindThreadPool" unbind="unbindThreadPool"/>
```

## Dynamic Services Configuration

```
<scr:component enabled="true"
         name="org.apache.sling.event.impl.DistributingEventHandler">

  <implementation
       class="org.apache.sling.event.impl.DistributingEventHandler"/>

  <service servicefactory="false">
    <provide interface="org.osgi.service.event.EventHandler"/>
  </service>

  <property name="repository.path" value="/var/eventing/distribution"/>
  <property name="cleanup.period" type="Integer" value="15"/>

  <reference name="threadPool"
             interface="org.apache.sling.event.ThreadPool"
             cardinality="1..1" policy="static"
             bind="bindThreadPool" unbind="unbindThreadPool"/>
```

# Dynamic Services Configuration

```
<scr:component enabled="true"
         name="org.apache.sling.event.impl.DistributingEventHandler">

  <implementation
       class="org.apache.sling.event.impl.DistributingEventHandler"/>

  <service servicefactory="false">
    <provide interface="org.osgi.service.event.EventHandler"/>
  </service>

  <property name="event.topics" value="*"/>
  <property name="cleanup.period" type="Integer" value="15"/>

  <reference name="threadPool"
           interface="org.apache.sling.event.ThreadPool"
           cardinality="1..1" policy="static"
           bind="bindThreadPool" unbind="unbindThreadPool"/>
```

# Dynamic Services Configuration

```
<scr:component enabled="true"
         name="org.apache.sling.event.impl.DistributingEventHandler">

  <implementation
       class="org.apache.sling.event.impl.DistributingEventHandler"/>

  <service servicefactory="false">
    <provide interface="org.osgi.service.event.EventHandler"/>
  </service>

  <property name="repository.path" value="/var/eventing/distribution"/>
  <property name="cleanup.period" type="Integer" value="15"/>

  <reference name="threadPool"
             interface="org.apache.sling.event.ThreadPool"
             cardinality="1..1" policy="static"
             bind="bindThreadPool" unbind="unbindThreadPool"/>
```

## Declarative Services

- Reads XML configs on bundle start
- Registers services
- Keeps track of dependencies
  - Starts/stops services
- Invokes optional activation and deactivation method
  - Provides access to configuration
- Caution: A service is by default only started if someone else uses it!
  - Immediate flag forces a service start

## Config Admin

- OSGi Config Admin
  - "The" solution to handle configurations
  - Configuration Manager
  - Persistence storage
  - **API** to retrieve/update/remove configs
- Works with Declarative Services
  - Configuration changes are propagated to the components
- Various Implementations
  - Apache Felix (Reference Implementation)

## Metatype and Web Console

- OSGi Metatype Service
  - Description of bundle metadata
  - Description of service configurations
    - Property type, name, and description
  - Implementation in Apache Felix
- Apache Felix Web Console
  - Great solution to configure the system
  - Especially component configurations
  - Uses metatype description

## Apache Felix SCR Tooling

- Combines everything (DS, ConfigAdmin, Metatype, Maven/Ant)
- Annotation-based (even for 1.4)
- Annotate components
  - Properties with default values
  - Service providers
  - Services references (policy and cardinality)
- Generates DS XML
- Generates Metatype config
- Generates Java code

ApacheCon

## 7 OSGi – Part 3 Dynamics

## Event Admin

- OSGi Specification for event handling
- Publish/Subcribe
- Based on hierarchical topics
  – org/apache/felix/foo
- Map of properties

## Publishing Events

- Get the EventAdmin service
- Create event object
  - Topic and properties
- Send the event
  - Sync or async

## Receiving Events

- Implement the EventHandler interface
- Register service with properties
  - Interested topics
  - Optional filter

## Event Admin

- OSGi Specification for event handling
- Loose coupling of services
- “Predefined events”
  - Framework events (bundles)
  - Service changes
  - Configuration changes
- Apache Felix EventAdmin implementation

## Everything is a Bundle

- How to structure bundles?
  - API vs implementation bundle
  - Fine-grained vs coarse-grained
  - No “One Size Fits All”
- Simple Rules
  - Stable code vs changing code
  - Optional parts

## Third Party Libraries

- Use them as bundles
  - Project delivers already a bundle
    - Apache Commons, Apache Sling etc.
  - Use special bundle repositories
    - Felix Commons, Spring etc.
    - But check included metadata!
  - Create your own wrapper
    - Easy with the Felix maven bundle plugin
- Include classes in your bundle
  - Again: easy with the Felix maven bundle plugin

## Everything is Dynamic

- Bundles can come and go!
  - Packages
  - Services
- Services can come and go!
- Be prepaired!
  - Application code must handle dynamics!

## Loose Coupling of Services

- Easy through OSGi service registry
  – Contract defined through interface
- Minimal OSGi knowledge required with available solutions
  – Apache Felix SCR Tooling
- Alternatives
  – Apache Felix iPojo
  – Blueprint Container Specification
    - Apache Aries (Incubator)

## 8 Famous Final Words

# Suggestions for Development

- Think about modularity!
  - Clean package space
- Think about dynamics!
- Consider OSGi
- Check out the spec and other projects
  - Attend today's OSGi/Felix track
- Minimize dependencies to OSGi
  - but only if it makes sense

## Suggestions for Using OSGi

- Think about dynamics
  - Optional bundles
  - Optional services
  - Handle these cases
- Use your preferred logging library
  - LogManager takes care
- Use available tooling
- Be part of the community!

## Check It Out

- Read the OSGi spec
  - Framework
  - Config Admin, Metatype, Declarative Services
  - Event Admin, OBR
- Download Apache Felix
  - Try tutorials and samples
  - Quickstart with Apache Karaf
- Download Apache Sling :)
- Explore the web – **embrace OSGi**

# Embrace ~~Change~~ OSGi

Questions?