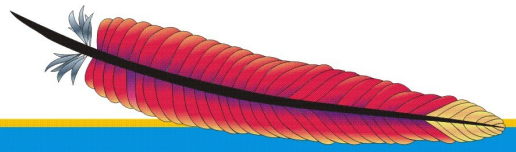
ApacheCon

Apache Felix –
A Standard Plugin Model for Apache

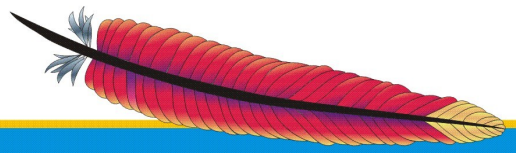
Richard S. Hall

Atlanta, Georgia U.S.A.
November 13th, 2007



Agenda

① Why OSGi technology?
② OSGi technology overview
③ Apache Felix status
④ Example application
⑤ OSGi application approaches
⑥ Example application demo
⑦ Advanced approaches
⑧ Conclusion

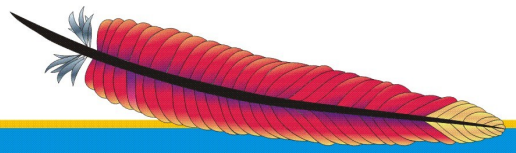


ApacheCon

① Why OSGi Technology?

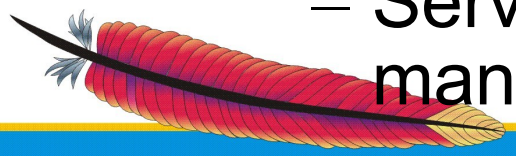
(Addressing Java's Limitations)

② ③ ④ ⑤ ⑥ ⑦ ⑧



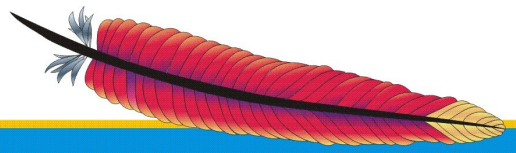
Motivation (1/2)

- Growing complexity requires not only highly modular code, but also systems that are dynamically extensible
- This is true no matter which problem domain is your area of concern
  - Embedded systems need to adapt to changing requirements even though they are deployed out in the field
  - Client applications must respond to user desires for new functionality instantaneously
  - Server applications must be configurable and manageable without down time



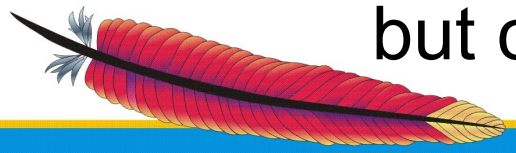
Motivation (2/2)

- Java provides the mechanisms to do these things, but they are
  - Low level
  - Error prone
  - Ad hoc
- Java's shortcoming are particular evident in its support for both modularity and dynamism



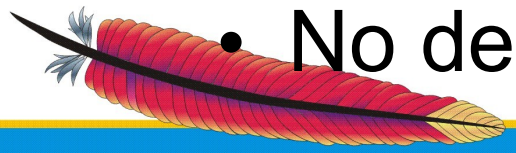
Java Modularity Limitations (1/2)

- Limited scoping mechanisms
  - No module access modifier
- Simplistic version handling
  - Class path is first version found
  - JAR files assume backwards compatibility at best
- Implicit dependencies
  - Dependencies are implicit in class path ordering
  - JAR files add improvements for extensions, but cannot control visibility



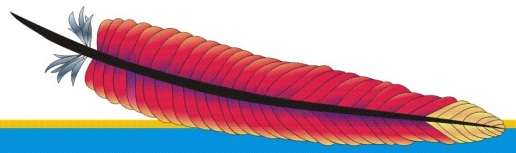
Java Modularity Limitations (2/2)

- Split packages by default
  - Class path approach searches until it finds, which leads to shadowing or version mixing
  - JAR files can provide sealing
- Unsophisticated consistency model
  - Cuts across previous issues, it is difficult to ensure class space consistency
- Missing module concept
  - Classes are too fine grained, packages are too simplistic, class loaders are too low level
- No deployment support



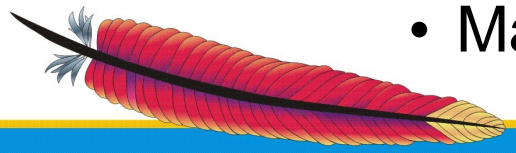
Java Dynamism Limitations

- Low-level support for dynamics
  - Class loaders are complicated to use and error prone
- Support for dynamics is still purely manual
  - Must be completely managed by the programmer
  - Leads to many ad hoc, incompatible solutions
- No deployment support

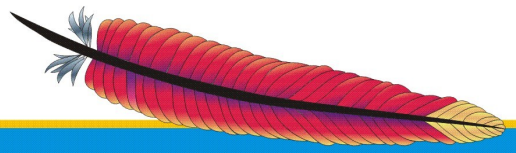


OSGi Technology

- Resolves many deficiencies associated with standard Java support for modularity and dynamism
  - Defines a module concept
    - Explicit sharing of code (i.e., importing and exporting)
  - Automatic management of code dependencies
    - Enforces sophisticated consistency rules for class loading
  - Code life cycle management
    - Manages dynamic deployment and configuration



ApacheCon



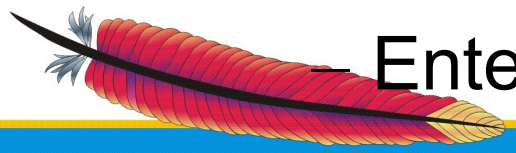
Leading the Wave
of Open Source

② OSGi Technology Overview

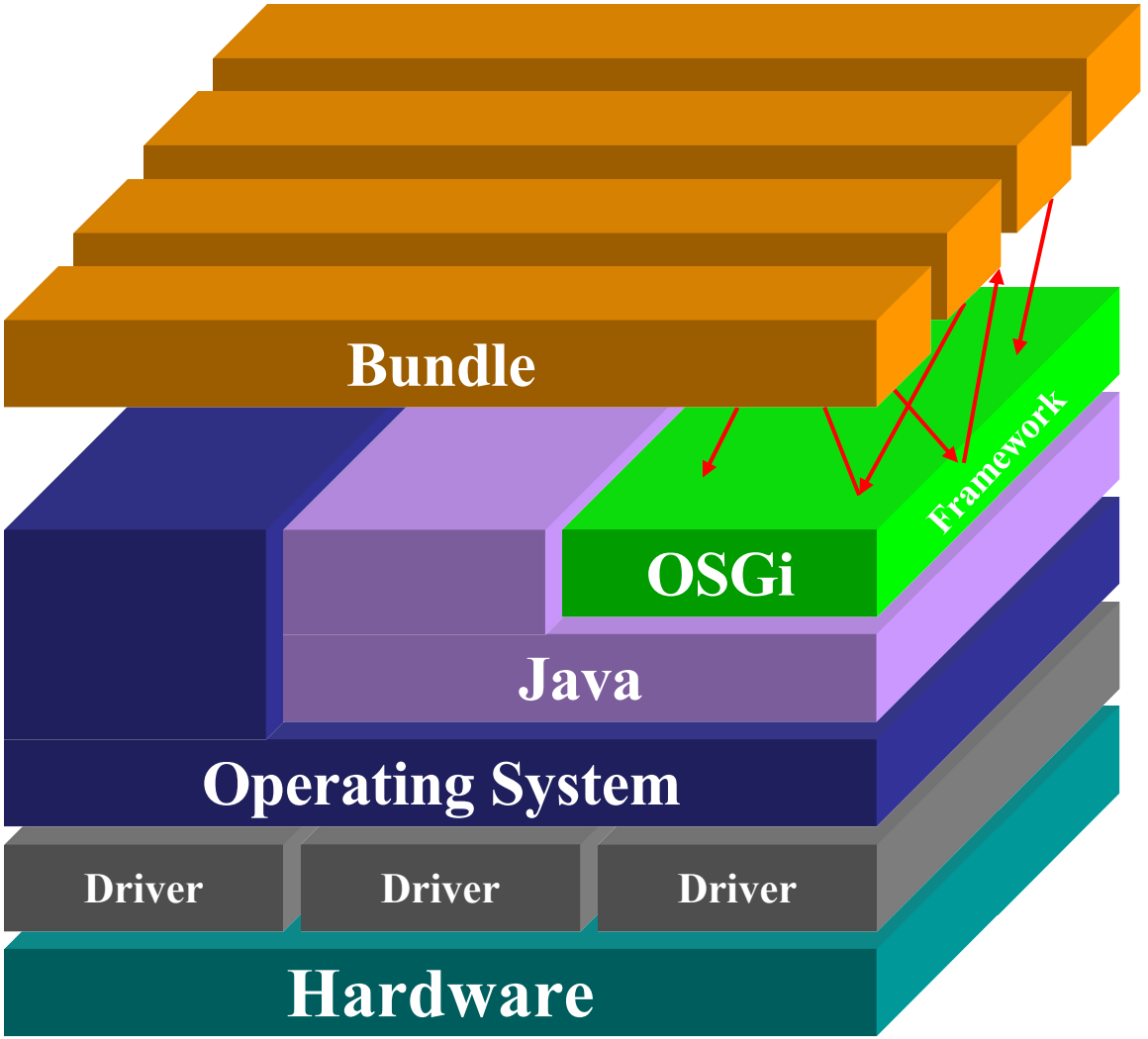
③④⑤⑥⑦⑧

OSGi Alliance

- Industry consortium
- Defines OSGi Service Platform
  - 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)

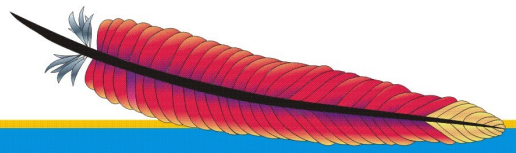


OSGi Architectural Overview



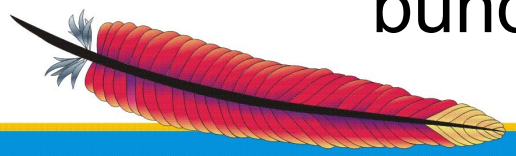
OSGi Framework (1/2)

- Component-oriented framework
  - *Bundles* (i.e., modules/components)
  - Package sharing and version management
  - Life-cycle management and notification
- Service-oriented architecture
  - Publish/find/bind intra-VM service model
- Open remote management architecture
  - No prescribed policy or protocol

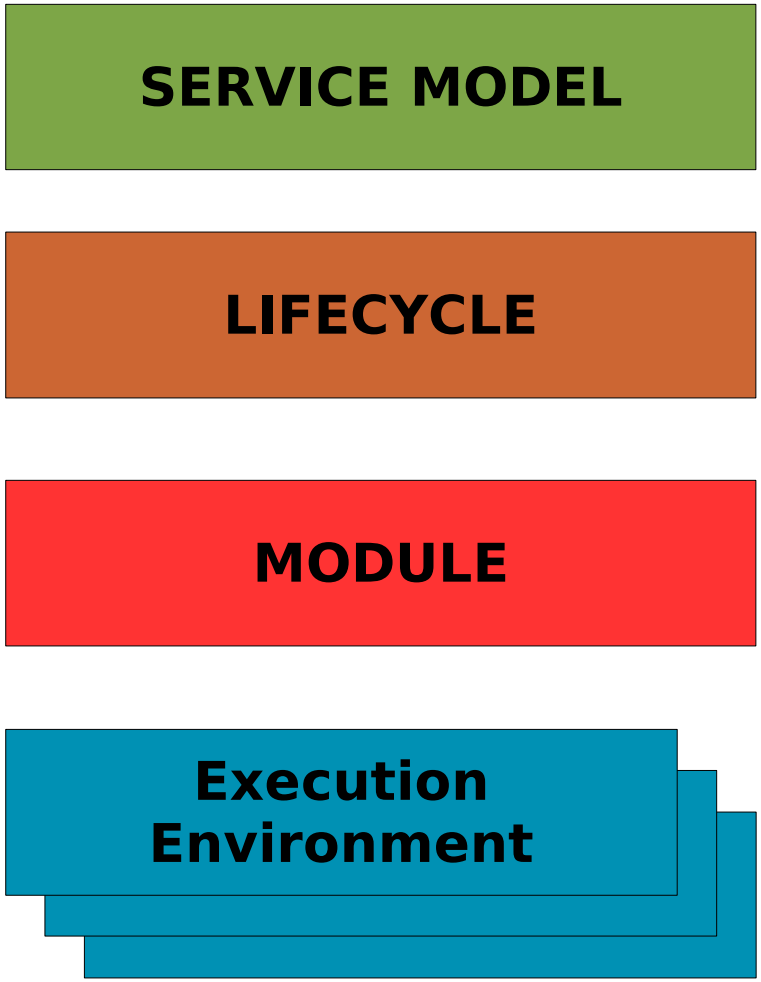


OSGi Framework (2/2)

- 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, update, and uninstall of bundles



OSGi Framework Layering



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

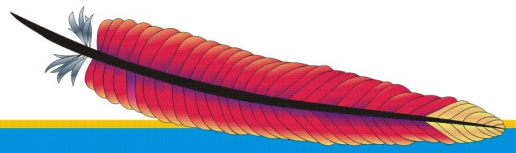
L2 - Manages the lifecycle of bundle in a bundle repository without requiring the VM be restarted

L1 - Creates the concept of modules (aka. bundles) that use classes from each other in a controlled way according to system and bundle constraints

L0 -
- OSGi Minimum Execution Environment
- CDC/Foundation
- JavaSE

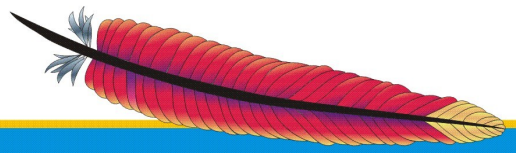
OSGi Modularity (1/4)

- Multi-version support
  - i.e., side-by-side versions
- Explicit code boundaries and dependencies
  - i.e., package imports and exports
- Support for various sharing policies
  - i.e., arbitrary version range support



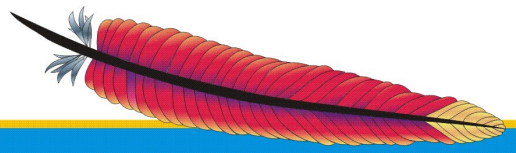
OSGi Modularity (2/4)

- Arbitrary export/import attributes for more control
  - Influence package selection
- Sophisticated class space consistency model
  - Ensures code constraints are not violated
- Package filtering for fine-grained class visibility
  - Exporters may include/exclude specific classes from exported package



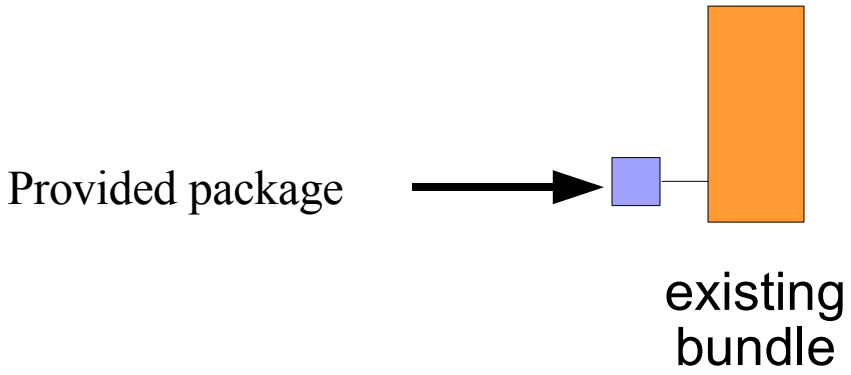
OSGi Modularity (3/4)

- Bundle fragments
  - A single logical module in multiple physical bundles
- Bundle dependencies
  - Allows for tight coupling when required



OSGi Modularity (4/4)

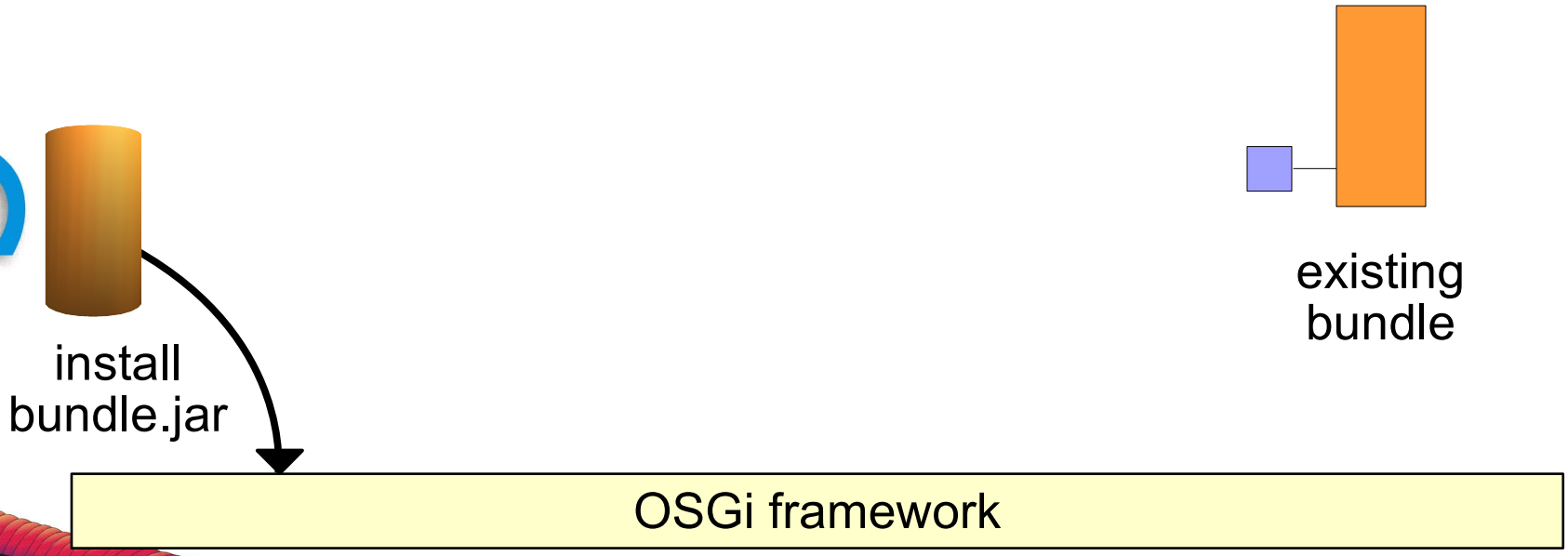
- Dynamic module deployment and dependency resolution



OSGi framework

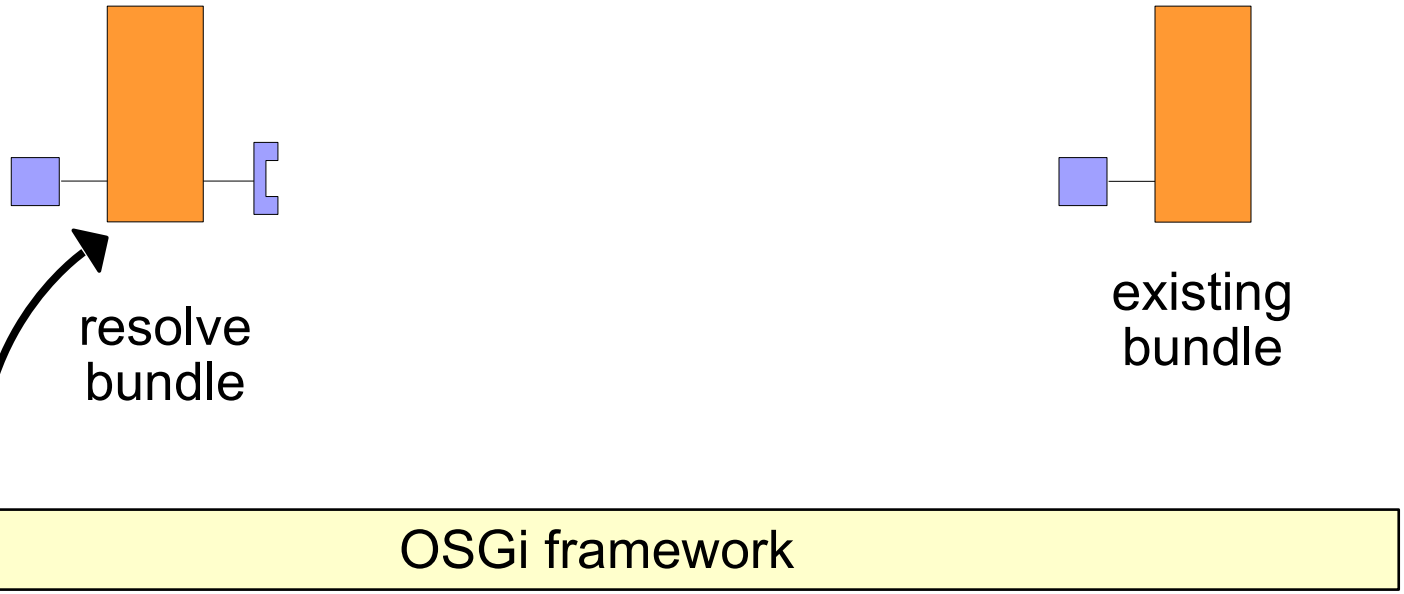
OSGi Modularity (4/4)

- Dynamic module deployment and dependency resolution



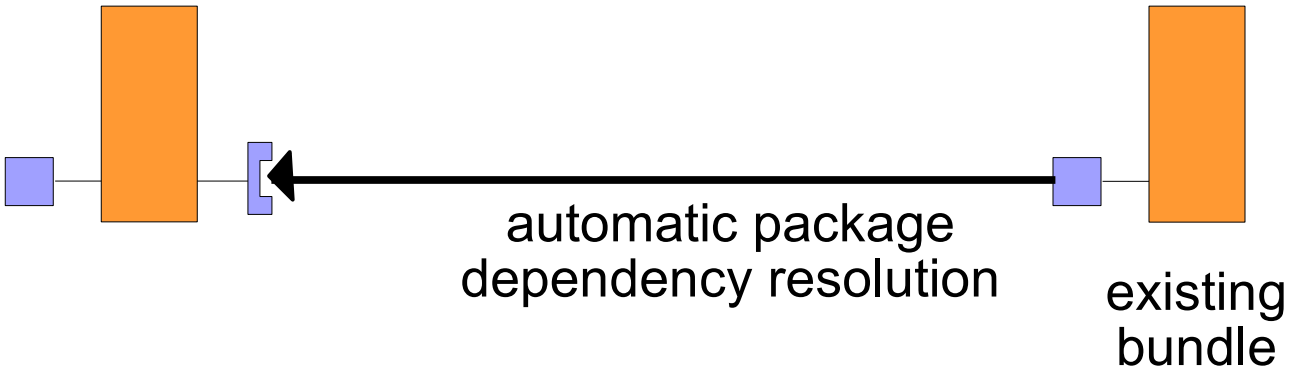
OSGi Modularity (4/4)

- Dynamic module deployment and dependency resolution



OSGi Modularity (4/4)

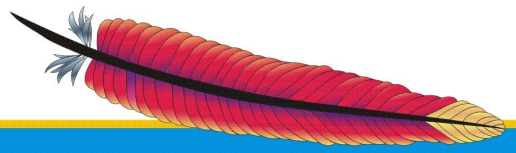
- Dynamic module deployment and dependency resolution



OSGi framework

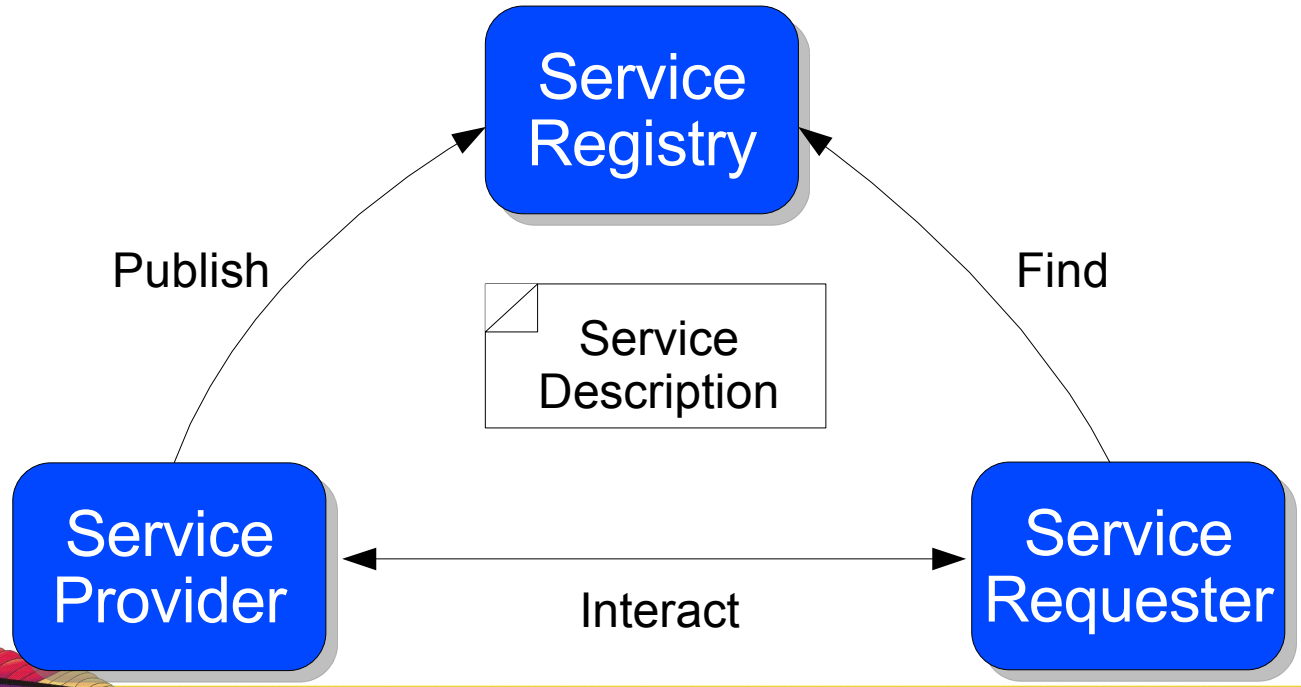
Leveraging OSGi Modularity

- Text editor + `jar`
  - Just add metadata to your JAR file's manifest
- Eclipse
  - Plug-in Development Environment (**PDE**) directly supports bundles
- Bundle packaging tools
  - **BND** from Peter Kriens
  - Apache Felix ***maven-bundle-plugin*** based on BND



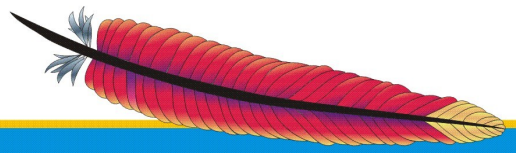
OSGi Services (1/3)

- OSGi framework promotes service-oriented interaction pattern among bundles
  - Possible to use modules without services



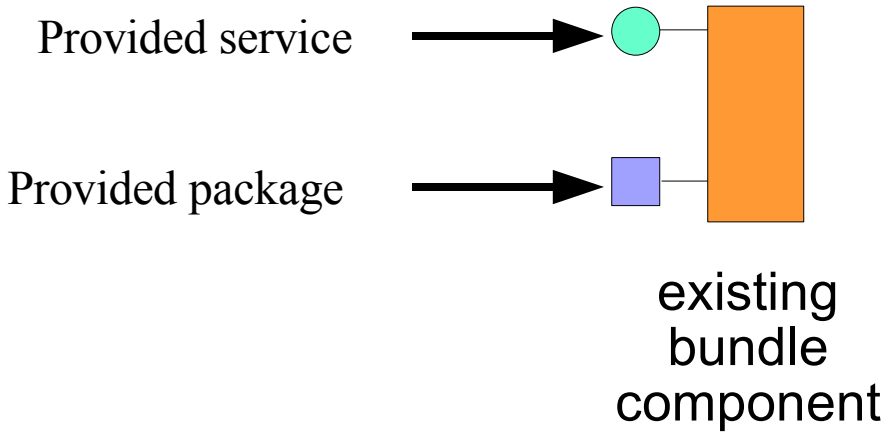
OSGi Services (2/3)

- 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)

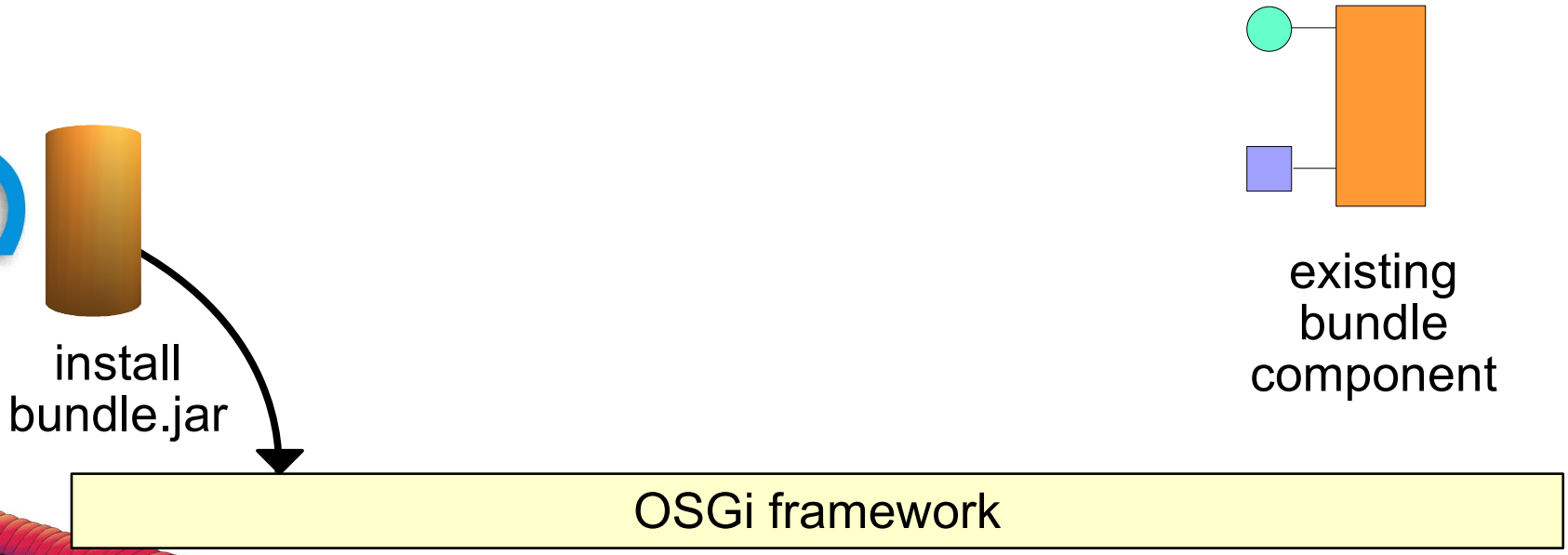
- Dynamic service lookup



OSGi framework

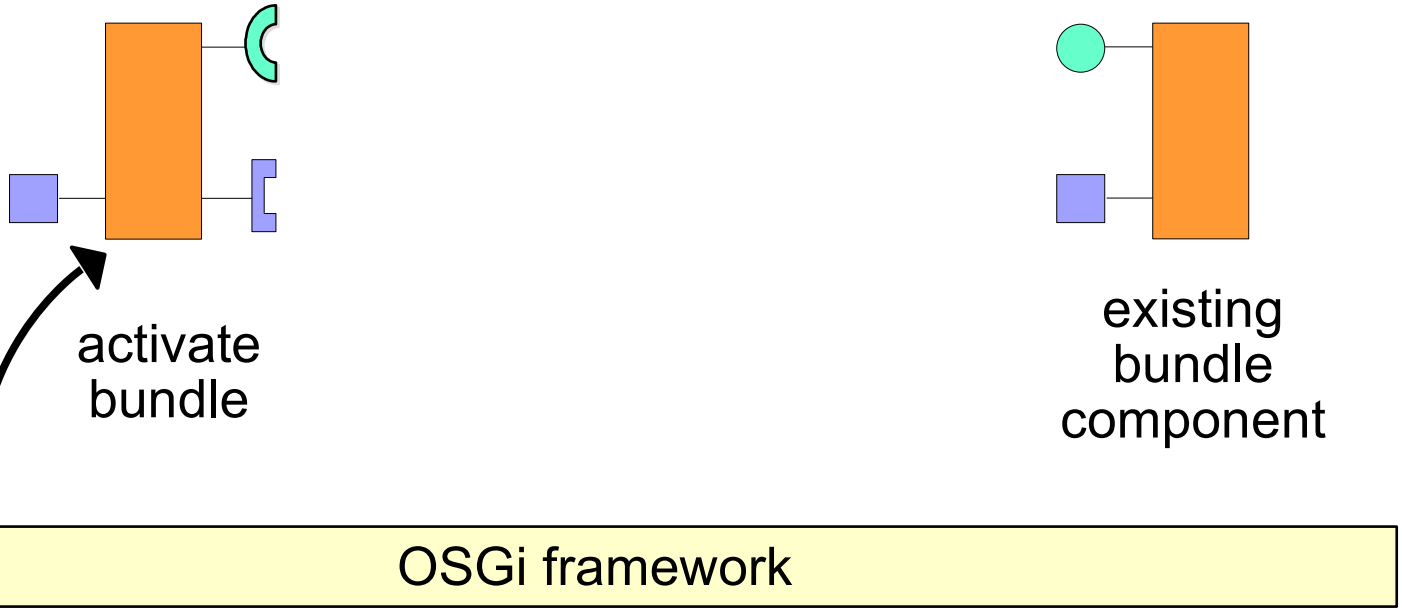
OSGi Services (3/3)

- Dynamic service lookup



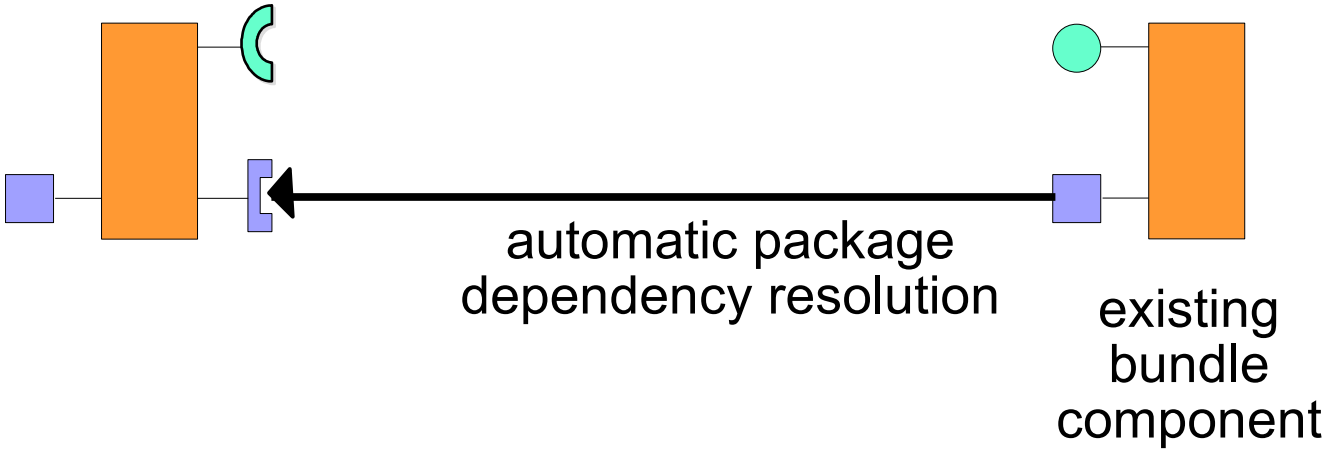
OSGi Services (3/3)

- Dynamic service lookup



OSGi Services (3/3)

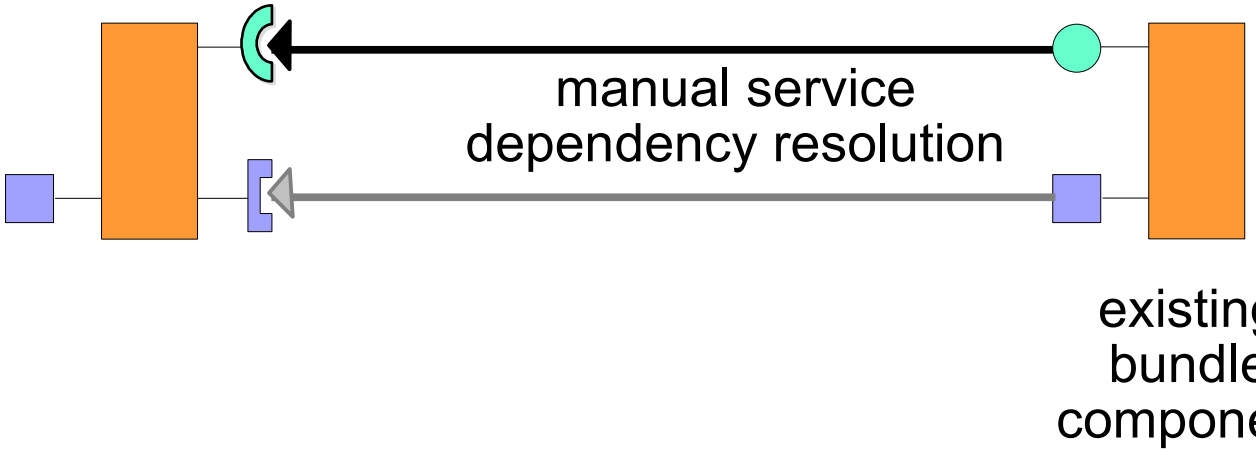
- Dynamic service lookup



OSGi framework

OSGi Services (3/3)

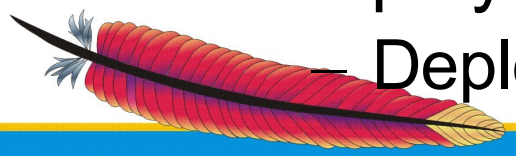
- Dynamic service lookup



OSGi framework

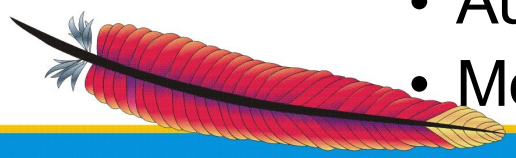
OSGi Services Advantages

- Lightweight services
  - Direct method invocation
- Good design
  - Separates interface from implementation
  - Enables reuse, substitutability, loose coupling, and late binding
- Dynamics
  - Loose coupling and late binding make it possible to support run-time dynamism
- Application's configuration is simply the set of deployed bundles
  - Deploy only the bundles that you need



OSGi Services Issues

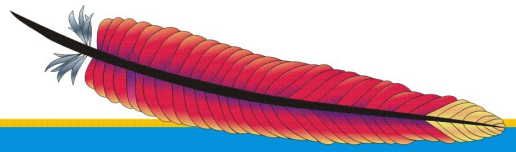
- Complicated
  - Requires a different way of thinking
    - Things might appear/disappear at any moment
  - Must manually resolve and track services
- There is help
  - Service Tracker
    - Still somewhat of a manual approach
  - Declarative Services, Spring-OSGi, iPOJO
    - Sophisticated service-oriented component frameworks
    - Automated dependency injection and more
    - More modern, POJO-oriented approaches



③

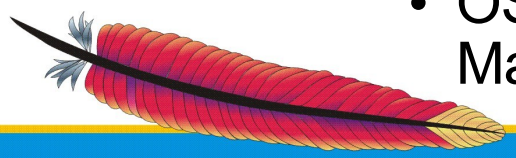
Apache Felix Status

④ ⑤ ⑥ ⑦ ⑧



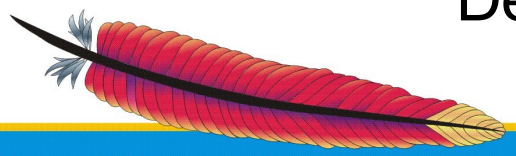
Apache Felix (1/4)

- Top-level project (April 2007)
- Apache licensed open source implementation of OSGi R4
  - Framework (in progress, stable and functional)
    - Version 1.0.1 currently available
  - Services (in progress, stable and functional)
    - Package Admin, Start Level, URL Handlers, Declarative Services, UPnP Device, HTTP Service, Configuration Admin, Preferences, User Admin, Wire Admin, Event Admin, Meta Type, and Log
    - OSGi Bundle Repository (OBR), Dependency Manager, Service Binder, Shell, **iPOJO**, Mangen



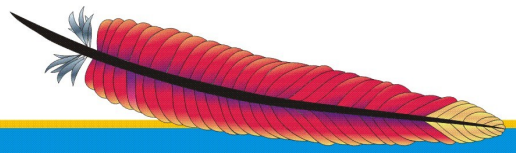
Apache Felix (2/4)

- Felix community is growing strong
  - 20 committers
  - Code granted and contributed from several organizations and communities
    - Grenoble University, ObjectWeb, CNR-ISTI, Ascert, Luminis, Apache Directory, INSA, DIT UPM, Day Management AG
    - Several community member contributions
  - Apache projects interested in Felix and/or OSGi
    - Directory, Cocoon, JAMES, Jackrabbit, Harmony, Derby



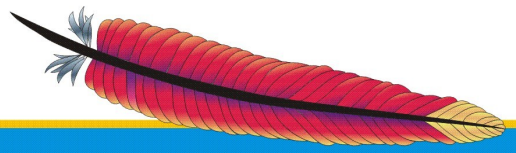
Apache Felix (3/4)

- Felix bundle developer support
  - Apache Maven2 bundle plugin
    - Merges OSGi bundle manifest with Maven2 POM file
    - Automatically generates metadata, such as Bundle-ClassPath, Import-Package, and Export-Package
      - Greatly simplifies bundle development by eliminating error-prone manual header creation process
    - Automatically creates final bundle JAR file
      - Also supports embed required packages, instead of importing them

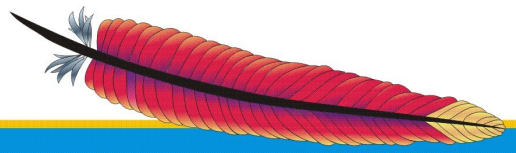


Apache Felix (4/4)

- Felix Commons
  - Effort to bundle-ize common open source libraries
    - Recently started
  - Includes 13 bundles, such as antlr, cglib, commons-collections, etc.
  - All community donated wrappers
- Roadmap
  - Continue toward R4 and R4.1 compliance
    - Largely only missing support for fragments



ApacheCon



④ Example Application

⑤ ⑥ ⑦ ⑧

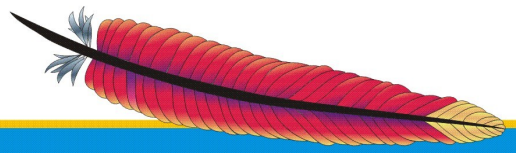
Simple Paint Program

- Defines a `SimpleShape` interface to draw shapes
  - Different implementations of `SimpleShape` can be created to draw different shapes
  - Each shape has name and icon properties
  - Available shapes are displayed in tool bar
- To draw a shape, click on its button and then click in the drawing canvas
  - Shapes cannot be dragged, but not resized
- Shape implementations can be dynamically installed/removed

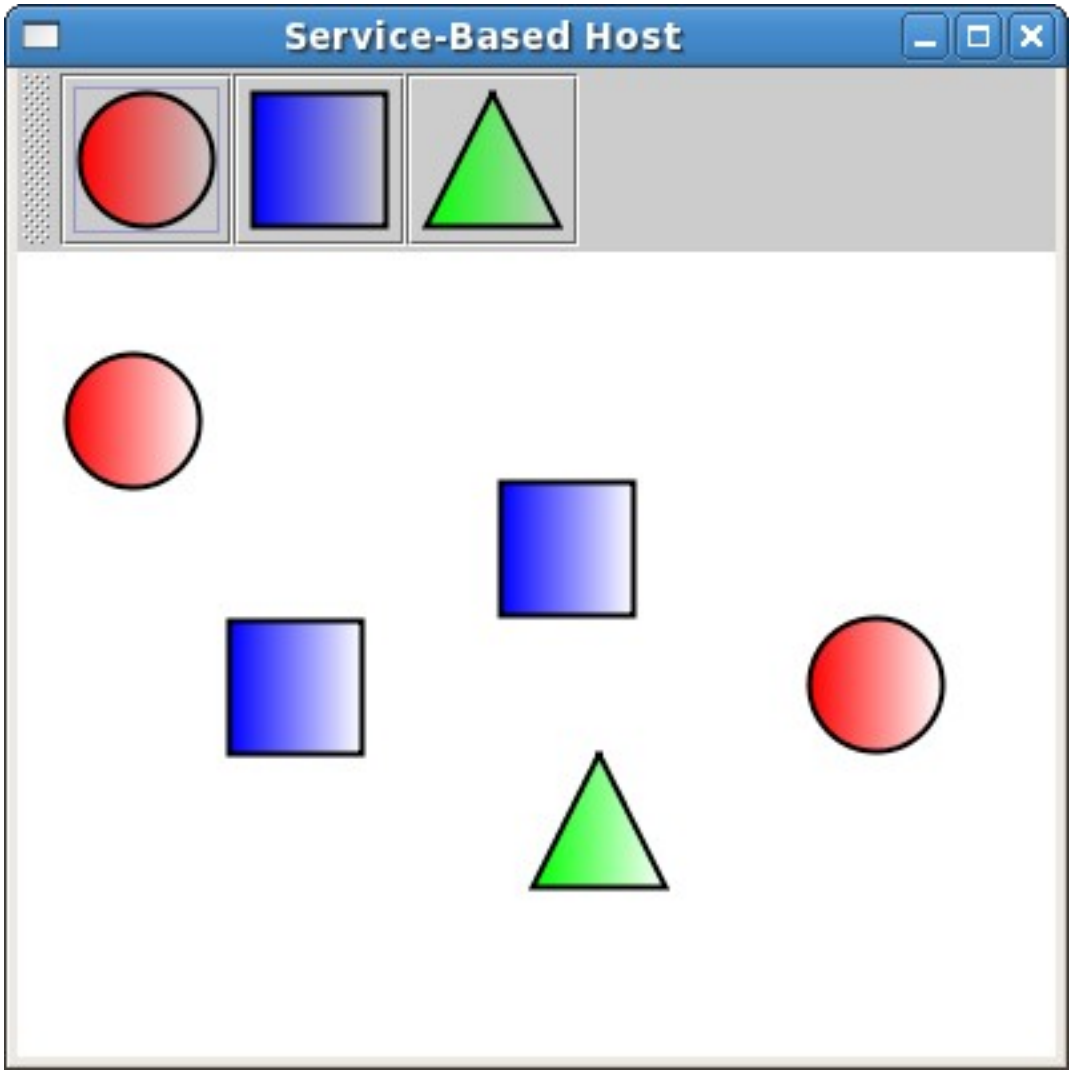
Shape Abstraction

- Conceptual `SimpleShape` interface

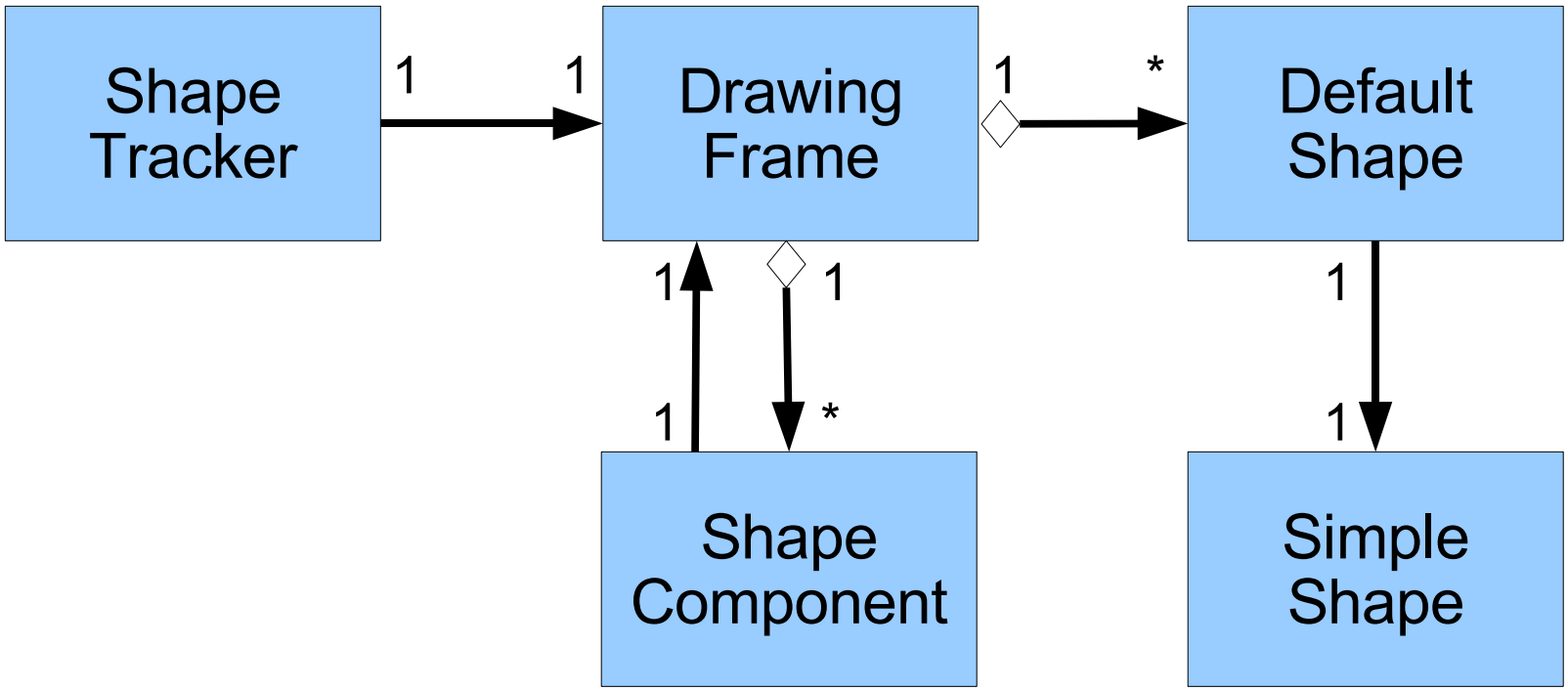
```
public interface SimpleShape
{
    /**
     * Method to draw the shape of the service.
     * @param g2 The graphics object used for
     *           painting.
     * @param p The position to paint the triangle.
    **/
    public void draw(Graphics2D g2, Point p);
}
```



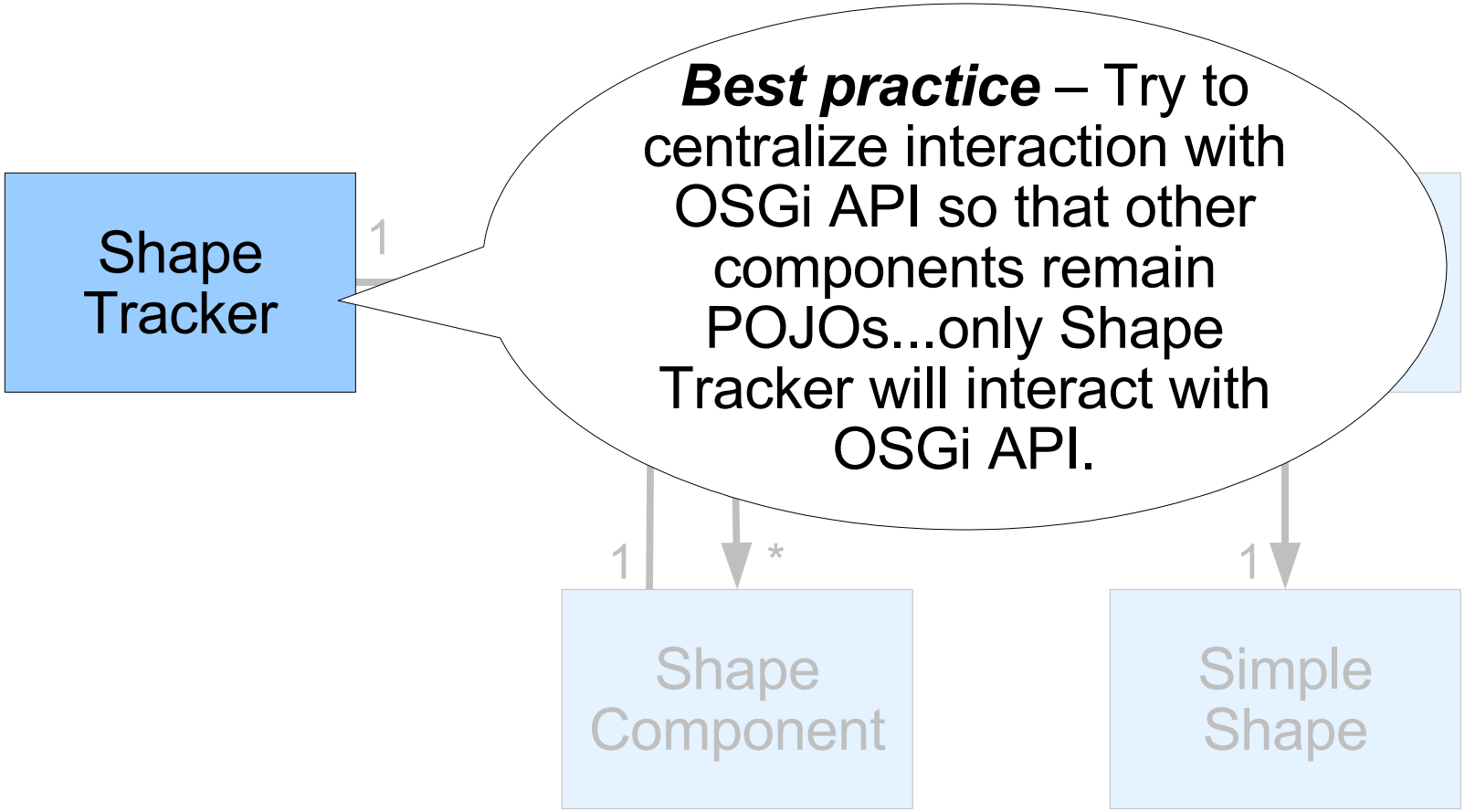
Paint Program Realization



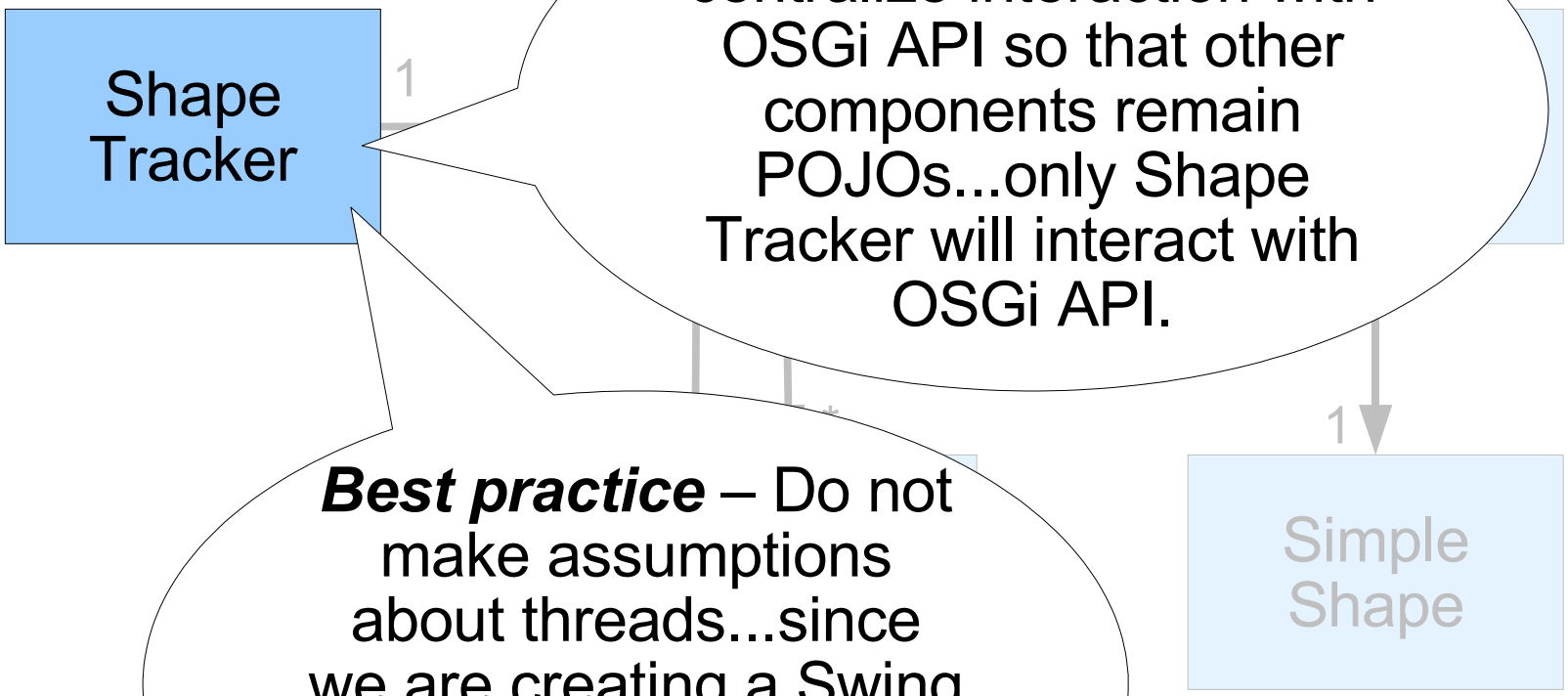
High-Level Architecture



High-Level Architecture



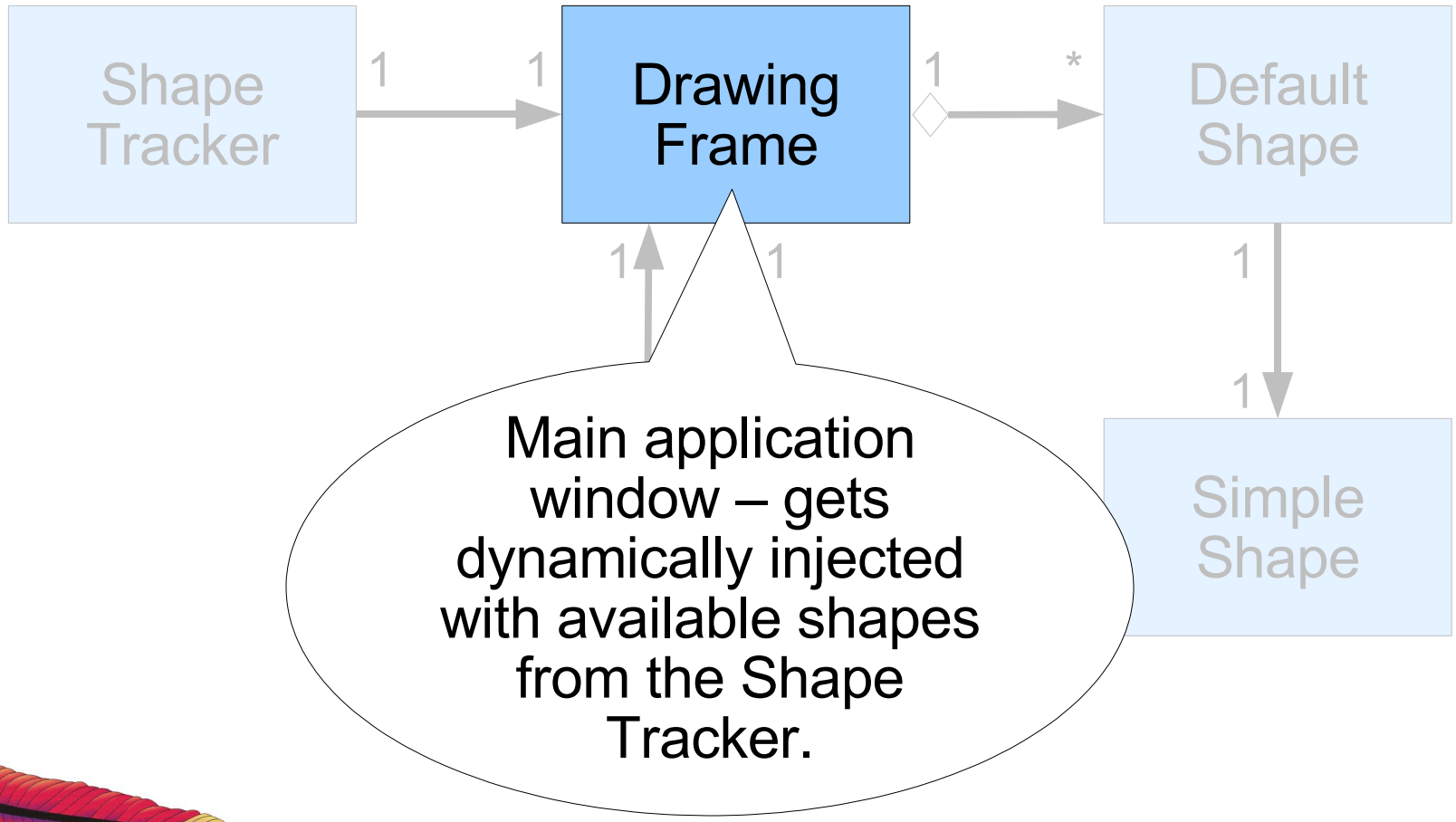
High-Level Architecture



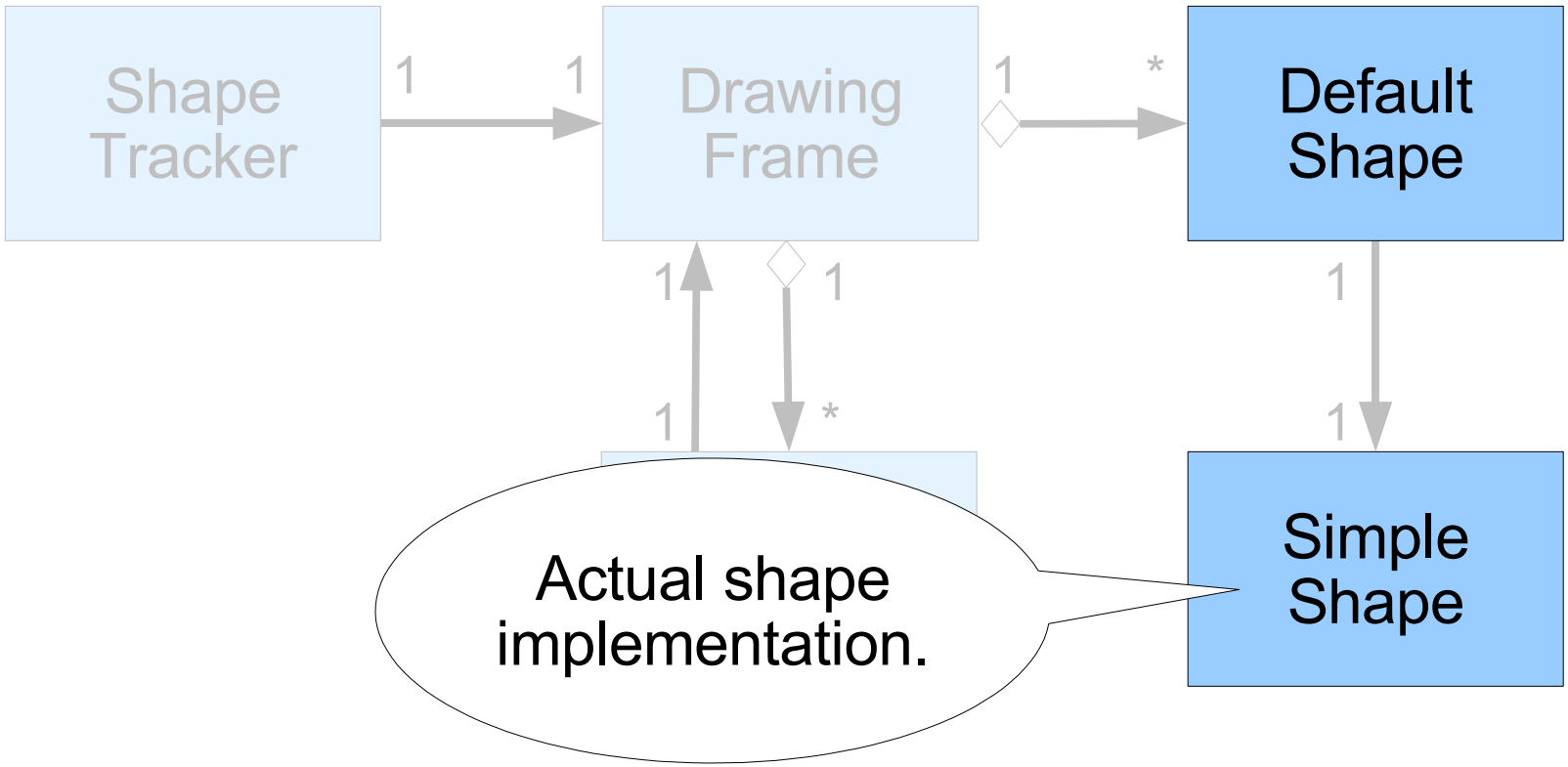
Best practice – Try to centralize interaction with OSGi API so that other components remain POJOs...only Shape Tracker will interact with OSGi API.

Best practice – Do not make assumptions about threads...since we are creating a Swing application, Shape Tracker sends events on Swing thread.

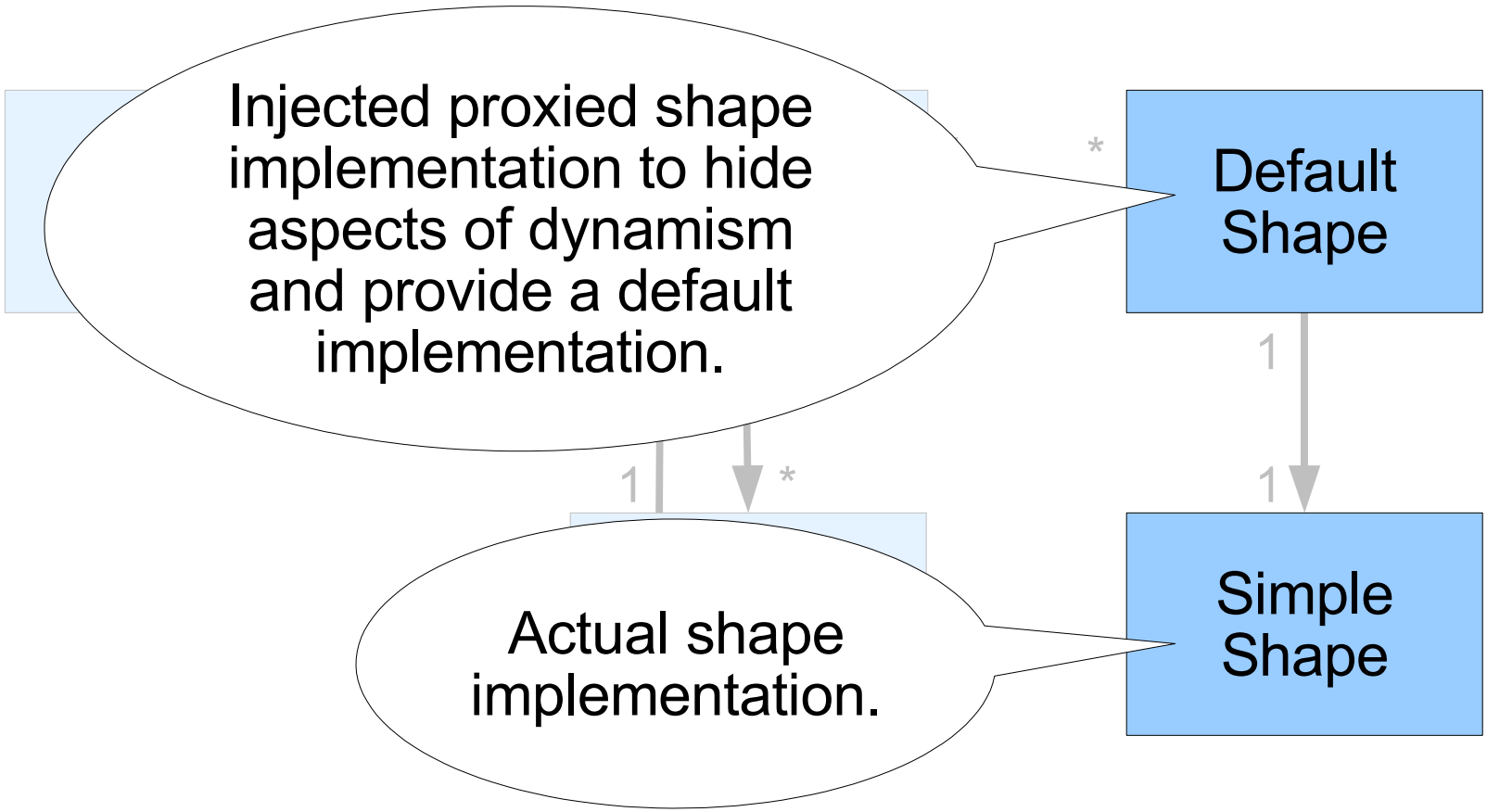
High-Level Architecture

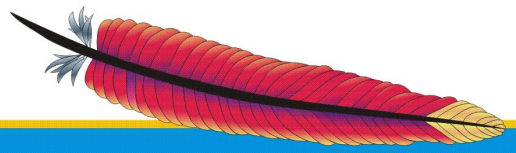


High-Level Architecture

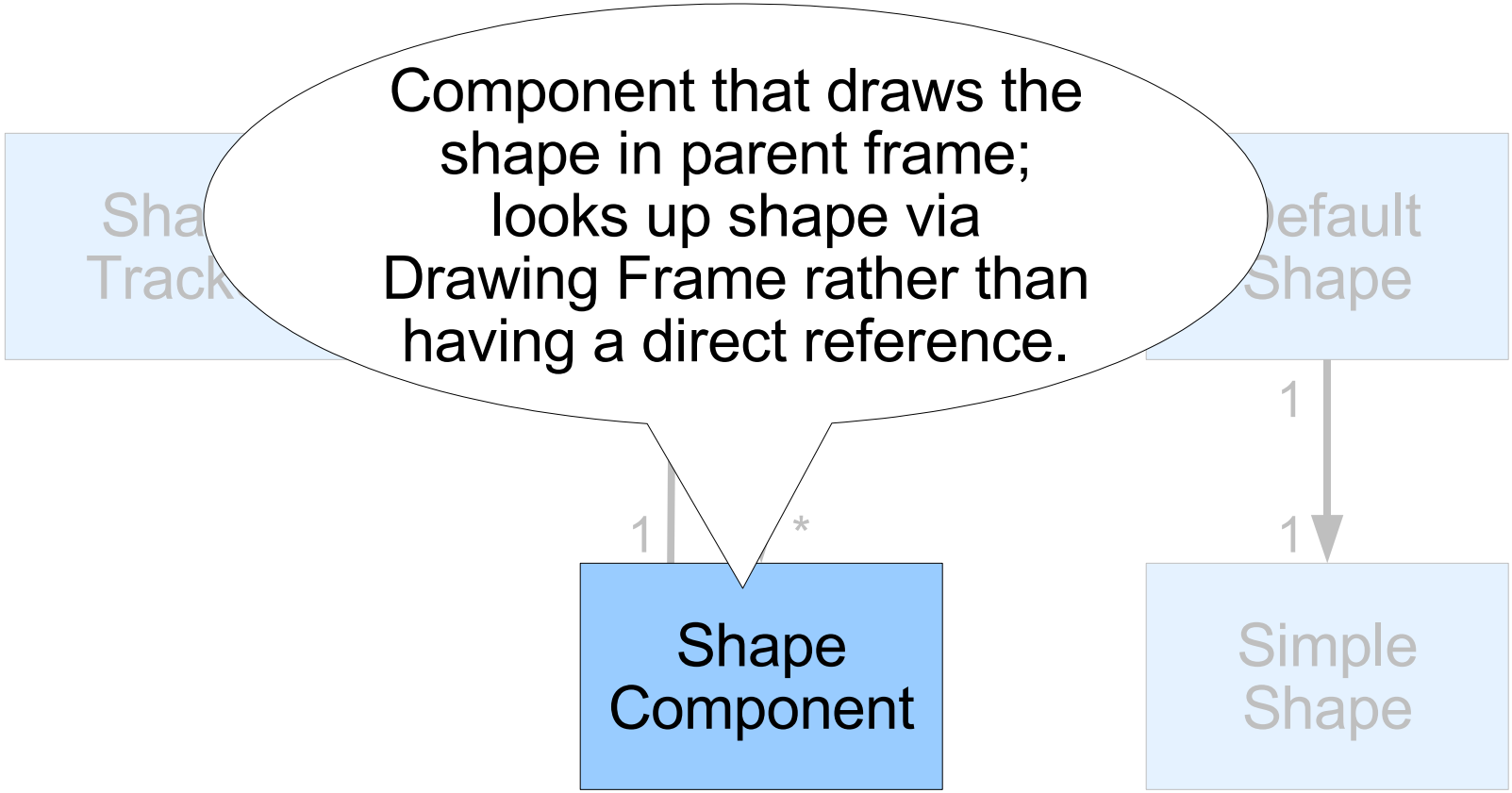


High-Level Architecture



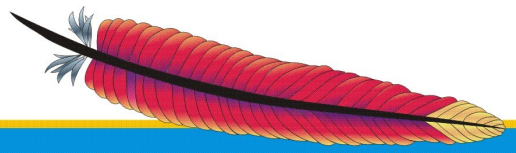


High-Level Architecture

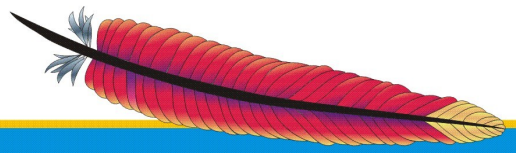


Implementing the Design

- The design is reasonably complete, but what is the precise approach we use for implementation?
  - It depends...
  - There are a few approach options when building OSGi-based applications...



ApacheCon



Leading the Wave
of Open Source

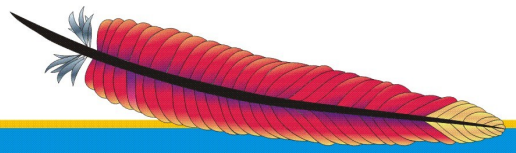
⑤

OSGi Application Approaches

⑥ ⑦ ⑧

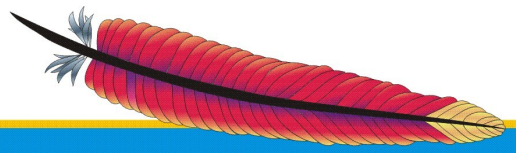
OSGi Application Approaches

- When creating an OSGi-based application there are two main orthogonal issues to consider
  - Service model vs. extender model
  - Bundled application vs. hosted framework
- The first issue is related to choosing the actual OSGi extensibility mechanism
- The second issue is an advanced topic to be discussed later, but is related to who is in control of whom



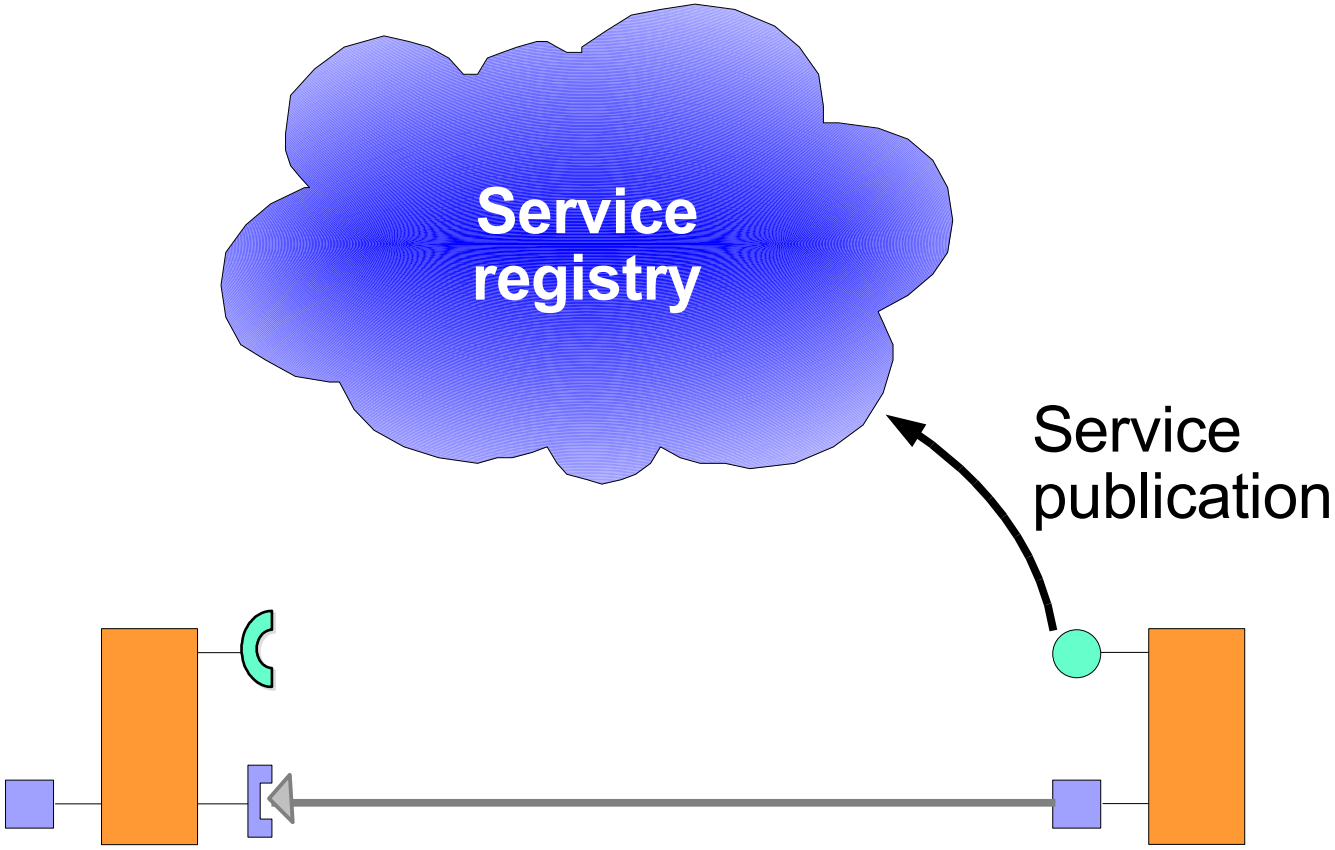
Service vs. Extender Models

- Two different approaches for adding extensibility to an OSGi-based application
  - The service-based approach uses the OSGi service concept and the service registry as the extensibility mechanism
  - The extender-based approach uses the OSGi installed bundle set as the extensibility mechanism
- Advantages and disadvantages for each
- Can be used independently or together

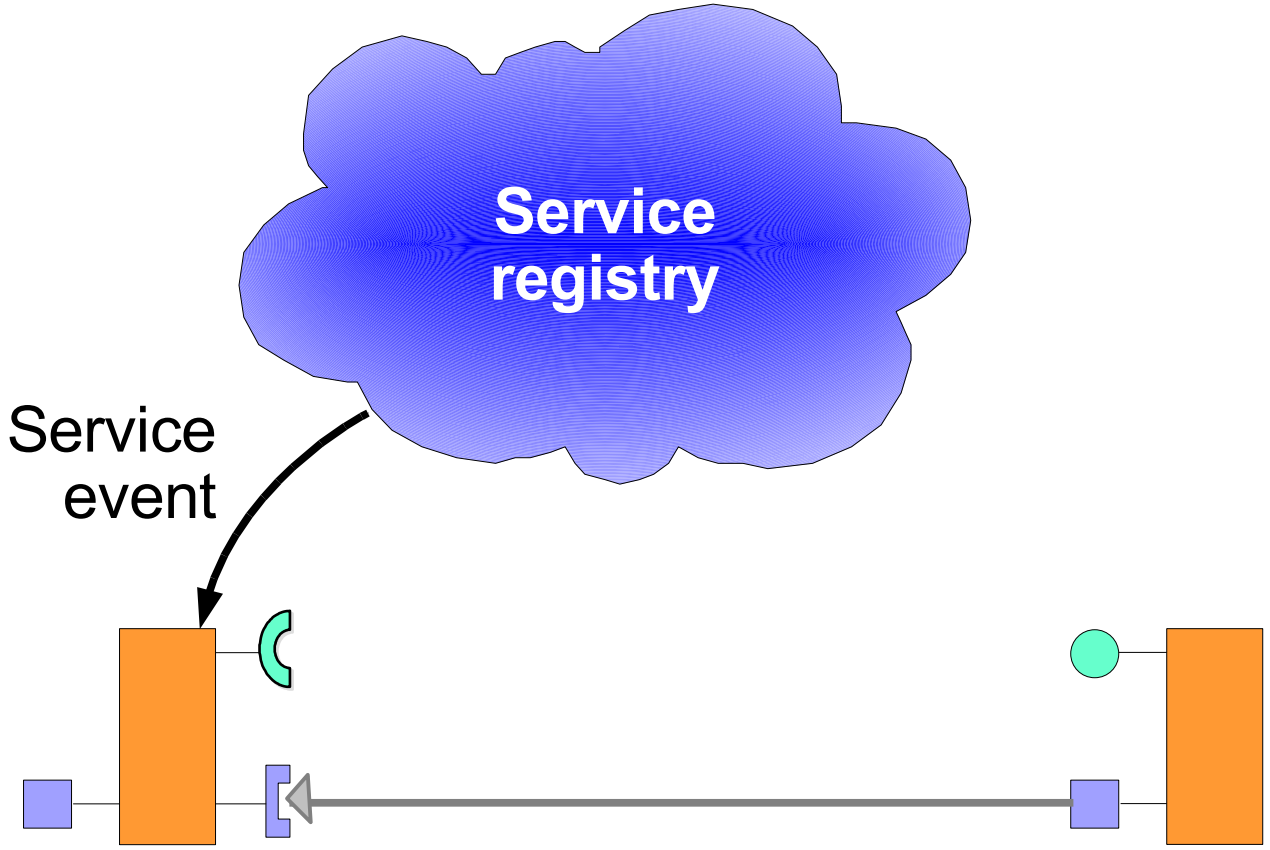


Service Model

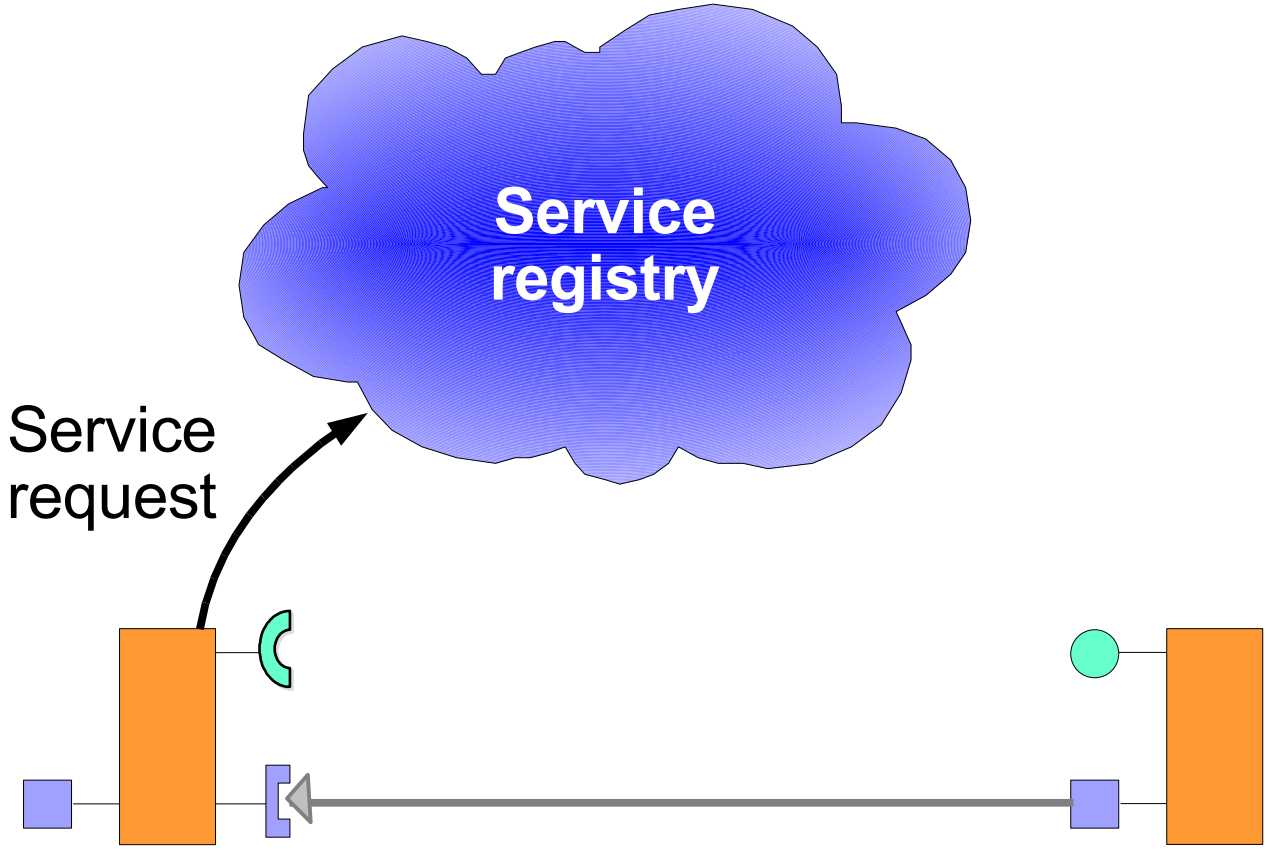
Service Model



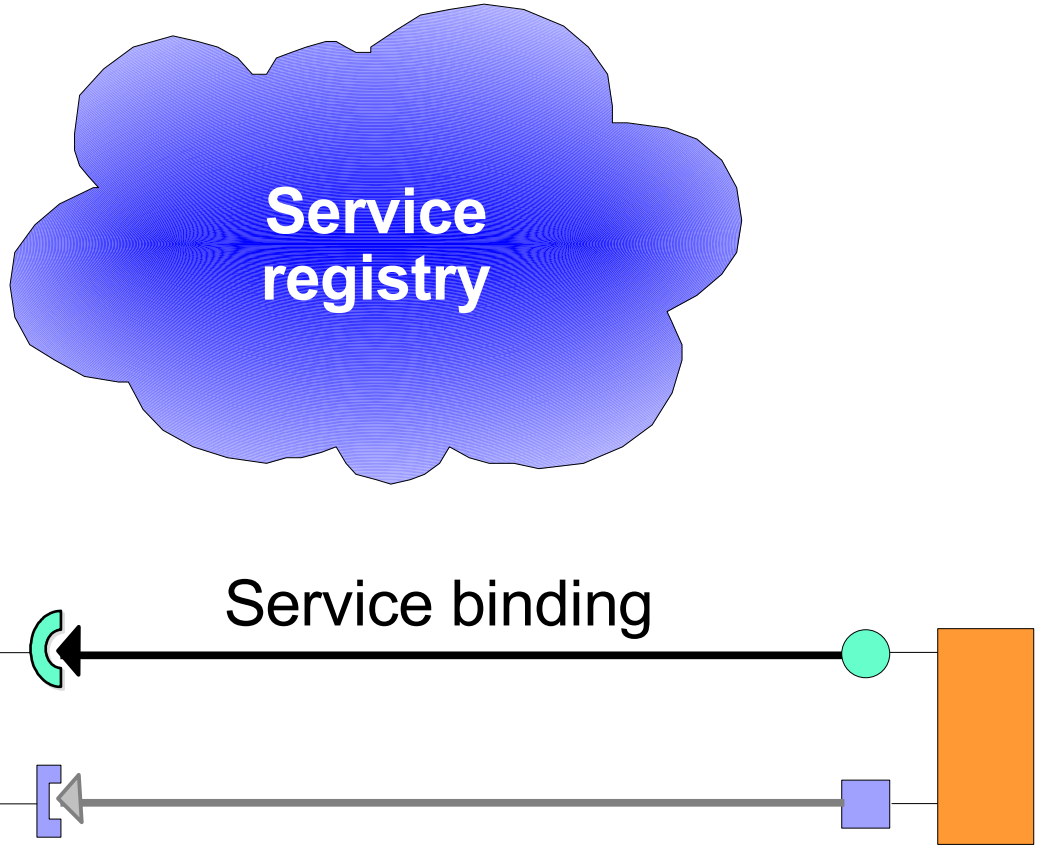
Service Model



Service Model

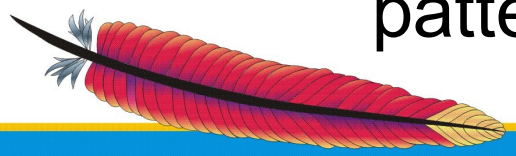


Service Model



Service Whiteboard Pattern

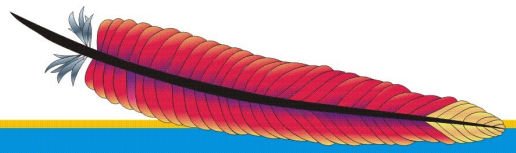
- Best practice
  - Instead of having clients look up and use a service interface, have clients register a service interface to express their interest
  - The service tracks the registered client interfaces and calls them when appropriate
- Simple, more robust, leverages the OSGi service model
- This is called the Whiteboard Pattern
  - It can be considered an Inversion of Control pattern



Service-Based Paint Program

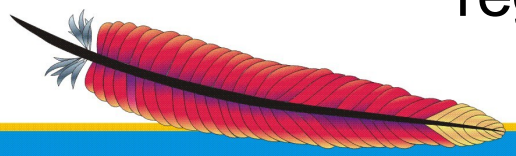
- `SimpleShape` service interface

```
public interface SimpleShape
{
    // A service property for the name of the shape.
    public static final String NAME_PROPERTY
        = "simple.shape.name";
    // A service property for the icon of the shape.
    public static final String ICON_PROPERTY
        = "simple.shape.icon";

    // Method to draw the shape of the service.
    public void draw(Graphics2D g2, Point p);
}
```

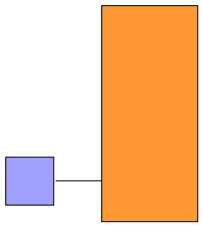


Service-Based Shape Tracker

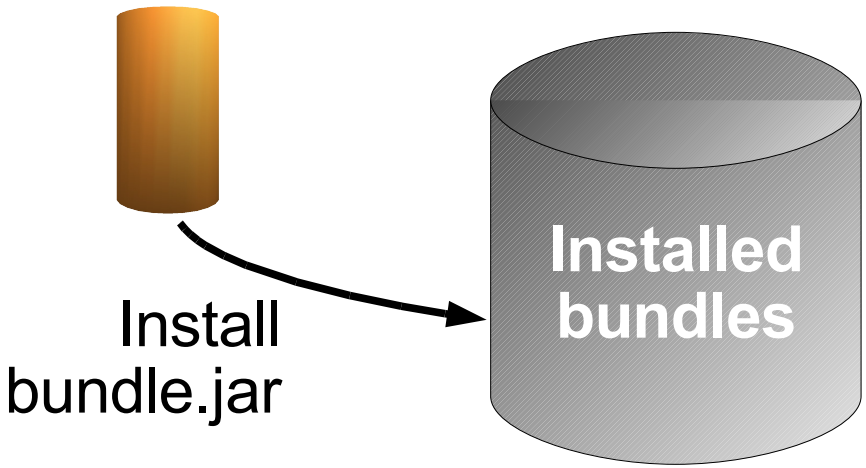
- Recall goal of the Shape Tracker
  - Use Inversion of Control principles to inject shapes into application
    - Puts tracking logic in one place
    - Isolates application from OSGi API
- Implemented as an OSGi Service Tracker subclass
  - Uses whiteboard pattern for services
  - Listens for `SimpleShape` service events
    - Result from service publications into OSGi service registry

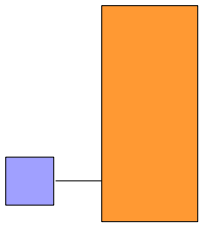


Extender Model

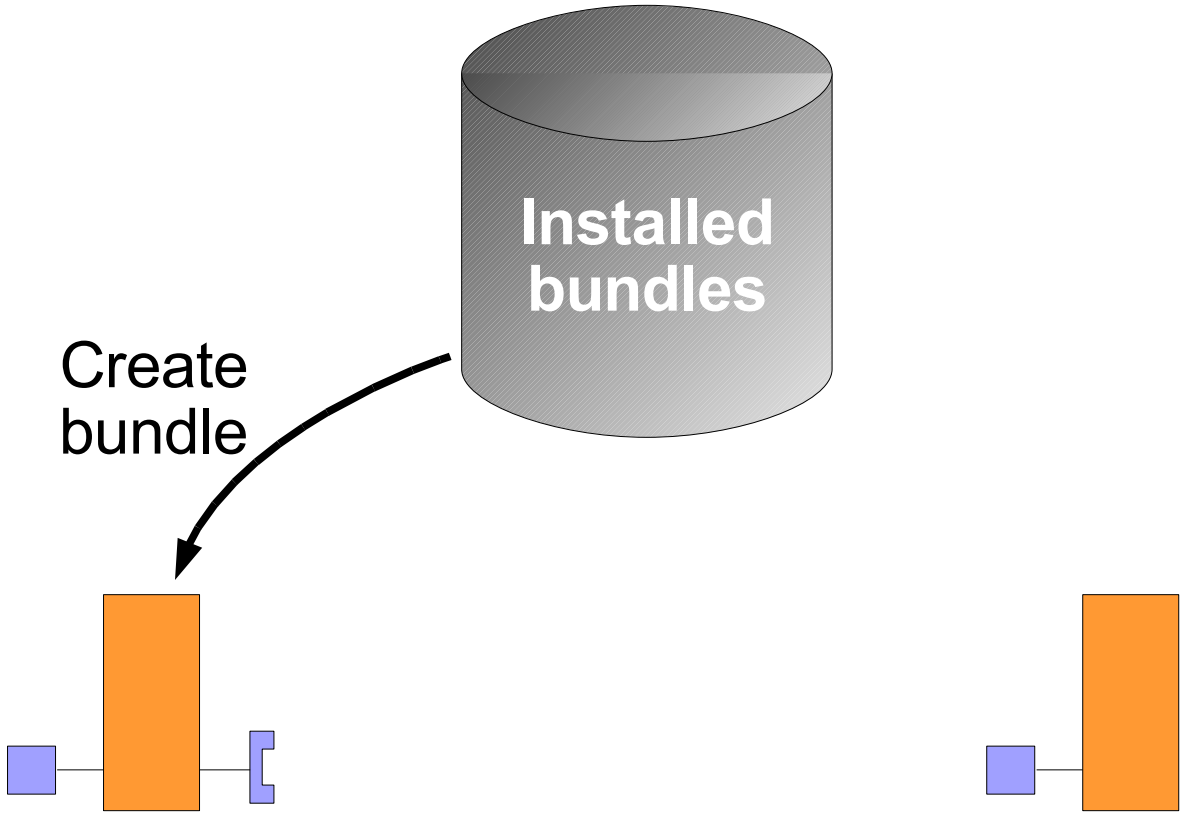


Extender Model

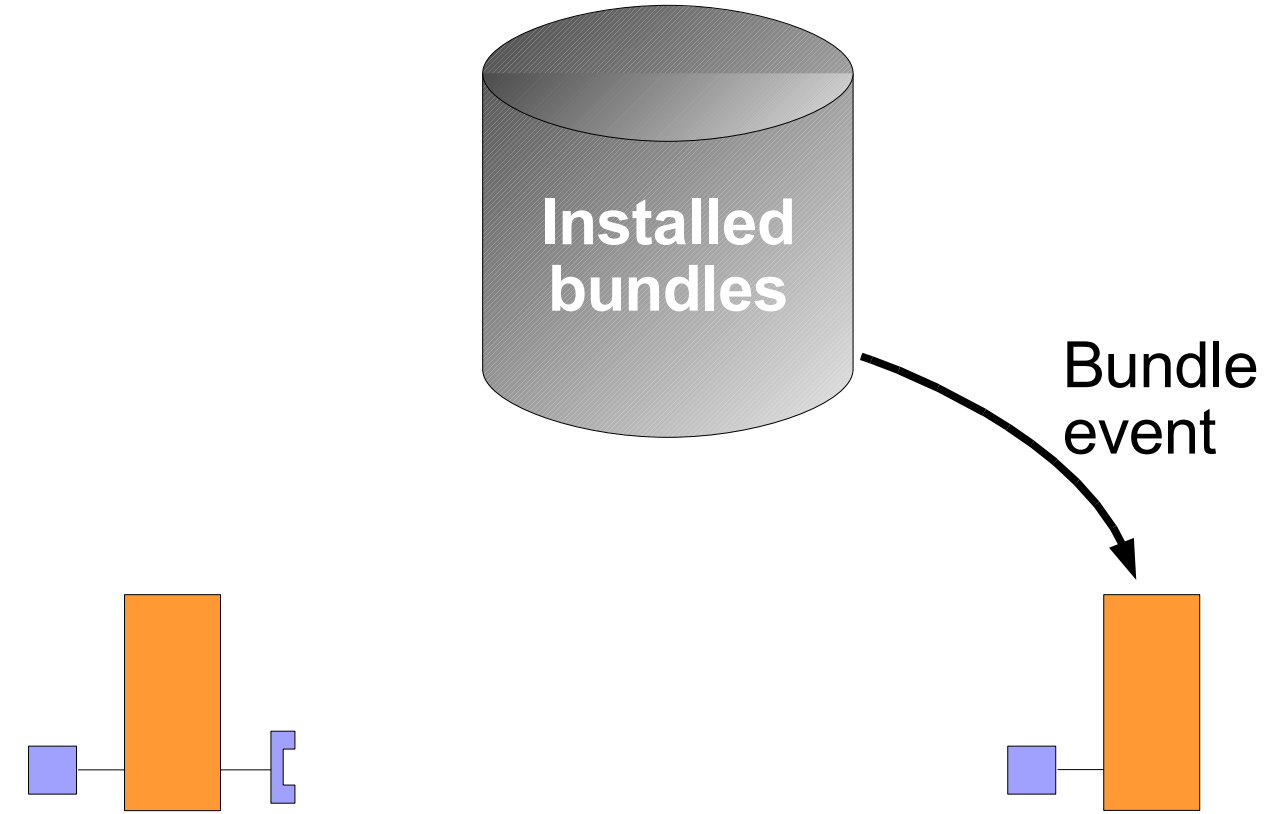




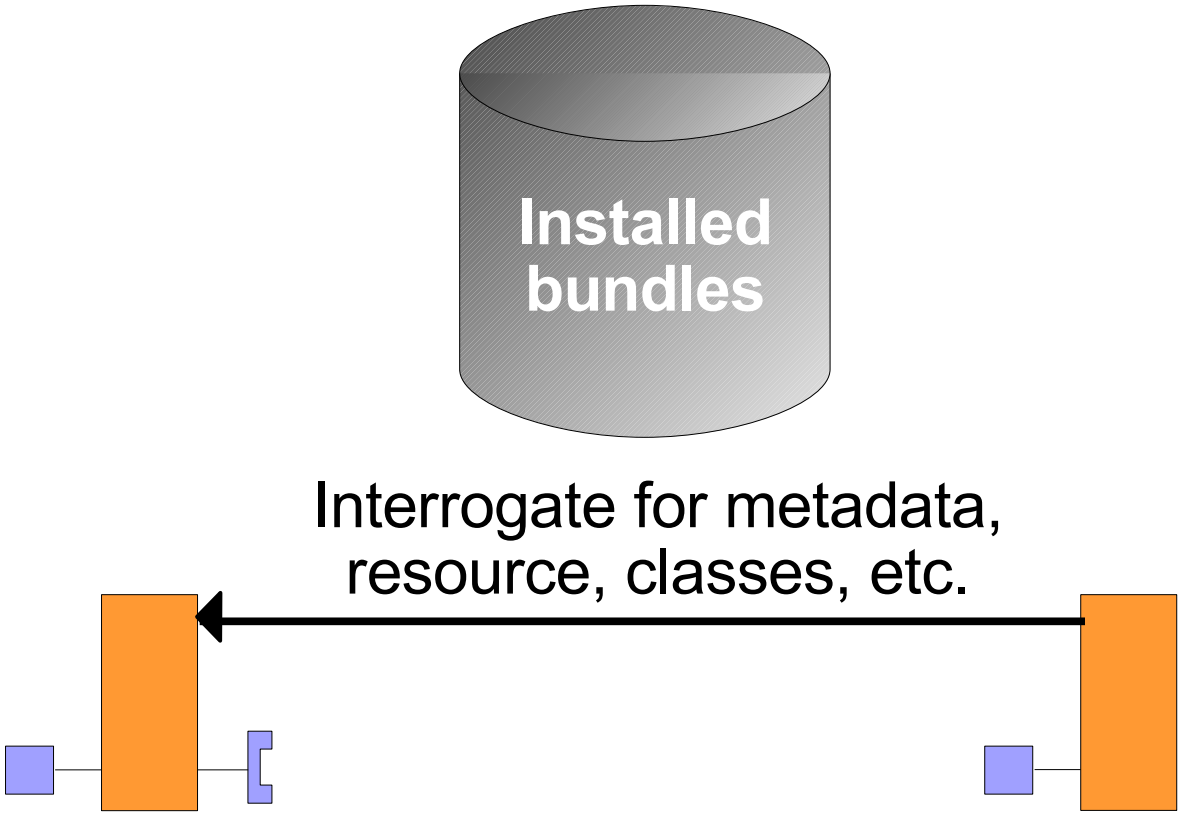
Extender Model



Extender Model



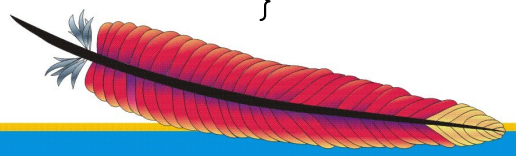
Extender Model



Extension-Based Paint Program

- `SimpleShape` extension interface

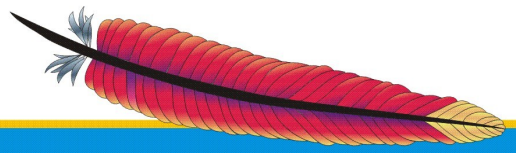
```
public interface SimpleShape
{
    // A property for the name of the shape.
    public static final String NAME_PROPERTY
        = "Extension-Name";
    // A property for the icon of the shape.
    public static final String ICON_PROPERTY
        = "Extension-Icon";
    // A property for the class of the shape.
    public static final String CLASS_PROPERTY
        = "Extension-Class";

    // Method to draw the shape of the extension.
    public void draw(Graphics2D g2, Point p);
}
```



Extension-Based Paint Program

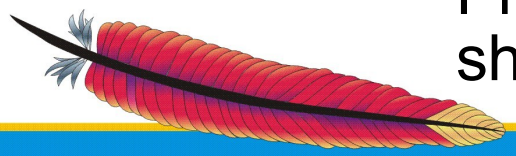
- Extension bundles include extension-related metadata in their JAR manifest
  - for example...

```
...
Extension-Name: Circle
Extension-Icon: org/apache/felix/circle/circle.png
Extension-Class: org.apache.felix.circle.Circle
...
```

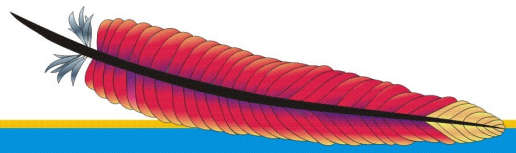


Extender-Based Shape Tracker

- Recall goal of the Shape Tracker
  - Use Inversion of Control principles to inject shapes into application
    - Puts tracking logic in one place
    - Isolates application from OSGi API
- Implemented as custom “bundle tracker”
  - Uses similar whiteboard pattern, but for installed bundles
  - Listens for bundle events
    - Specifically, STARTED and STOPPED events
    - Probes bundle manifests to see if bundles provide shape extensions



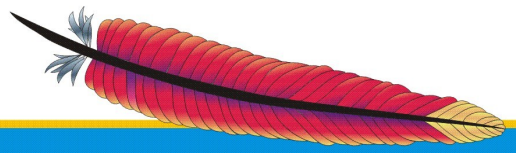
ApacheCon



⑥ Example Application Demo

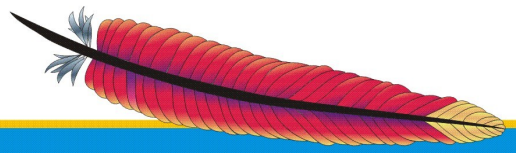
⑦ ⑧

7 Advances Issues



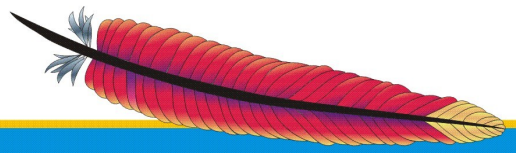
Bundled vs. Hosted

- Applications can leverage OSGi functionality in two ways
  - Build entire application as a set of bundles that will run on top of a framework instance
  - Host a framework instance inside application and externally interact with bundles/services



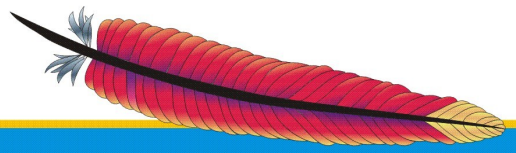
Bundled vs. Hosted

- Building application as a set of bundles is the preferred approach
  - Allows application to run on any framework
  - Not always possible for legacy applications
- Hosting framework instance allows piecemeal OSGi adoption
  - Will likely tie application to a framework implementation



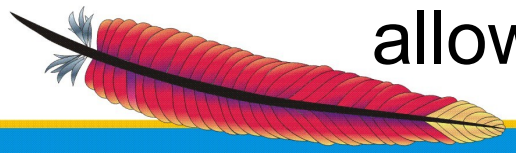
Hosted Framework

- More complicated since due to external/internal gap
  - e.g., unlike bundles, the host application does not have a bundle context by which it can access framework services
- Required host/framework interactions
  - Accessing framework functionality
  - Providing services to bundles
  - Using services from bundles



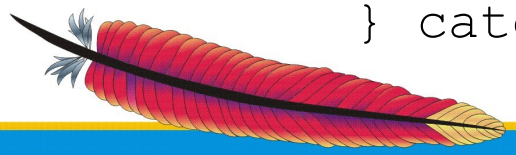
Hosted Framework

- Felix tries to simplify hosted instance scenarios
  - All configuration data is passed into constructor
  - Felix framework implements `Bundle` interface and acts as the System Bundle
    - Gives the host application an intuitive way to access framework functionality
  - Felix constructor also accepts “constructor activators” to extend system bundle
  - Felix tries to multiplex singleton resources to allow for multiple framework instances



Hosted Framework

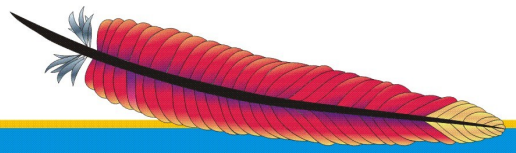
```
// Define configuration properties
Map configMap = new StringMap(false);
configMap.put(..., ...);
...
// Create application activators
List list = new ArrayList();
list.add(new Activator());

try {
    // Create a framework instance
    Felix felix = new Felix(configMap, list);
    // Start framework instance
    felix.start();
    ...
    // Stop framework instance
    felix.stop();
} catch (Exception ex) { ... }
```



Hosted Framework

- Providing a host application service

```
BundleContext bc = felix.getBundleContext();
bc.registerService(Service.class, svcObj, null);
```



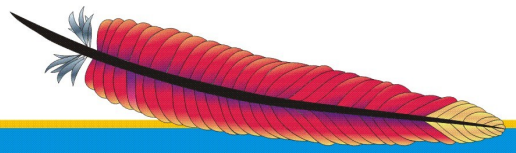
Hosted Framework

- Providing a host application service

```
BundleContext bc = felix.getBundleContext();
bc.registerService(Service.class, svcObj, null);
```

- Accessing internal bundle services

```
BundleContext bc = felix.getBundleContext();
ServiceReference ref =
   bc.getServiceReference(Service.class);
Service svcObj = (Service) bc.getService(ref);
```



Hosted Framework

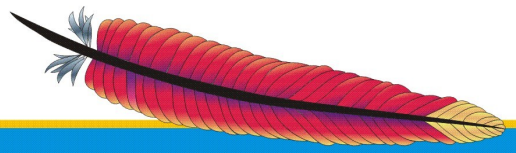
- Providing a host application service

```
BundleContext bc = felix.getBundleContext();
bc.registerService(Service.class, svcObj, null);
```

- Accessing internal bundle services

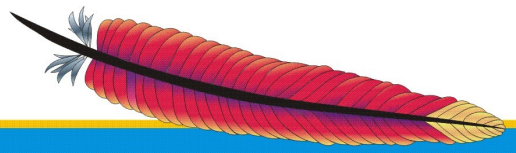
```
BundleContext bc = felix.getBundleContext();
ServiceReference ref =
   bc.getServiceReference(Service.class);
Service svcObj = (Service) bc.getService(ref);
```

- Better approach is to use a constructor activators since it is integrated with System Bundle (i.e., framework) starting and stopping



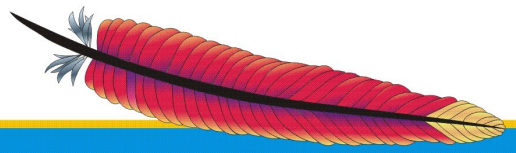
Hosted Framework

- Classes shared among host application and bundles ***must*** be on the application class path
  - Disadvantage of hosted framework approach, which limits dynamics
  - Use of reflection by host to access bundle services can eliminate this issue, but it is still not an optimal solution
- In summary, better to completely bundle your application if possible

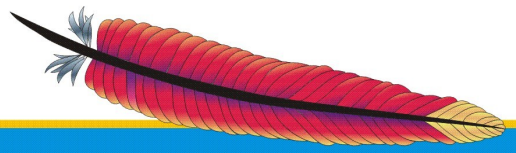


Custom Life Cycle Layer

- [placeholder]
- Can separate service and life cycle layers from modularity layer
- Create your own life cycle layer using the extender model to incorporate your own component model life cycle layer and/or component interaction layer



ApacheCon



⑧ Conclusion