OSGi University

Marcel Offermans
Karl Pauls

luminis

Agenda

- History of OSGi
- The Framework
- The Compendium
- OSGi Application Approaches
- Embedding
- Managing Service Dependencies
- Development Environment
- Open Source Frameworks

Agenda

- History of OSGi
- The Framework
- The Compendium
- OSGi Application Approaches
- Embedding
- Managing Service Dependencies
- Development Environment
- Open Source Frameworks

OSGi history

- Started as an embedded platform for the “home gateway”
- Originally under the JCP as JSR-8 (1999)
- OSGi alliance, consists of a large number of big companies, with the following mission:
  - Maintaining and publicizing the OSGi specification.
  - Certifying implementations.
  - Organising events.
- Current version: OSGi Release 4.1 (JSR-291)

OSGi releases

- R1: long ago :)
- R2: october 2001
  - Java Embedded Server (Sun), Oscar (SourceForge)
- R3: march 2003
  - Knopflerfish
- R4: august 2005
  - IBM joined and influenced this release, Equinox (Eclipse Foundation)
- R4.1: april 2007

OSGi today

OSGi technology is the dynamic module system for Java™

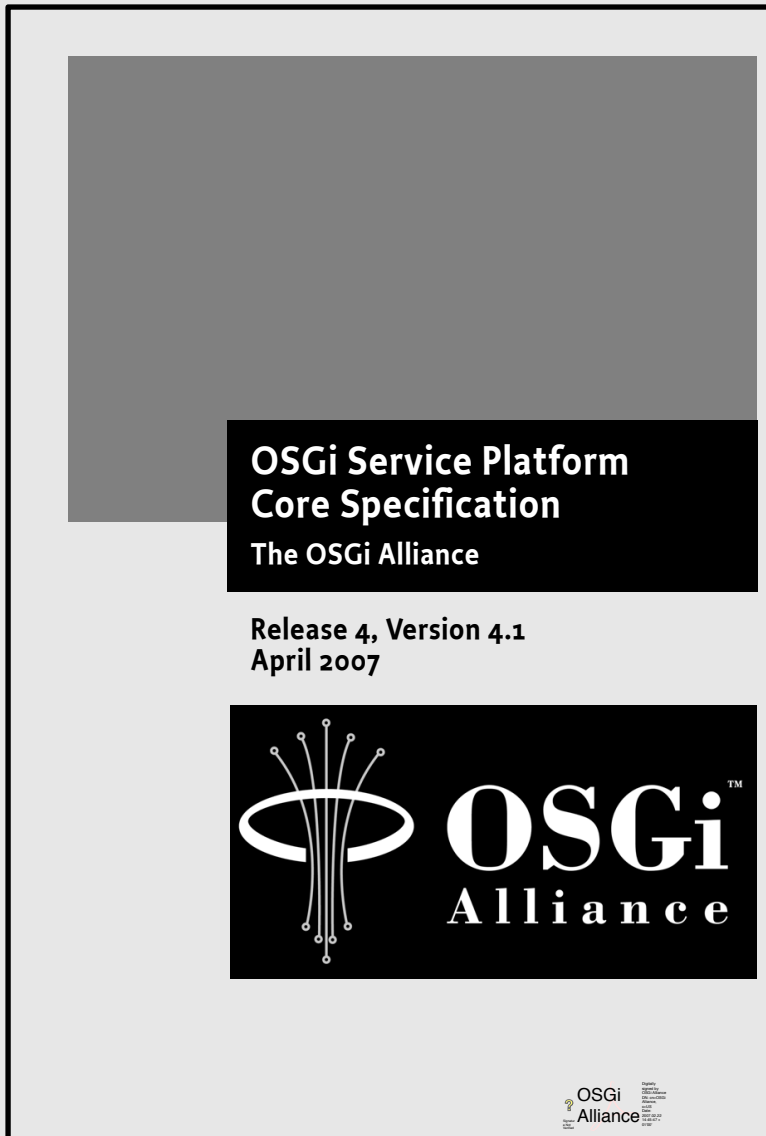
OSGi technology is Universal Middleware.

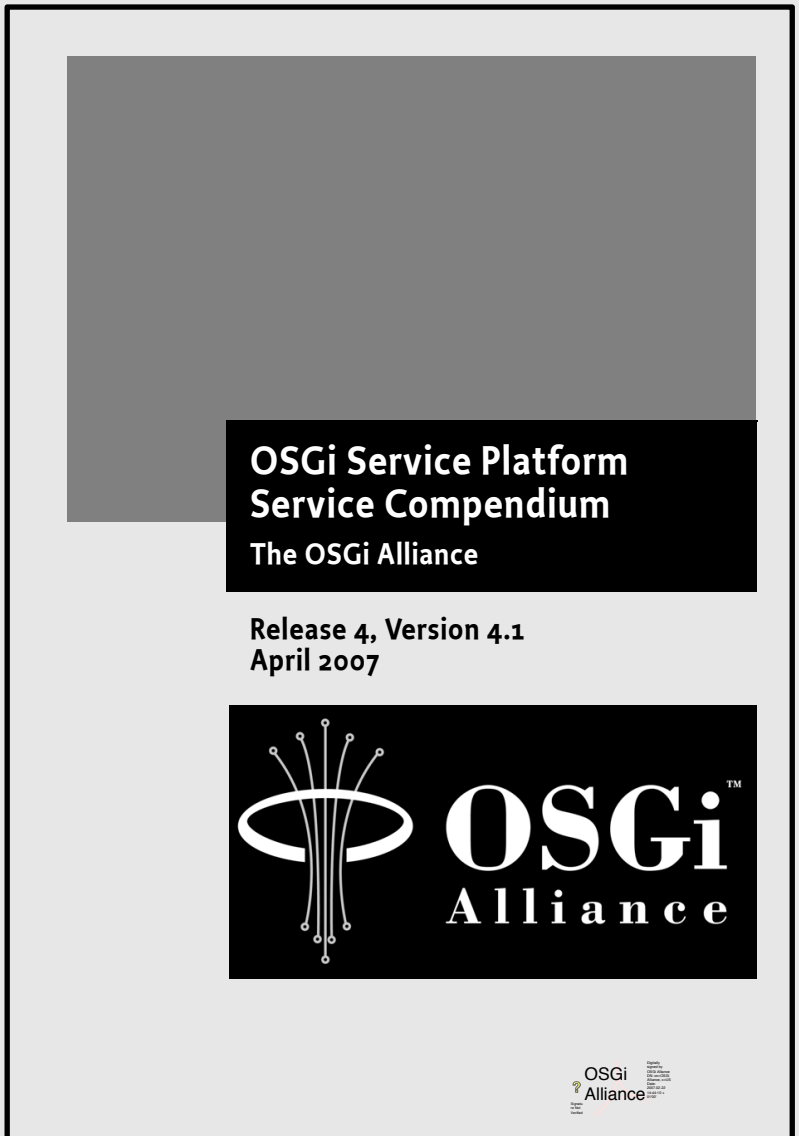
OSGi technology provides a service-oriented, component-based environment for developers and offers standardized ways to manage the software lifecycle. These capabilities greatly increase the value of a wide range of computers and devices that use the Java™ platform.

OSGi Alliance

- Expert Groups:
  - core platform (CPEG)
  - mobile (MEG)
  - vehicle (VEG)
  - enterprise (EEG)
  - residential (REG)
- Working Groups:
  - marketing
  - requirements

OSGi specification

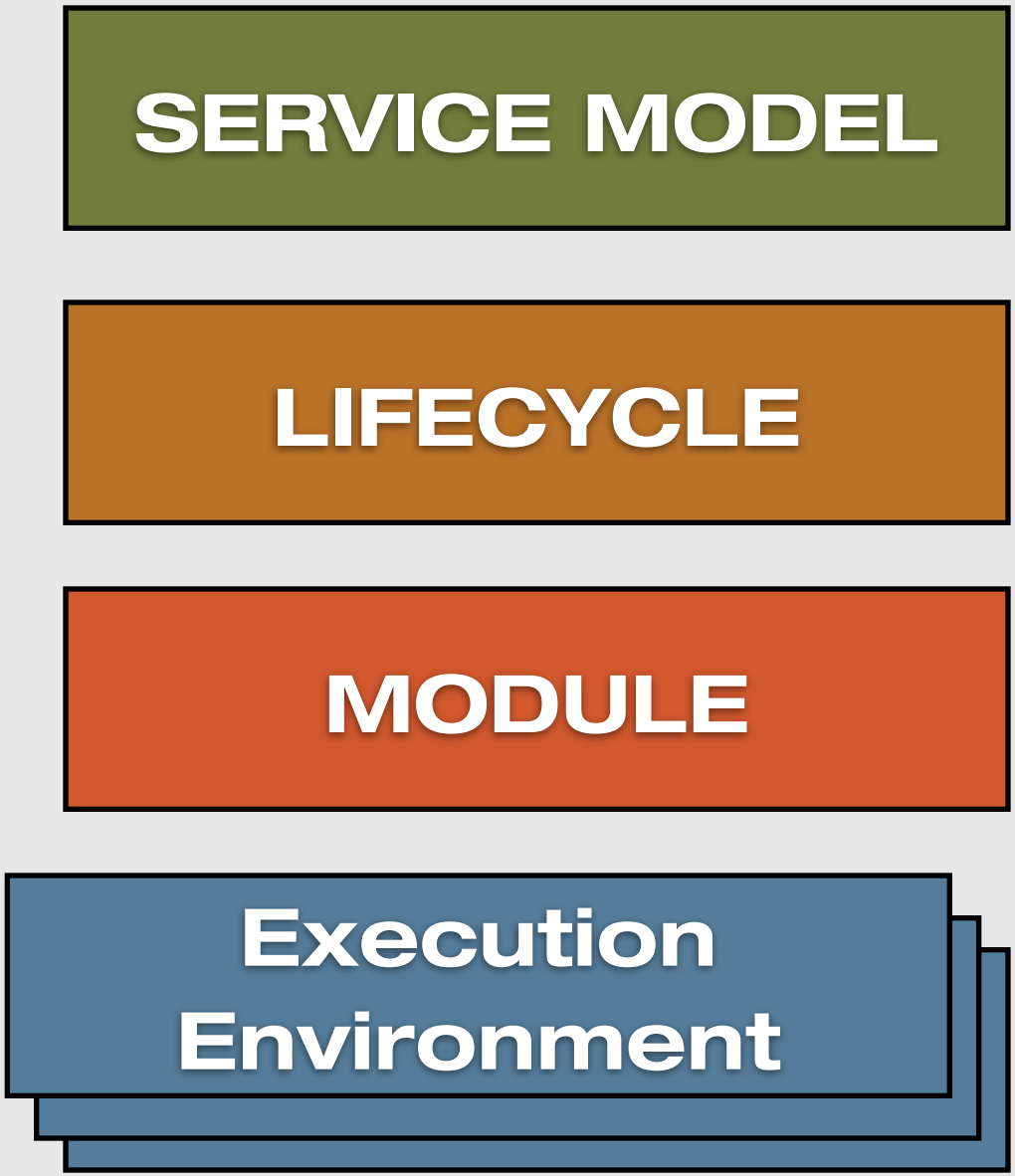




Agenda

- History of OSGi
- The Framework
- The Compendium
- OSGi Application Approaches
- Embedding
- Managing Service Dependencies
- Development Environment
- Open Source Frameworks

OSGi Framework Layering



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

L2 - Manages the life cycle of a bundle in a framework without requiring the vm to be restarted

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

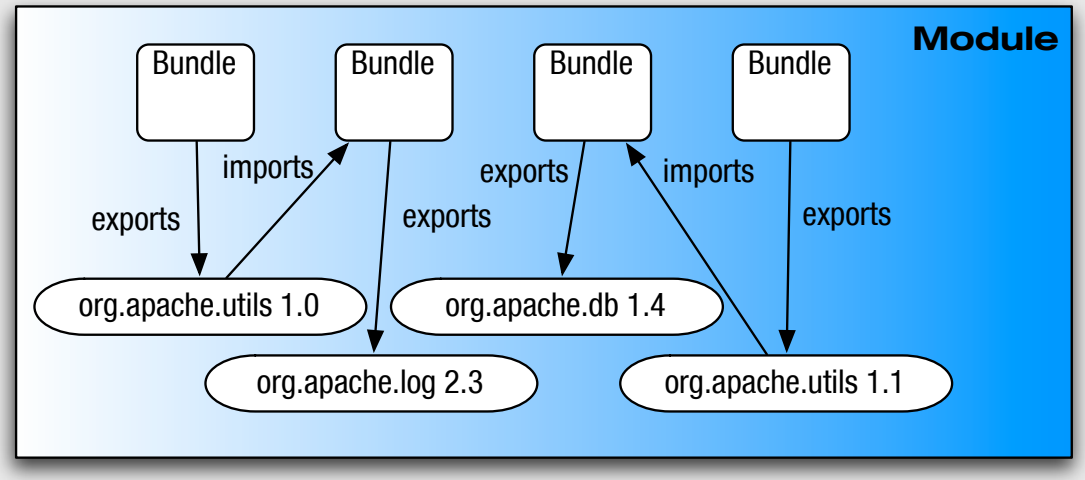
L0 -

OSGi Minimum Execution Environment
CDC/Foundation
JavaSE

Module Layer (1/3)

- Unit of deployment
  is the bundle i.e., a JAR
- Separate class loader
  per bundle
  - Class loader graph
  - Independent namespaces
  - Class sharing at the Java package level

Module Layer (1/3)

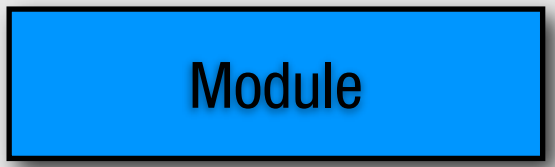


- Unit of deployment is the bundle i.e., a JAR
- Separate class loader per bundle
  - Class loader graph
  - Independent namespaces
  - Class sharing at the Java package level

Module

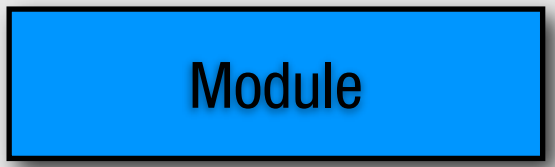
Module Layer (2/3)

- 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
- Arbitrary export/import attributes
  - Influence package selection



Module Layer (3/3)

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



Manifest

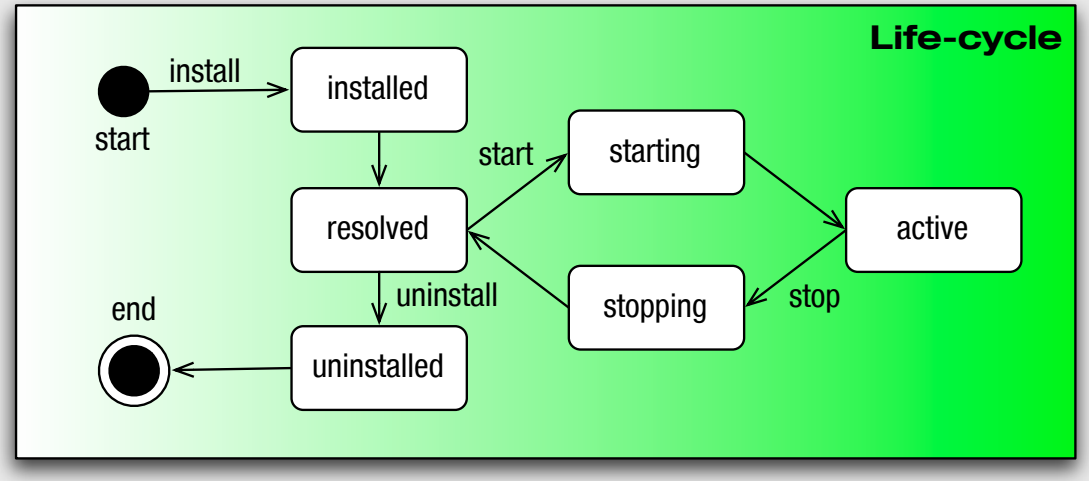
```
Bundle-Name: Example Bundle
Bundle-SymbolicName: net.luminis.example.bundle
Bundle-Version: 1.0.0
DynamicImport-Package:
    org.osgi.service.log
Import-Package:
    org.osgi.framework;version=“1.3“,
    org.osgi.service.event;version=“[1.1,2.0)“,
    net.luminis.foo;resolution:=“optional“
Export-Package:
    org.osgi.service.event;uses:=org.osgi.framework;version=“1.1“
Bundle-ManifestVersion: 2
```

Life-cycle Layer

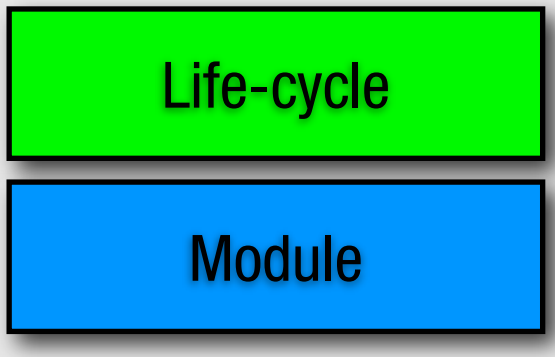
- Managed life cycle
  - States for each bundle;
- Allows updates of existing bundles.
  - Dynamically install, start, update, and uninstall

Module

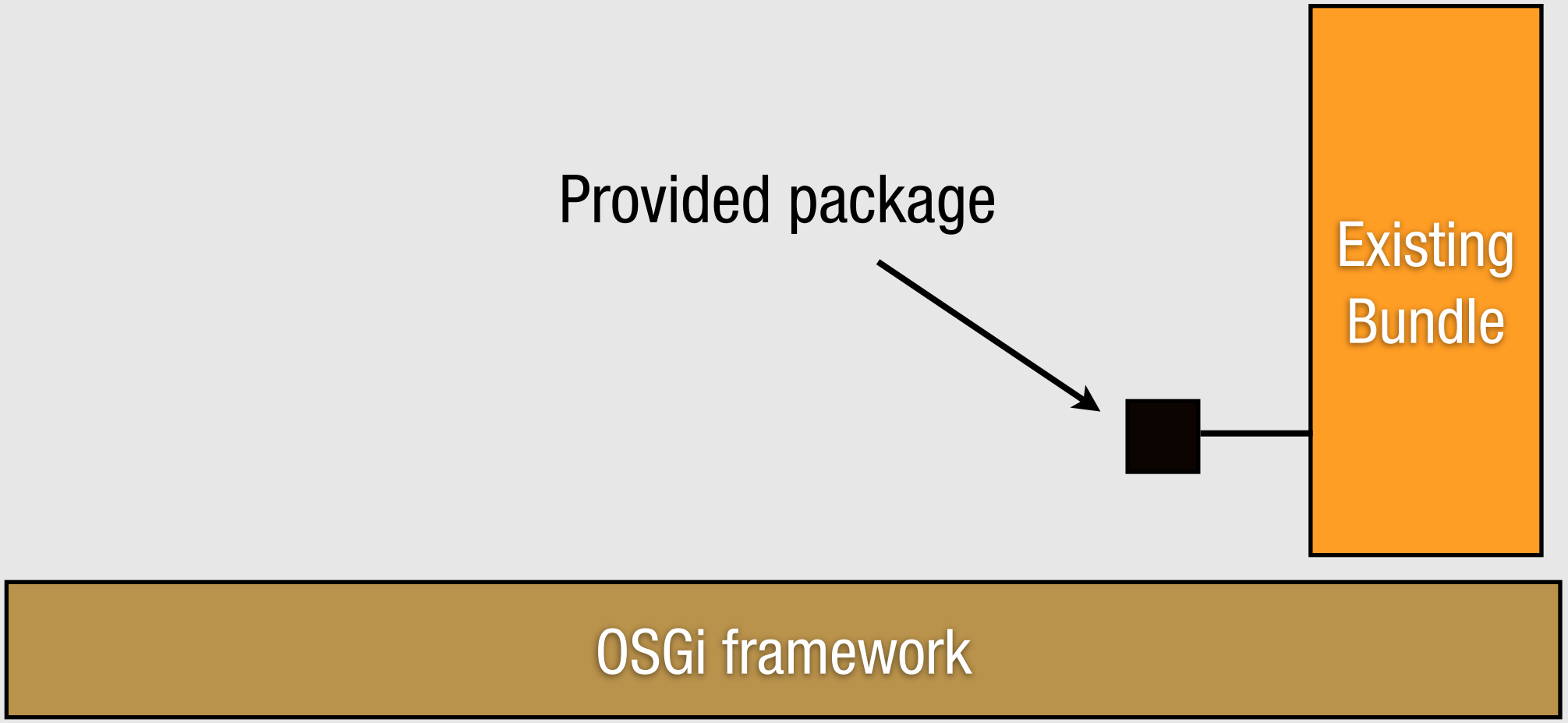
Life-cycle Layer



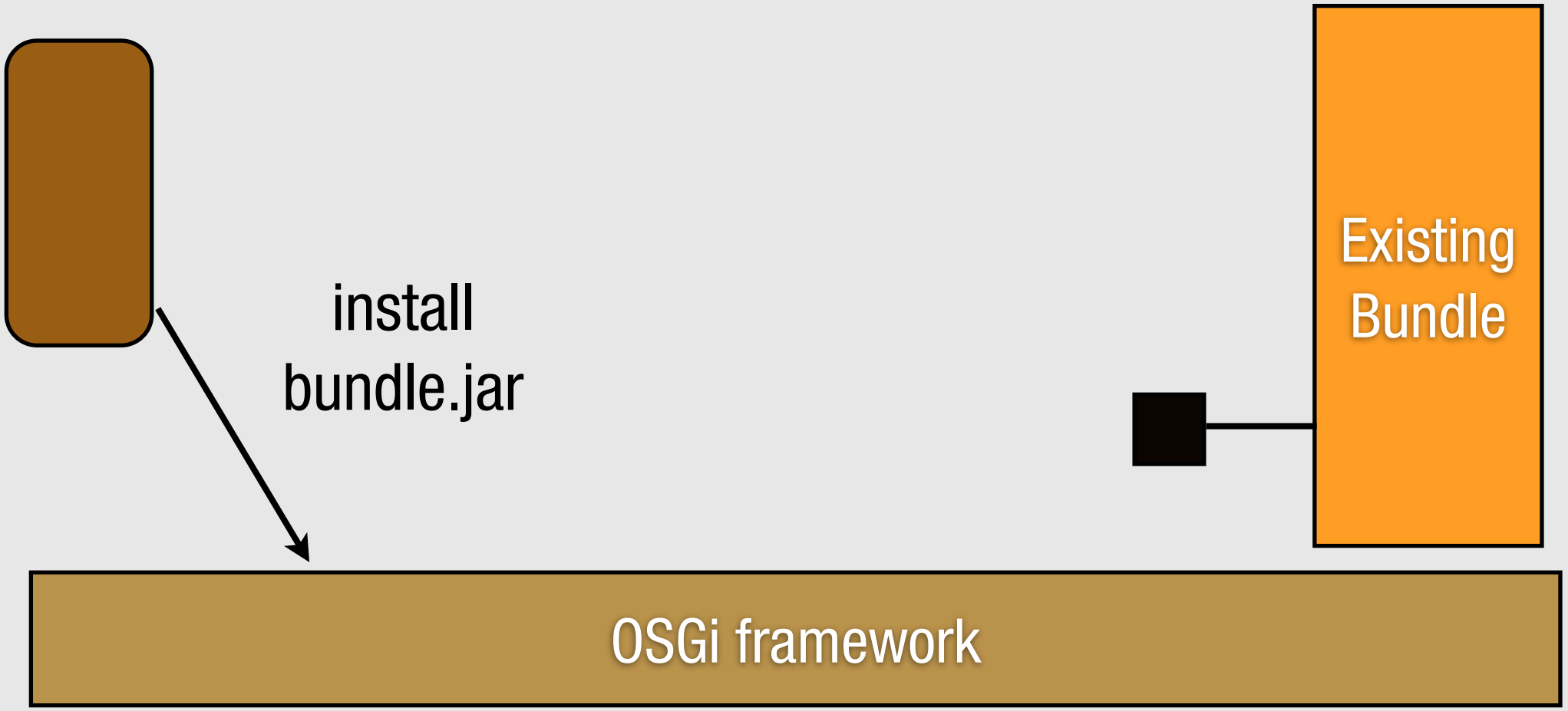
- Managed life cycle
  - States for each bundle;
- Allows updates of existing bundles.
  - Dynamically install, start, update, and uninstall



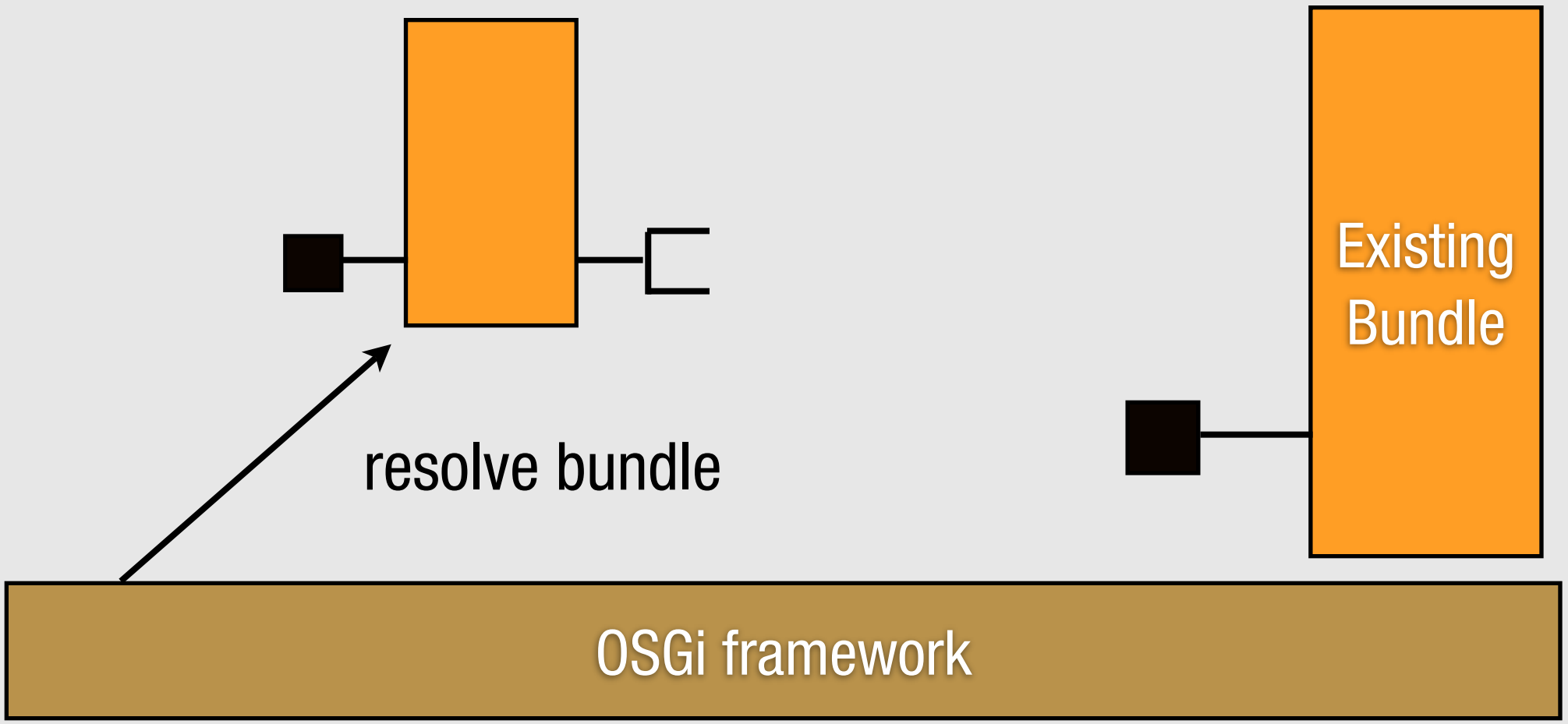
Visualization



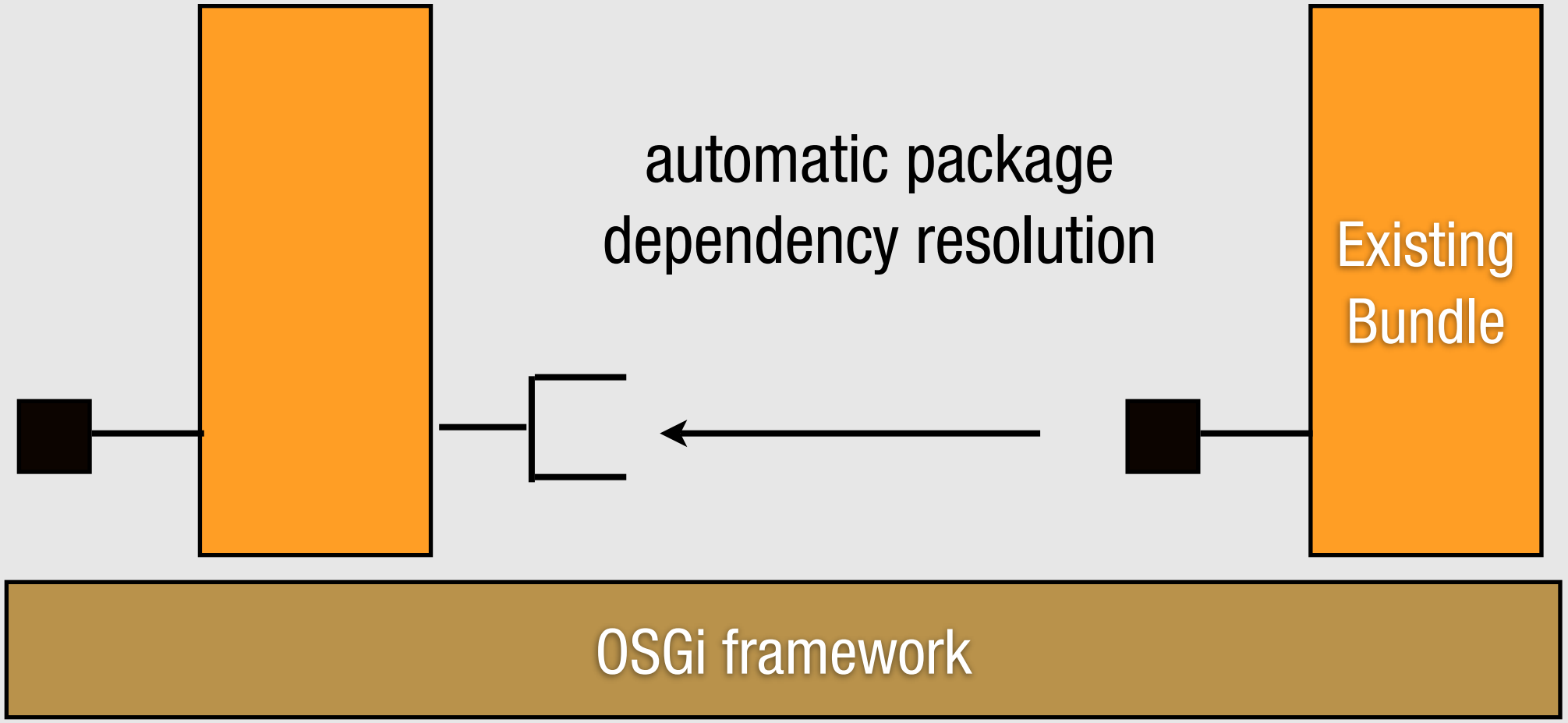
Visualization



Visualization

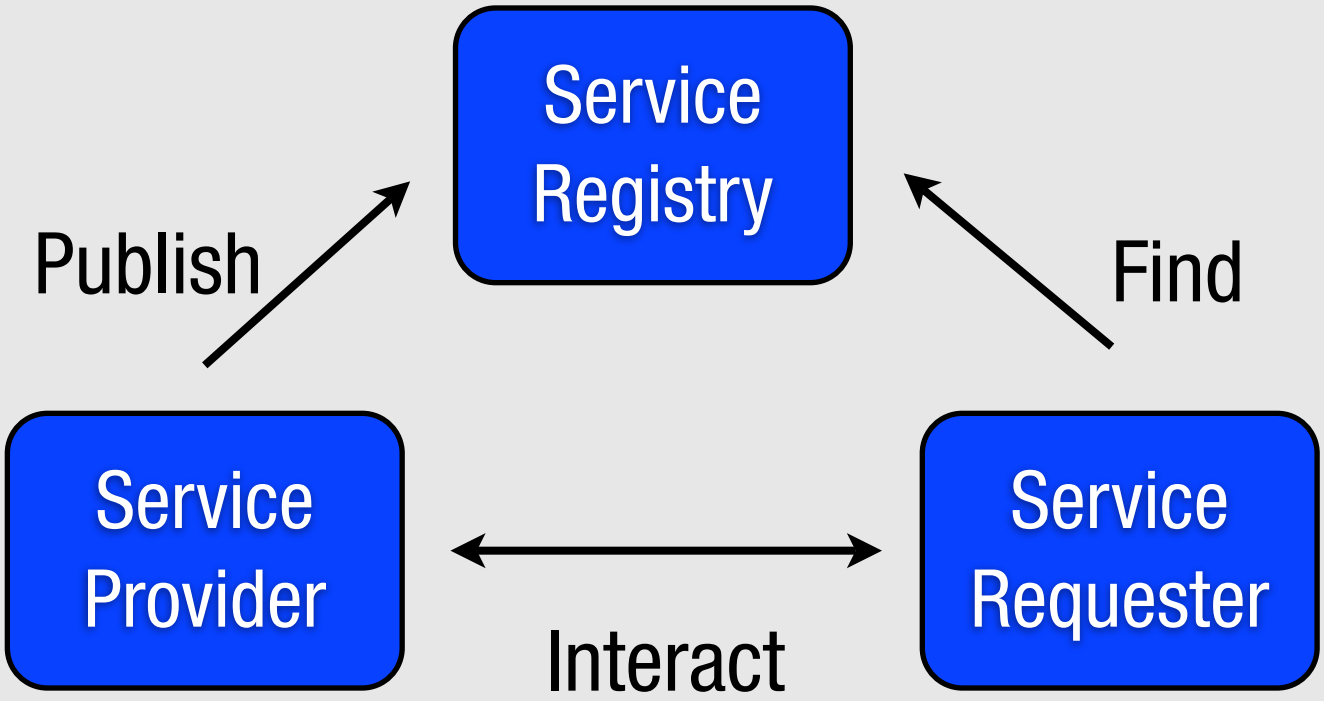


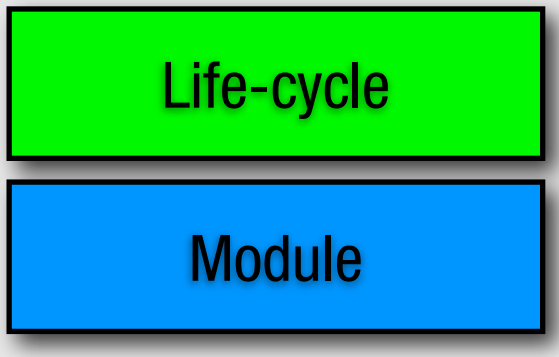
Visualization



Service Layer

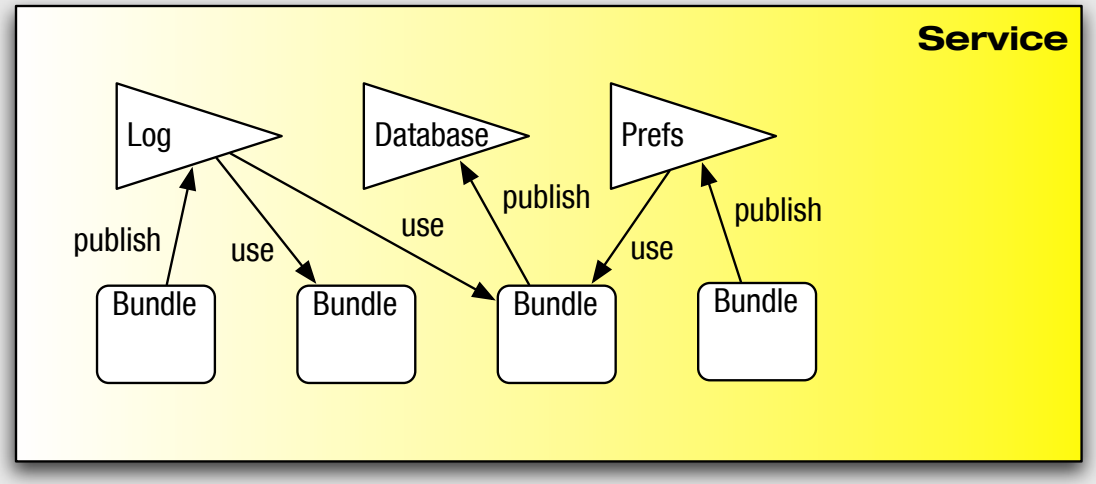
- OSGi framework promotes service oriented interaction pattern among bundles

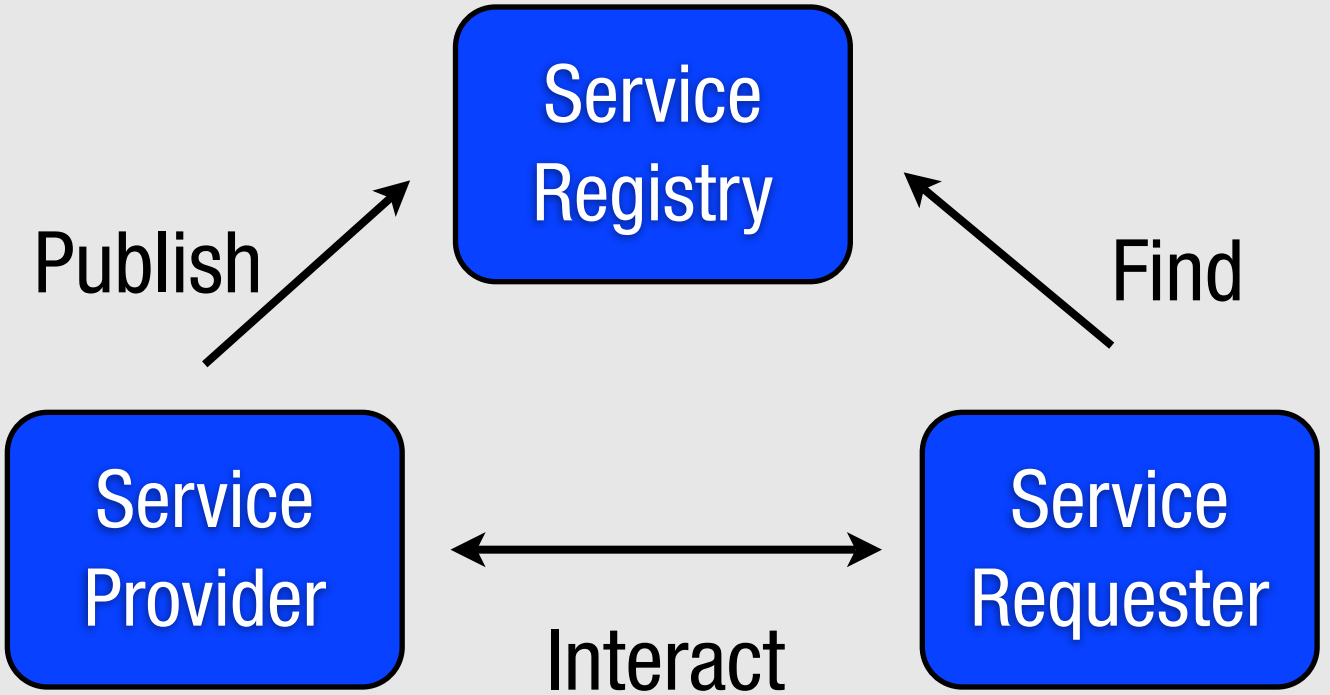


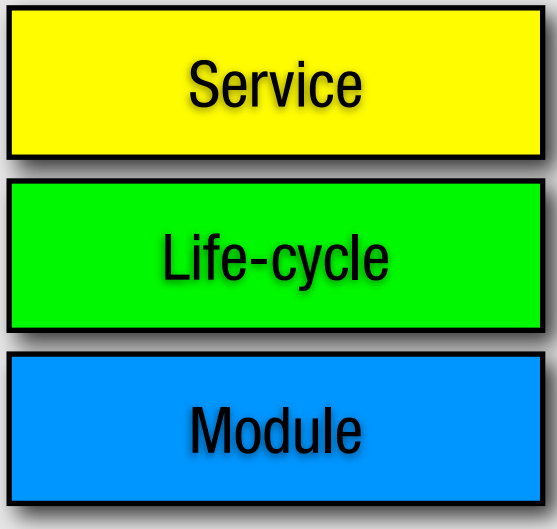


Service Layer

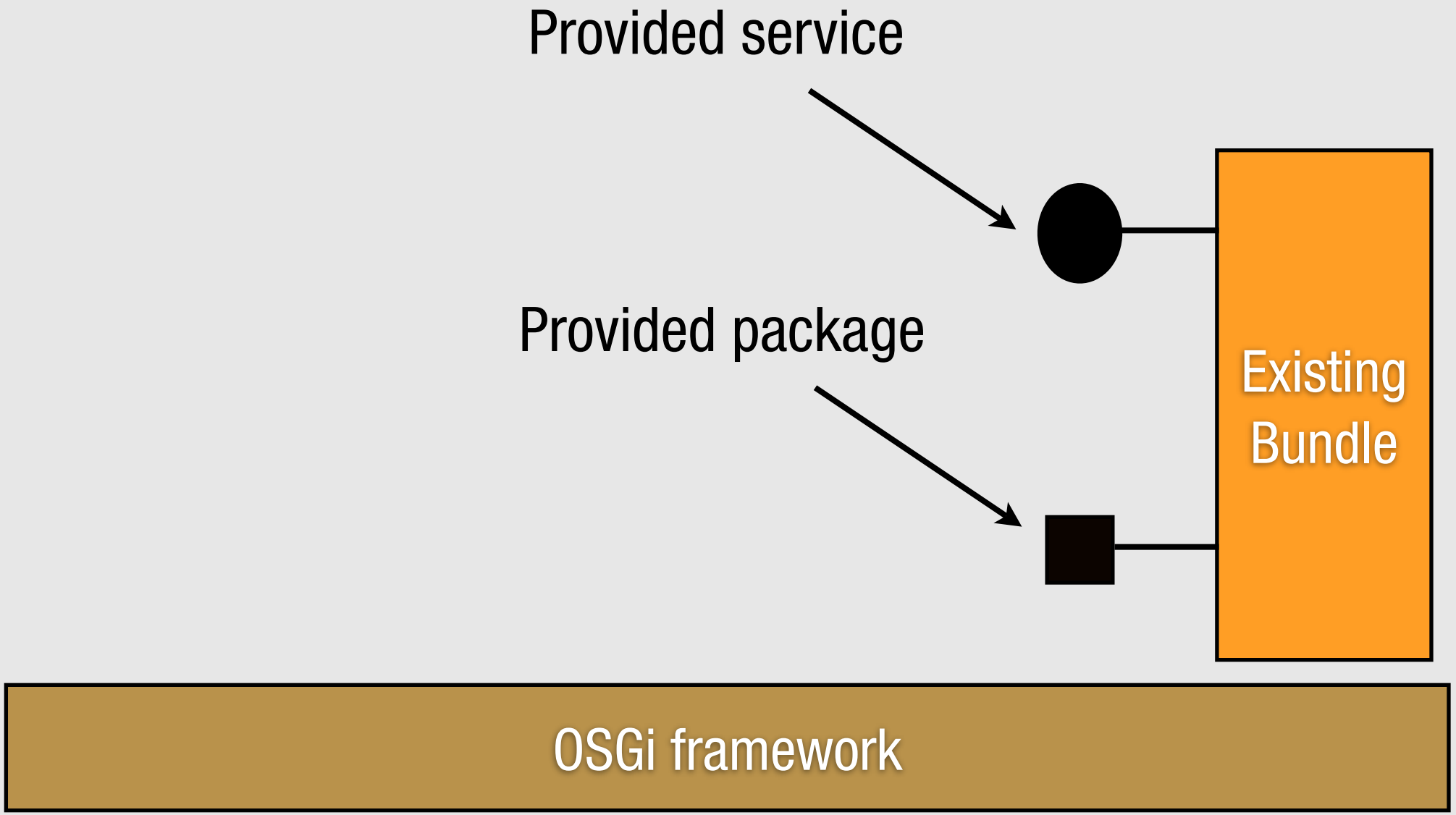
- OSGi framework promotes service oriented interaction pattern among bundles



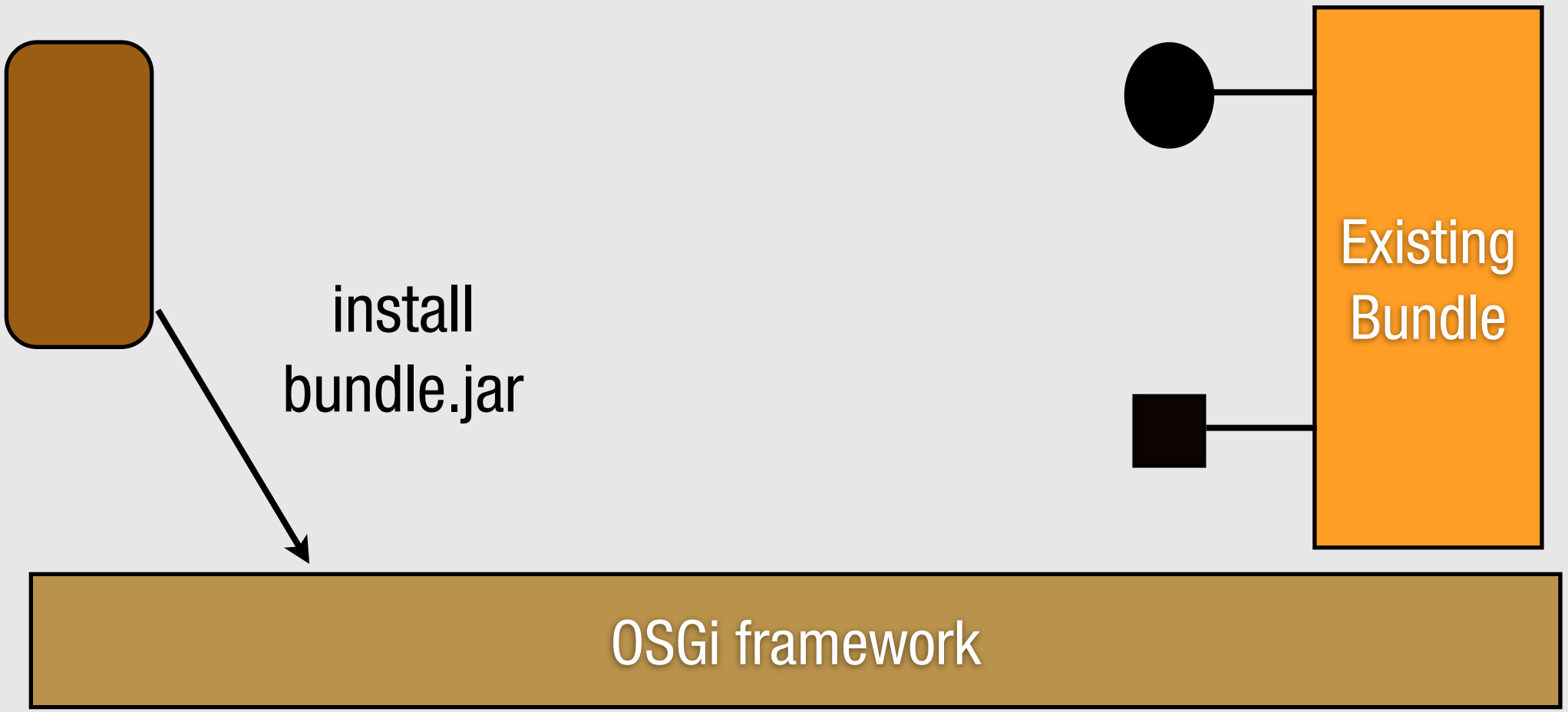




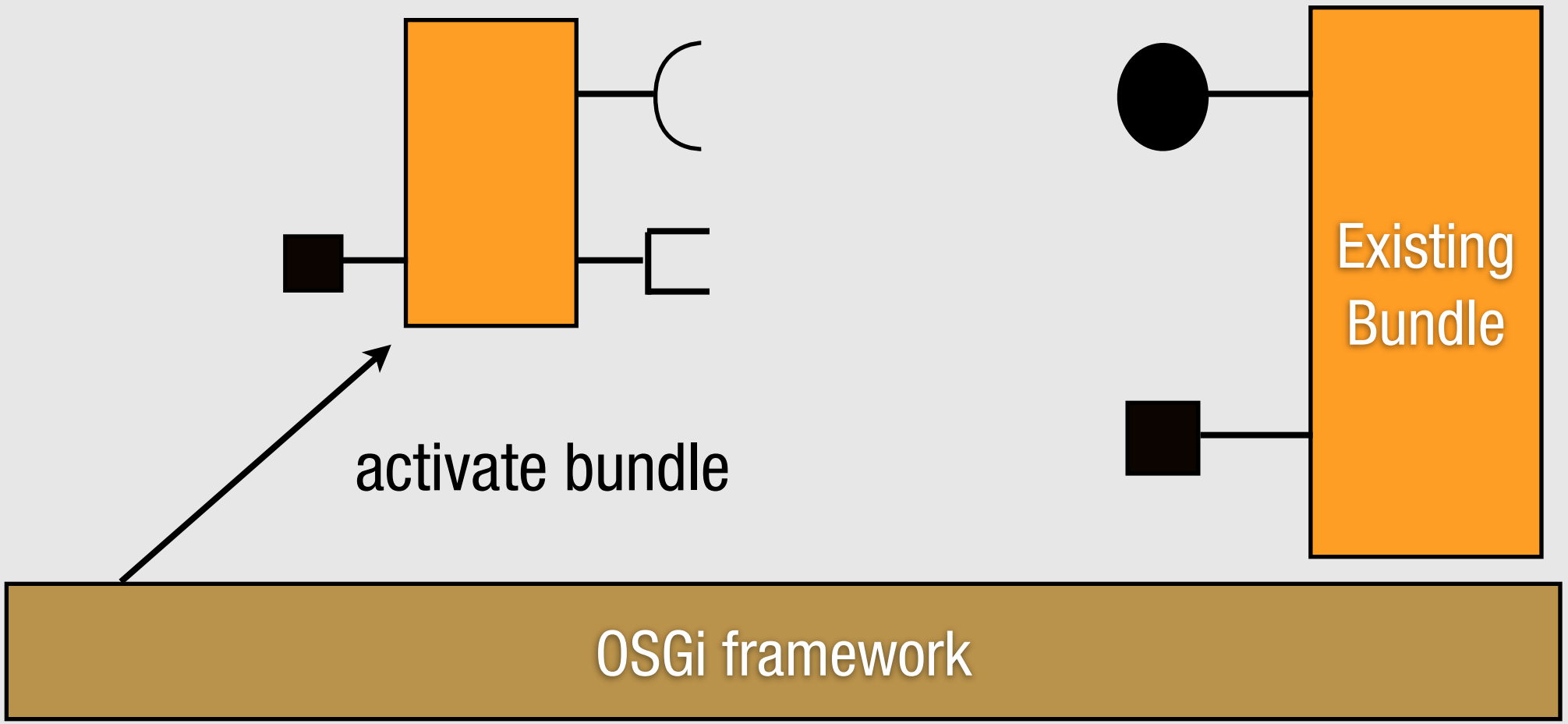
Visualization



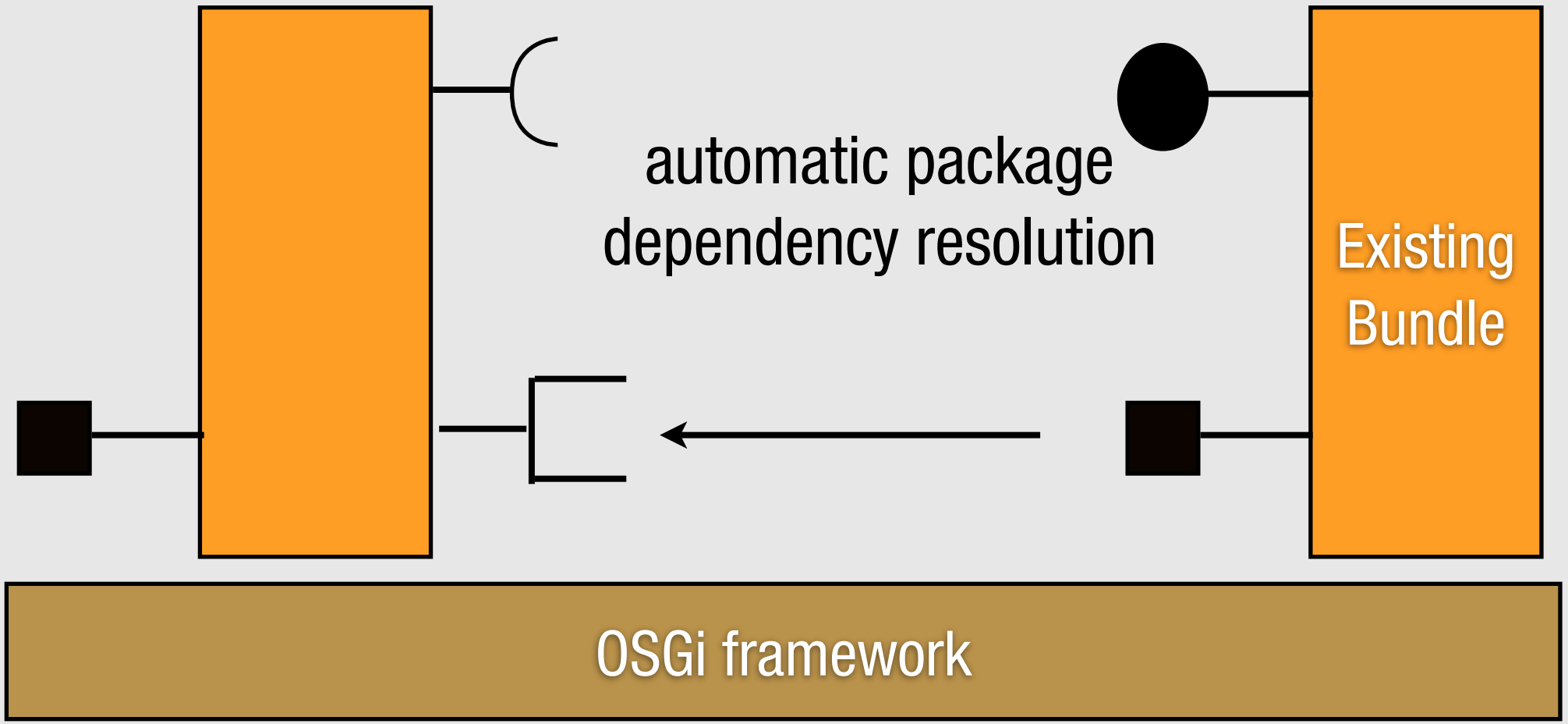
Visualization



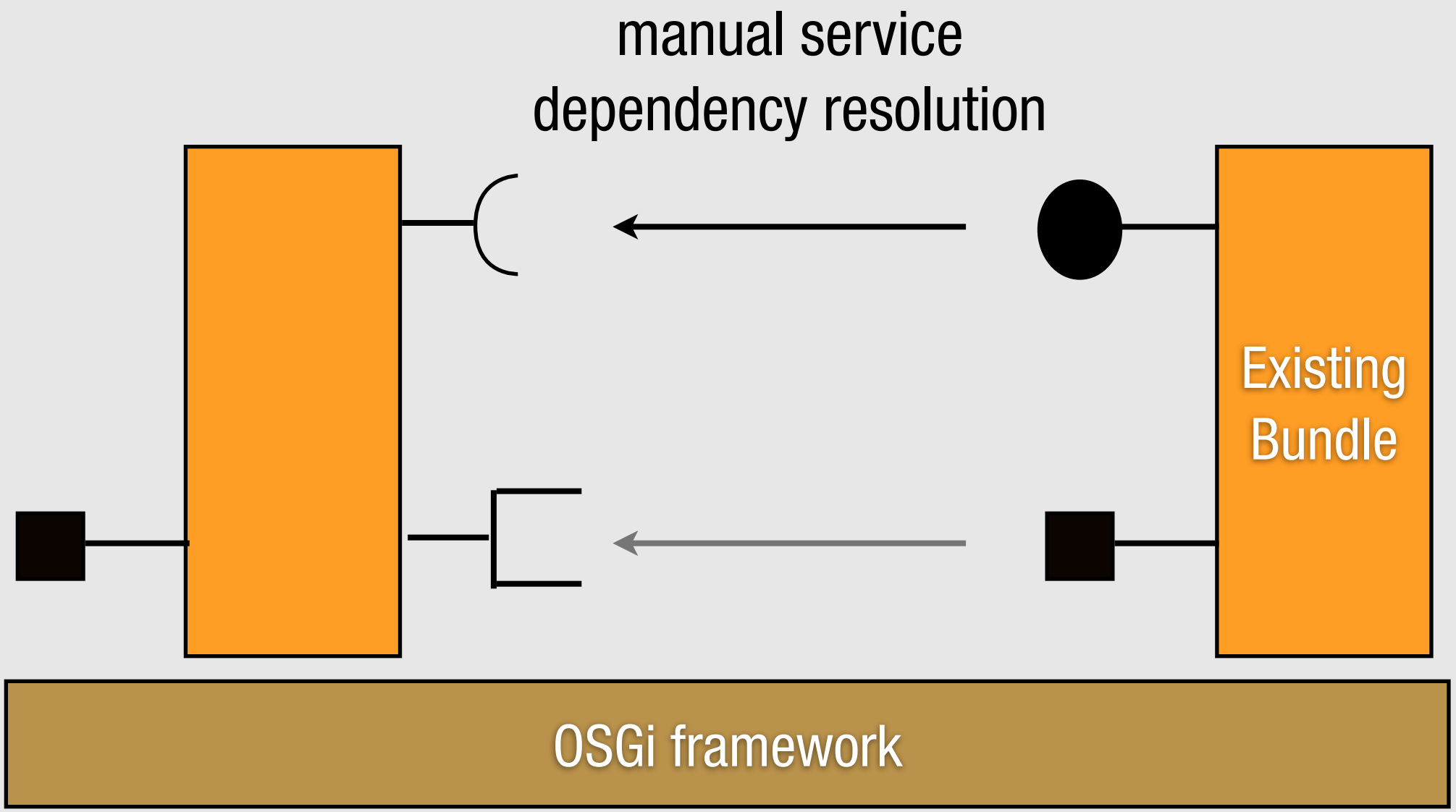
Visualization



Visualization

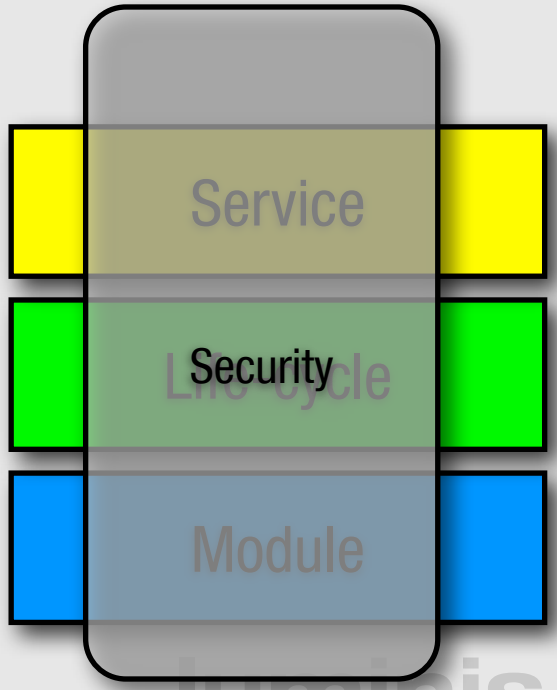


Visualization



Security

- Optional Security Layer based on Java permissions
- Infrastructure to define, deploy, and manage fine-grained application permissions
- Code authenticated by location or signer
- Well defined API to manage permissions
  - PermissionAdmin
  - ConditionalPermissionAdmin



Agenda

- History of OSGi
- The Framework
- The Compendium
- OSGi Application Approaches
- Embedding
- Managing Service Dependencies
- Development Environment
- Open Source Frameworks

Leveraging standard services

- Specification:
  - OSGi compendium – catalog of standard service descriptions
- Implementations:
  - OBR repository at bundles.osgi.org – over 1400 bundles, implement compendium and other services
  - Maven repository and third party OBR's
  - More and more projects are made OSGi compatible, for example:
    - Apache Commons OSGi

OSGi compendium

luminis

OSGi compendium

User Admin

Initial Provisioning

Wire Admin

Log

Device Access

XML Parser

Measurement and State

Preferences

UPnP™ Device

Configuration Admin

Position

Metatype

Service Tracker

Event Admin

HTTP

IO Connector

Execution Environment Spec

Declarative Services

User Admin

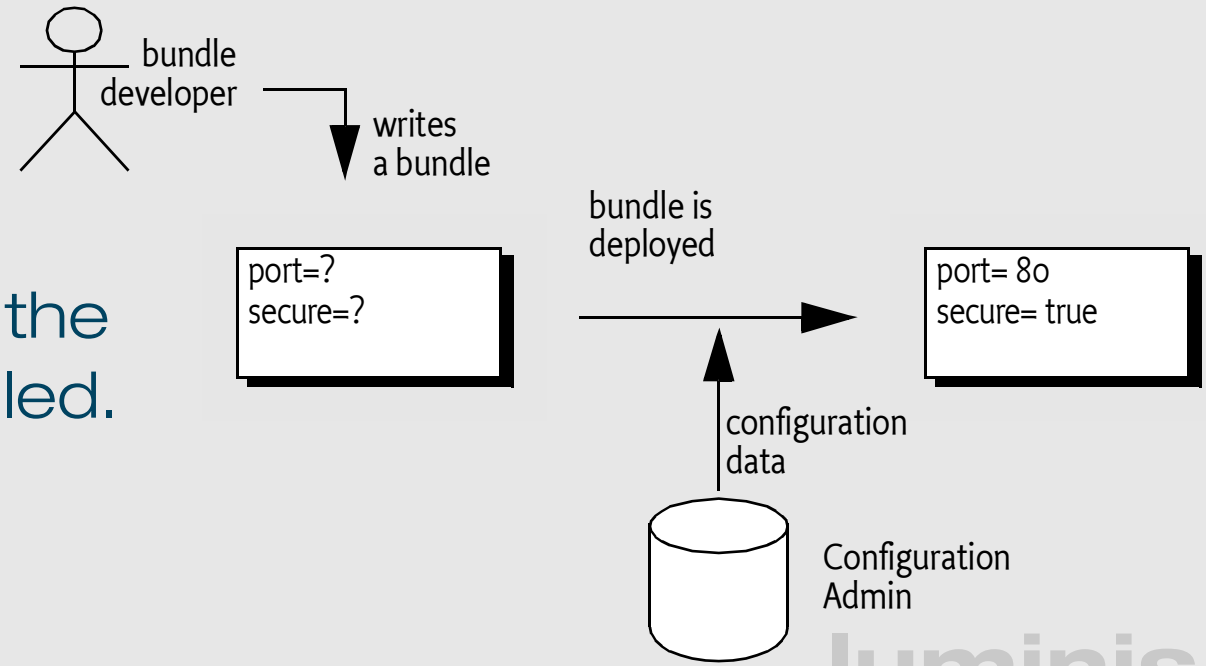
- Used in any application that needs role based access control
- Provides: users, roles and groups
- Can authenticate users
- Can determine autorization for authenticated users
- Fairly easy to plug-in to HTTP, SOAP, RMI, JMX or anything else

Config Admin

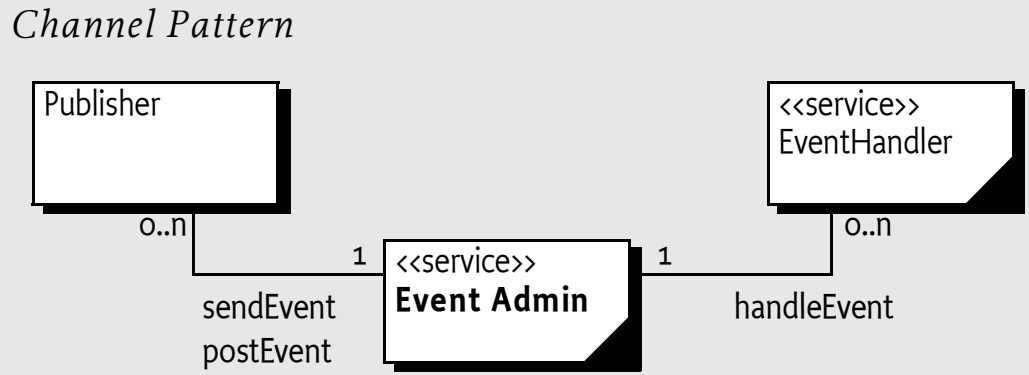
- Configuration Admin:
  - contains externally configurable settings for a service;
  - allows management systems to configure all settings;
  - settings can be created even before the actual bundle is installed.

Config Admin

- Configuration Admin:
  - contains externally configurable settings for a service;
  - allows management systems to configure all settings;
  - settings can be created even before the actual bundle is installed.



Event Admin



- Publish subscribe
- Asynchronous and synchronous
- Hierarchical topics
- Used within OSGi too

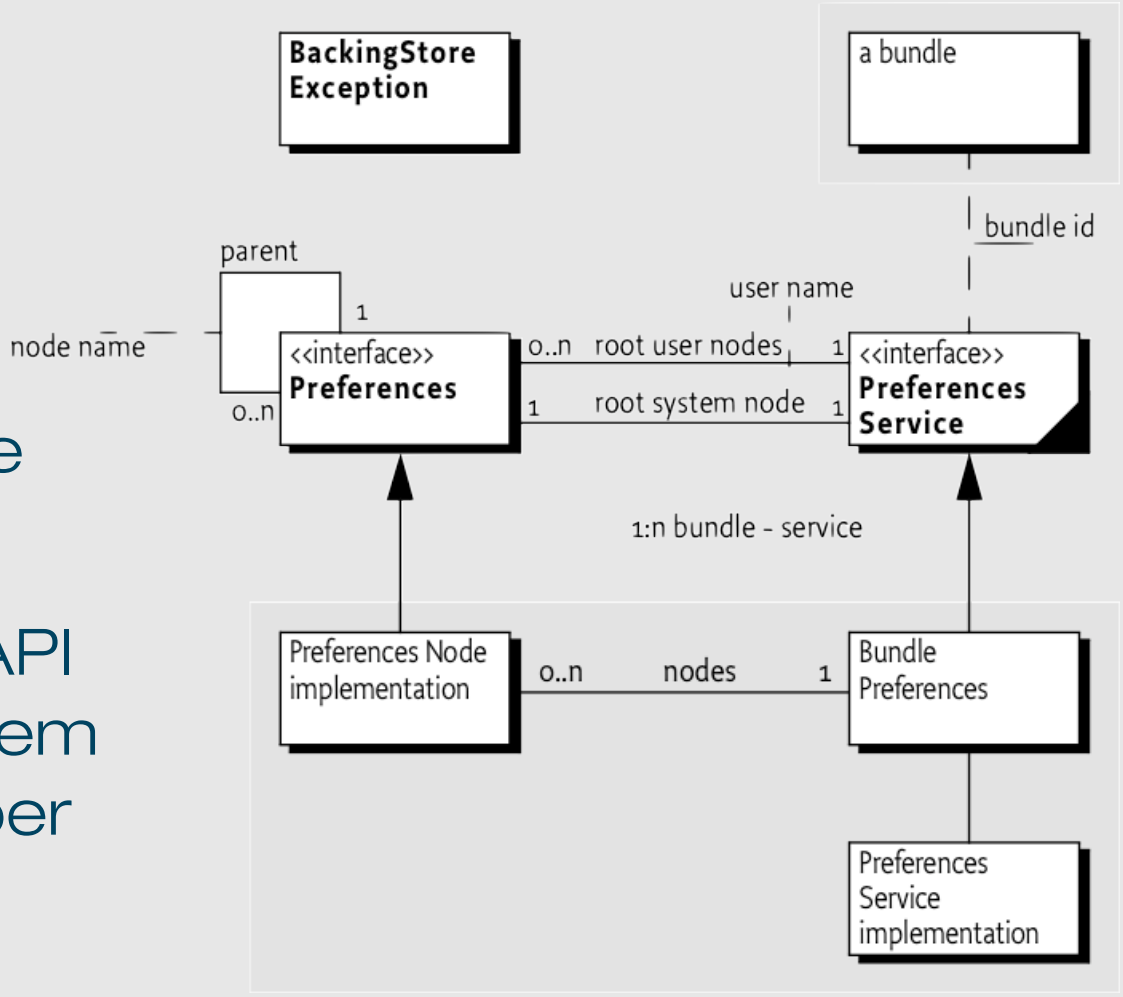
Event Admin Example

```
class Subscriber implements BundleActivator, EventHandler {
    final static String[] topics = new String[] {
        "org/osgi/service/log/LogEntry/LOG_WARNING",
        "org/osgi/service/log/LogEntry/LOG_ERROR" };

    public void start(BundleContext context) {
        Dictionary d = new Hashtable();
        d.put(EventConstants.EVENT_TOPIC, topics);
        d.put(EventConstants.EVENT_FILTER, "(bundle.symbolicName=com.acme.*)");
        context.registerService(EventHandler.class.getName(), this, d);
    }
    public void stop(BundleContext context) {
    }

    public void handleEvent(Event event) {
        //...
    }
}
```

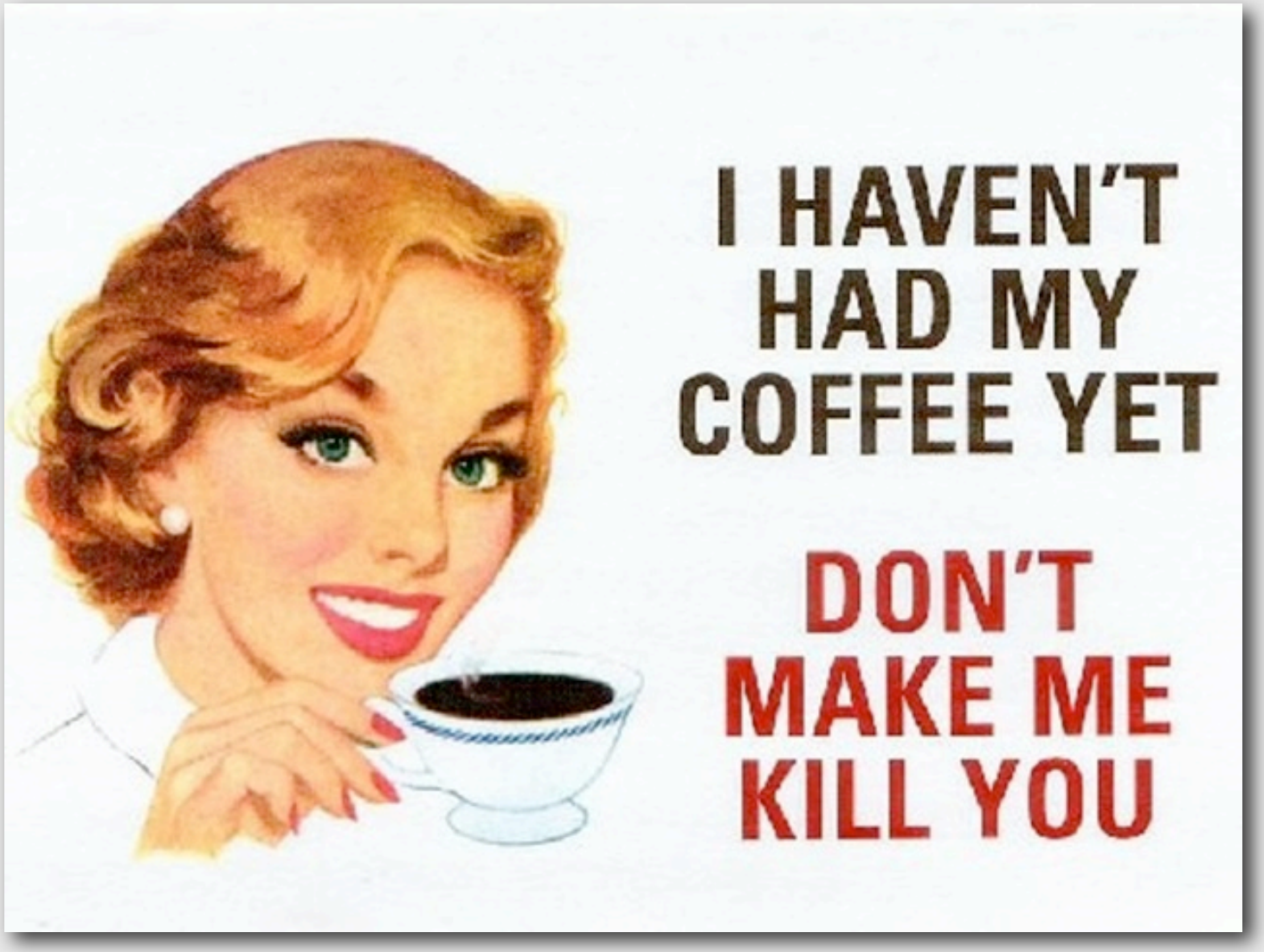
```
class Publisher {
    EventAdmin m_eventAdmin;

    public void send() {
        if (m_eventAdmin != null) {
            Dictionary properties = new Hashtable();
            properties.put("timestamp", new Date());
            m_eventAdmin.sendEvent(new Event("com/acme/timer", properties));
        }
    }
}
```

Preferences

- Preferences:
  - contains bundle private settings;
  - is coupled to the bundle life-cycle;
  - like the standard Java API there is a notion of system and user preferences per bundle.



Short break!



Agenda

- History of OSGi
- The Framework
- The Compendium
- OSGi Application Approaches
- Embedding
- Managing Service Dependencies
- Development Environment
- Open Source Frameworks

OSGi Application Approaches

- **Service model** vs. **extender model**
  - Choose an OSGi extensibility mechanism
- **Bundled application** vs. **hosted framework**
  - 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

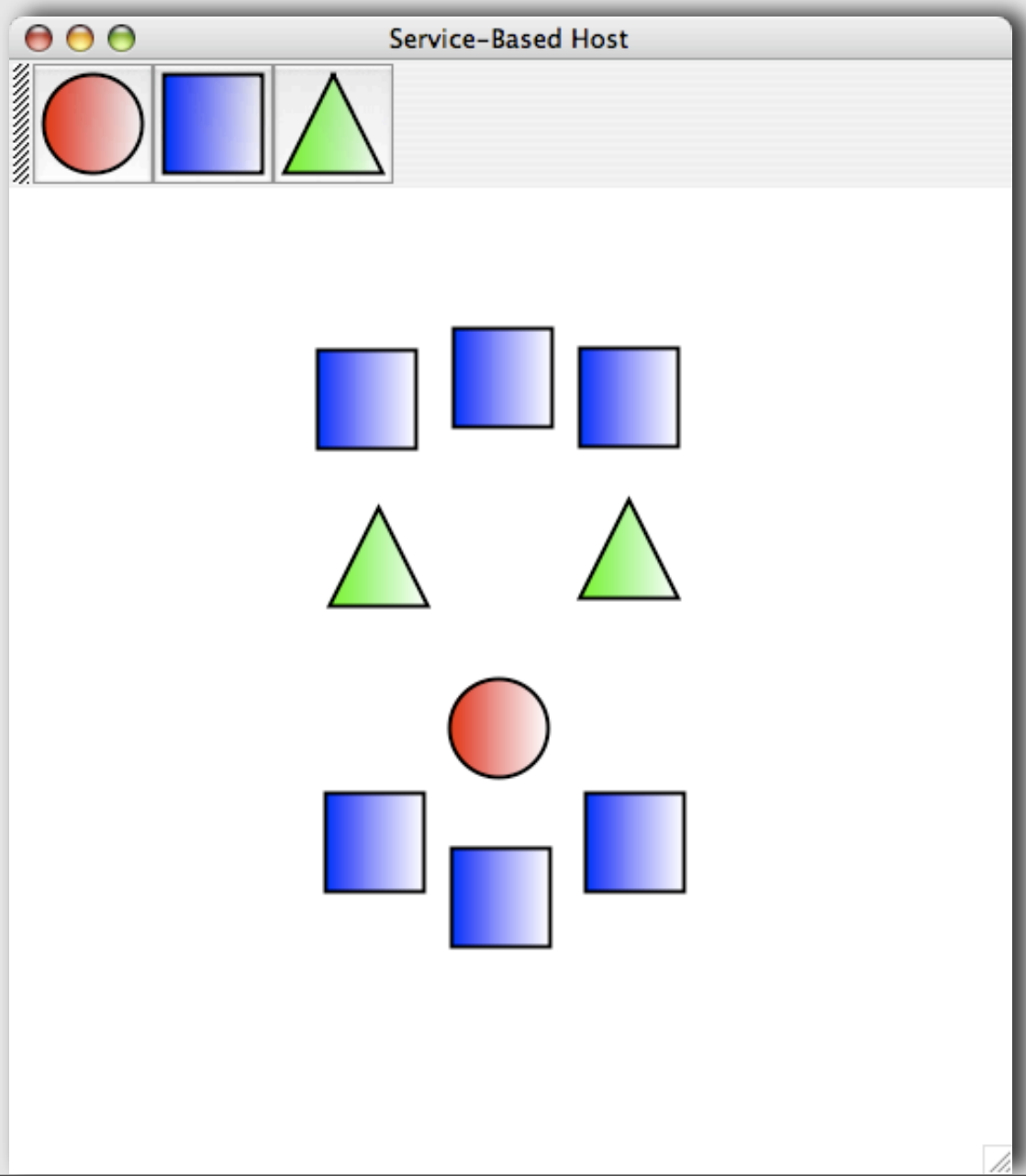
Bundled vs. Hosted

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

Example: Paint Program

- Create a simple Swing-based paint program
- Define a SimpleShape interface to draw shapes
  - Different implementations to draw different shapes
  - Each shape has name and icon properties
  - Available shapes are displayed in tool bar
- To draw a shape, click its button; then the canvas
  - Shapes can be dragged, but not resized
- Support dynamic deployment of shapes

Paint Program Mock Up

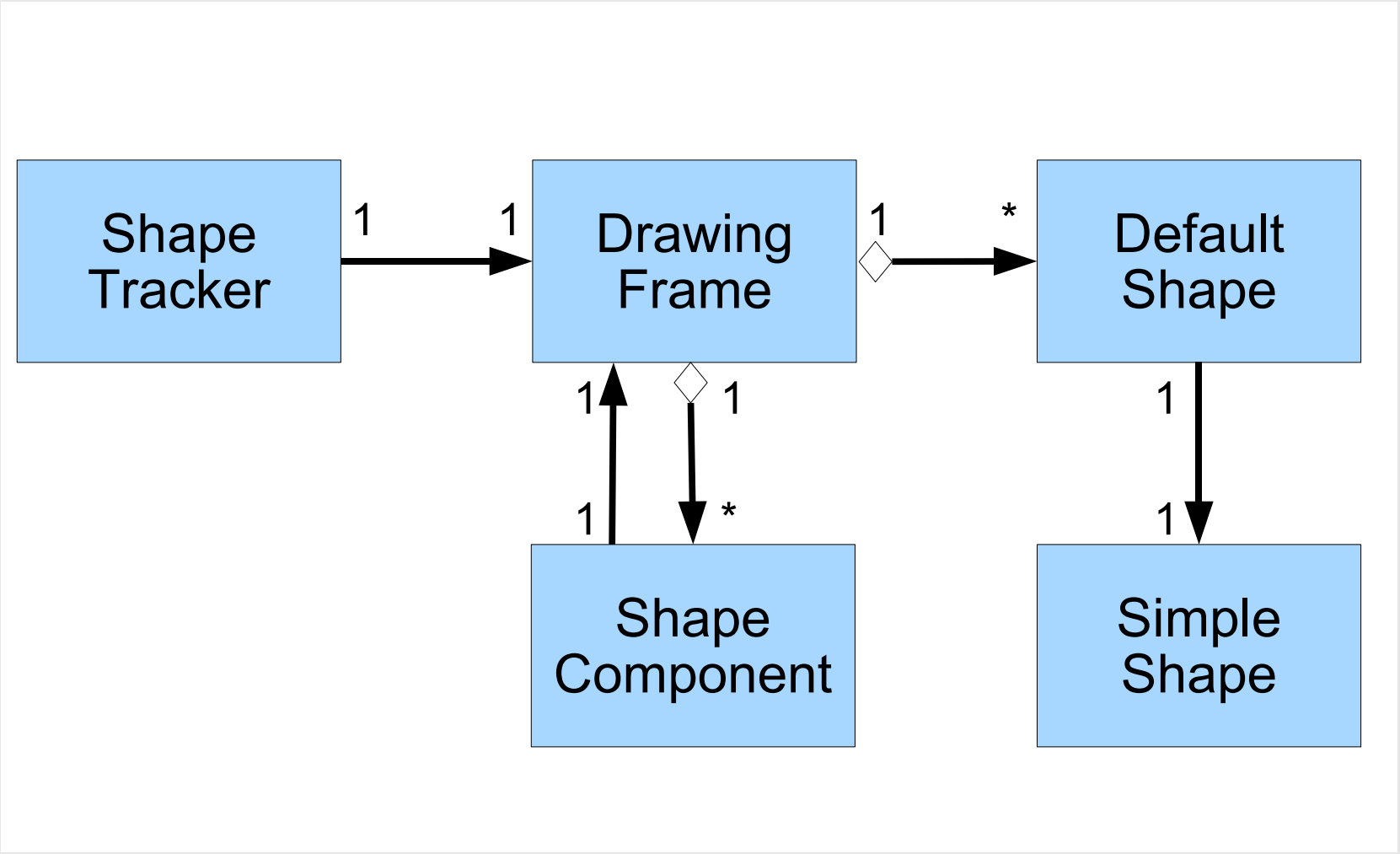


Shape Abstraction

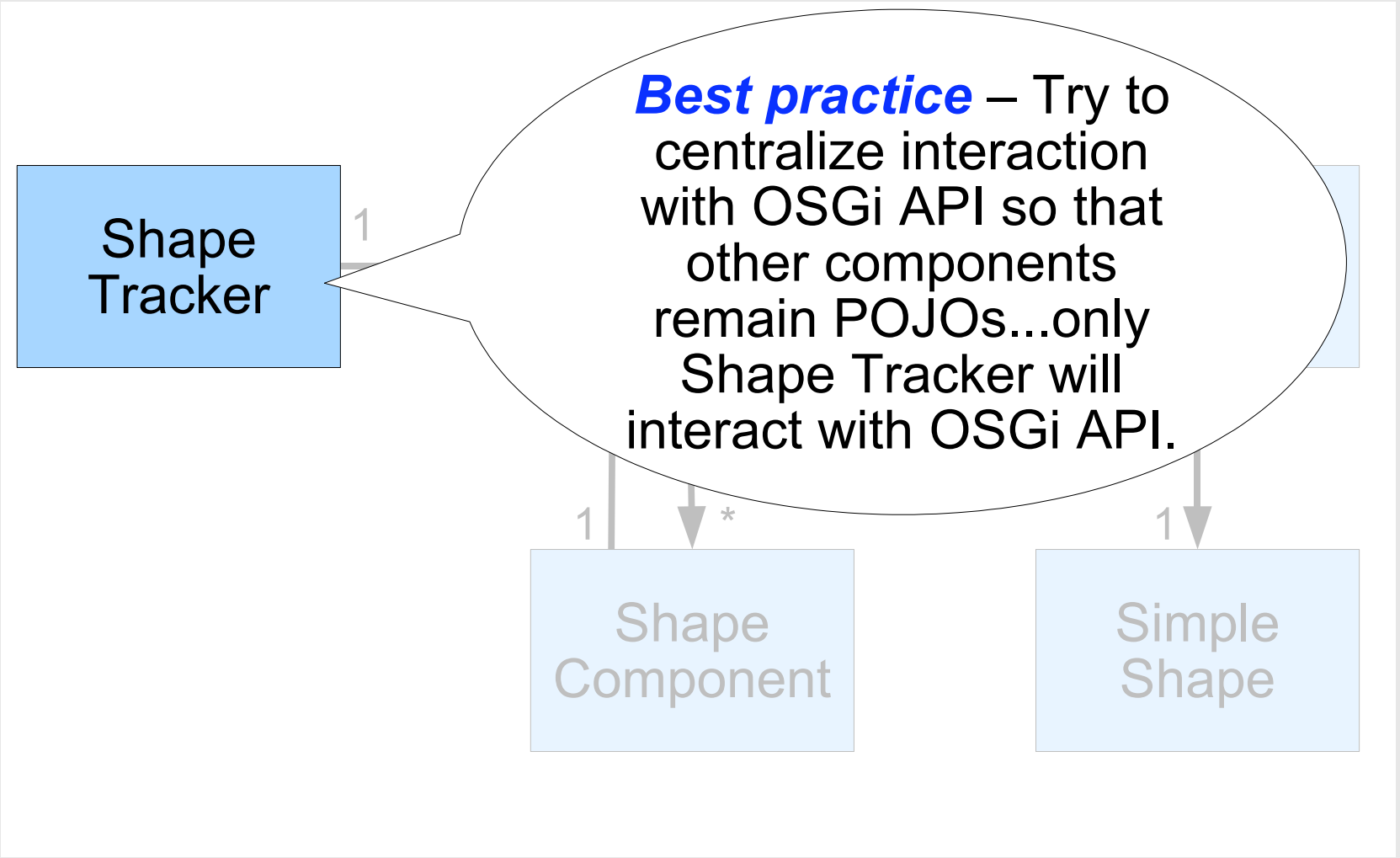
- Conceptual SimpleShape interface

```
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);
}
```

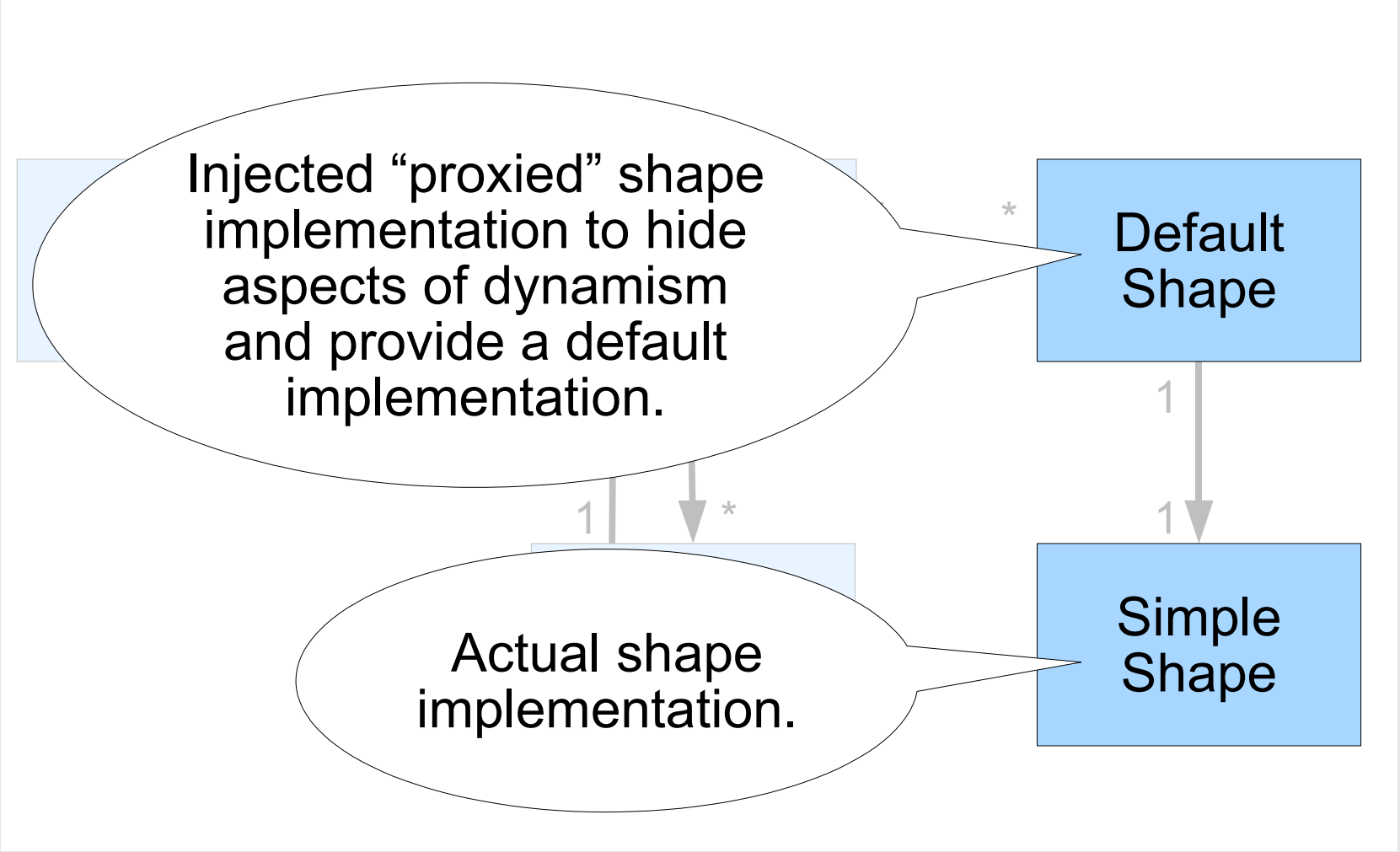
High-Level Architecture



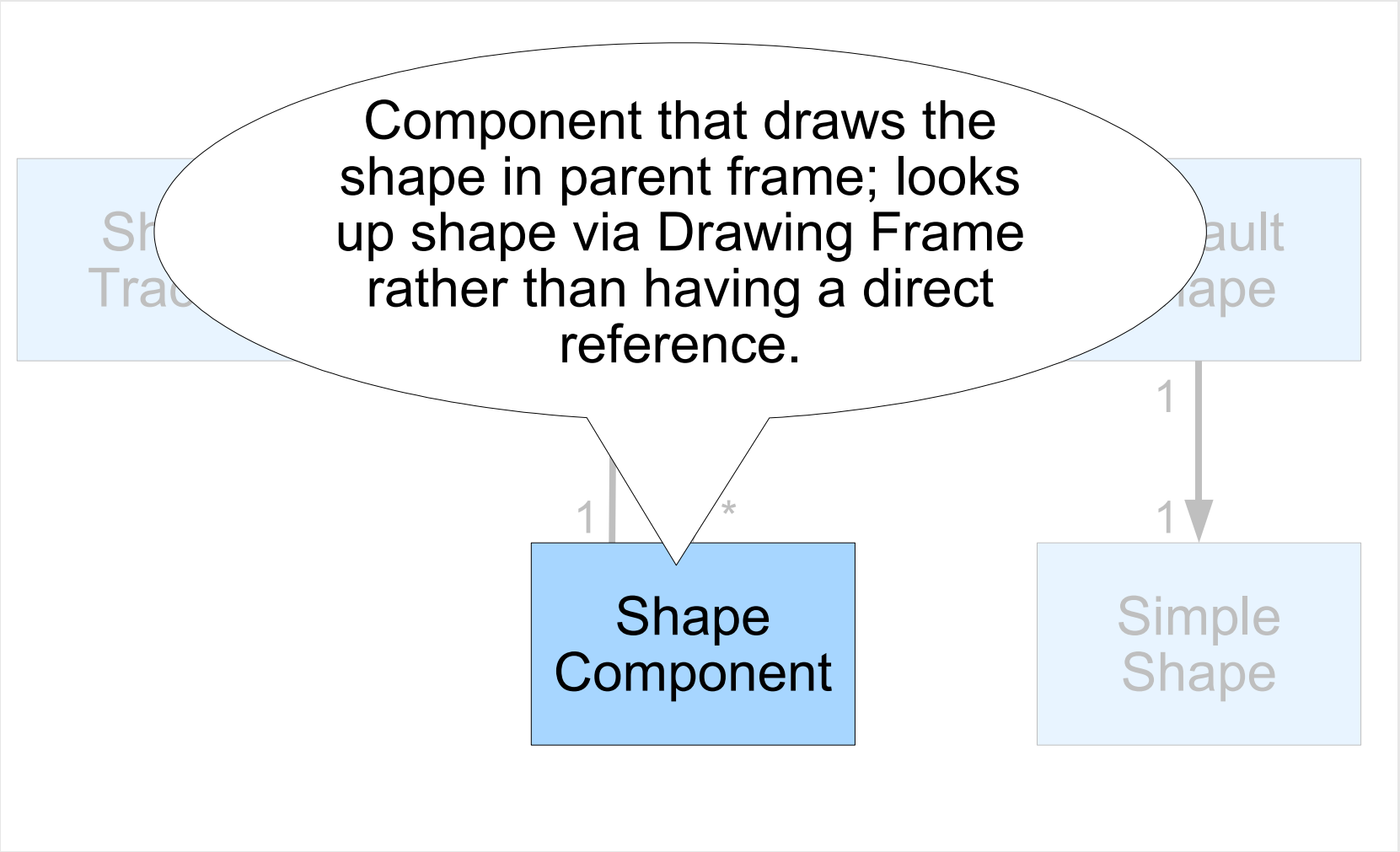
High-Level Architecture



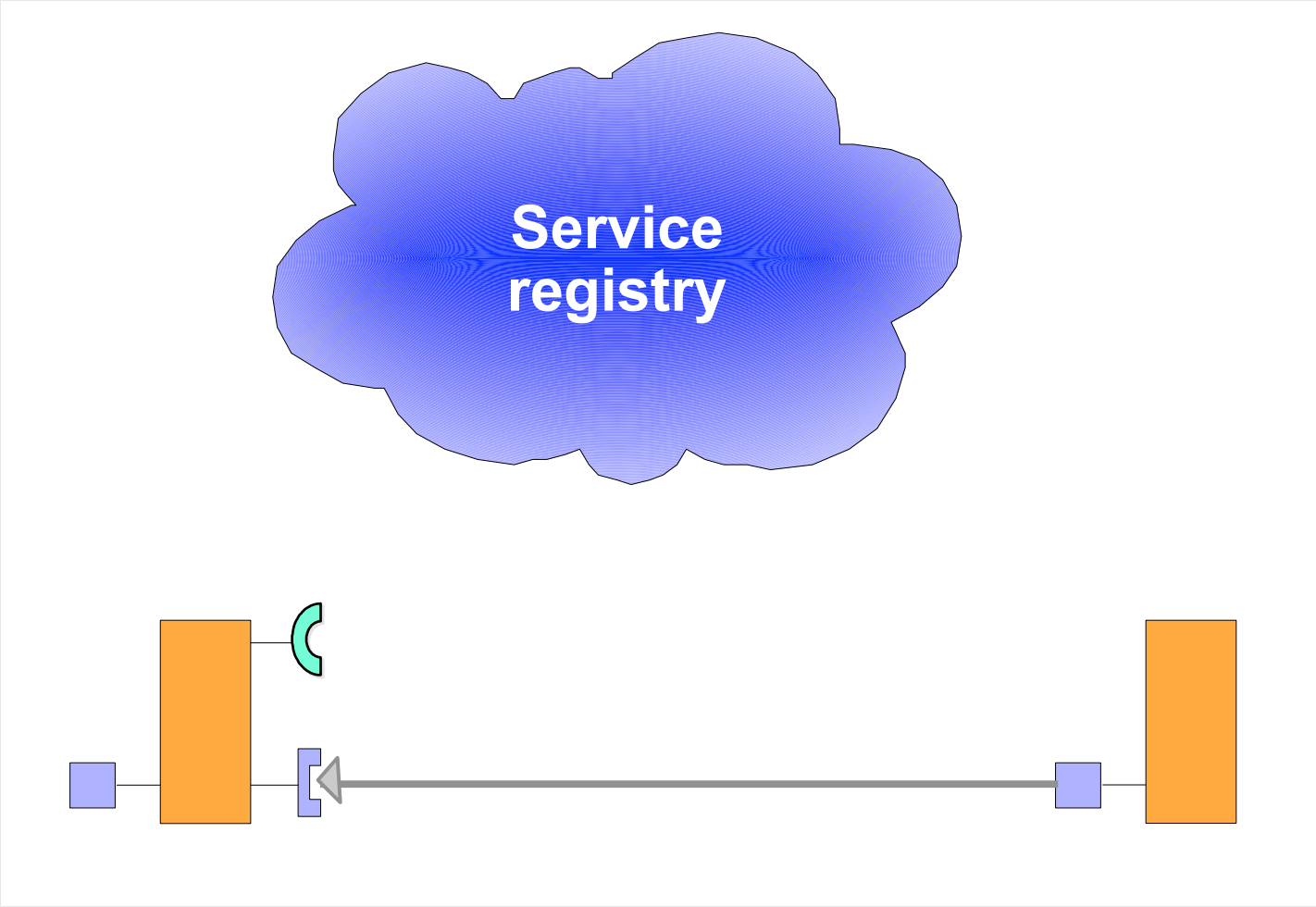
High-Level Architecture



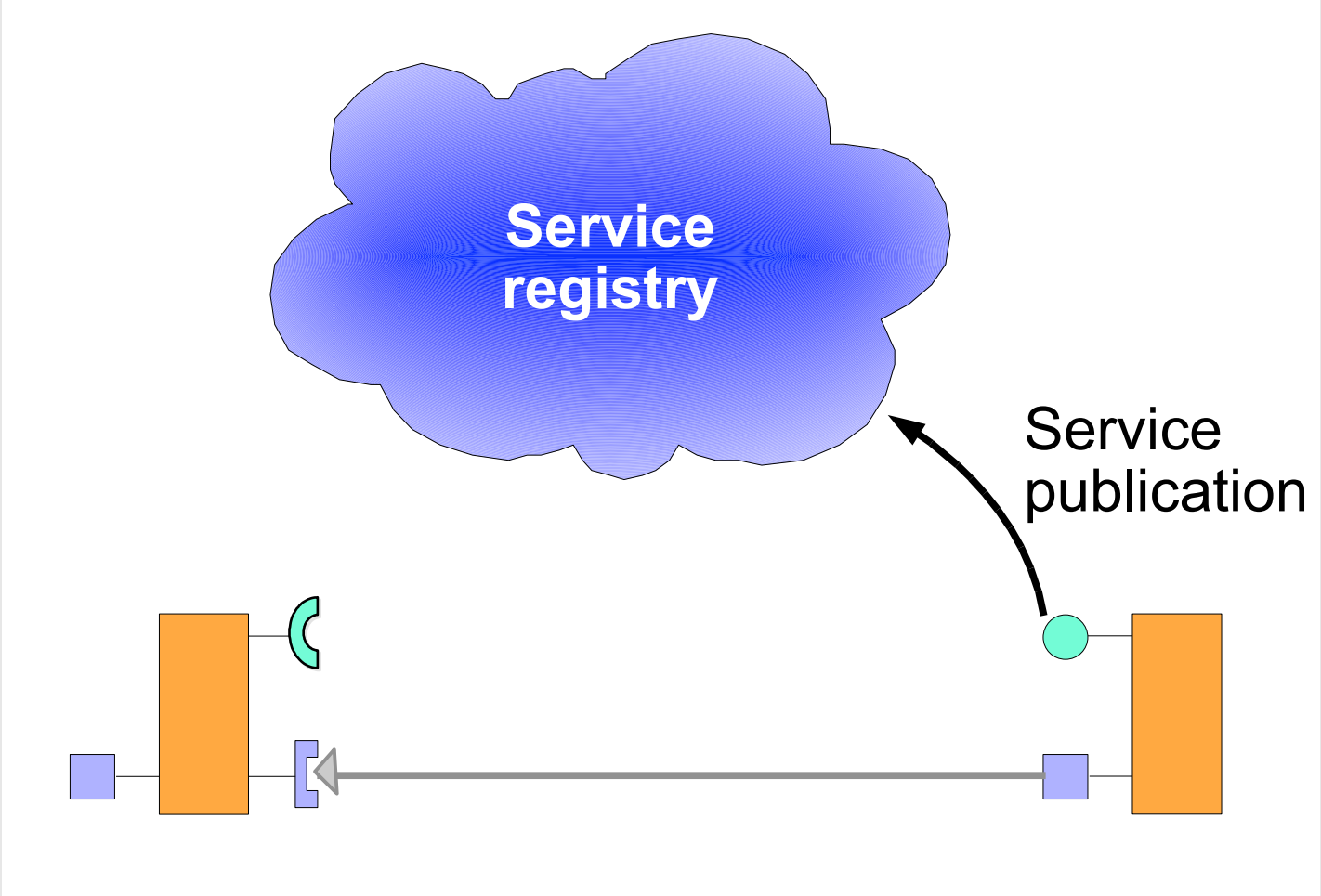
High-Level Architecture



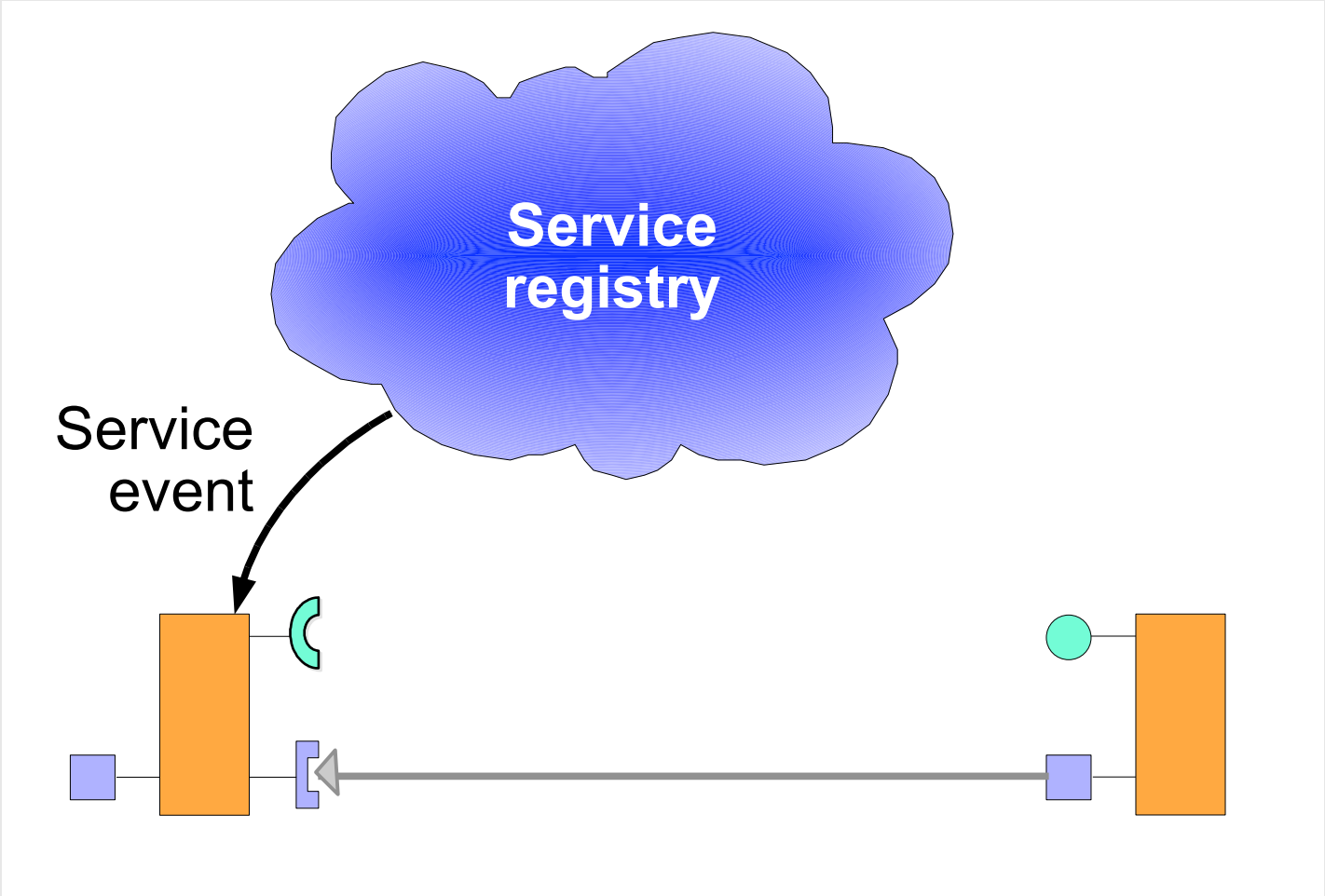
Service Model



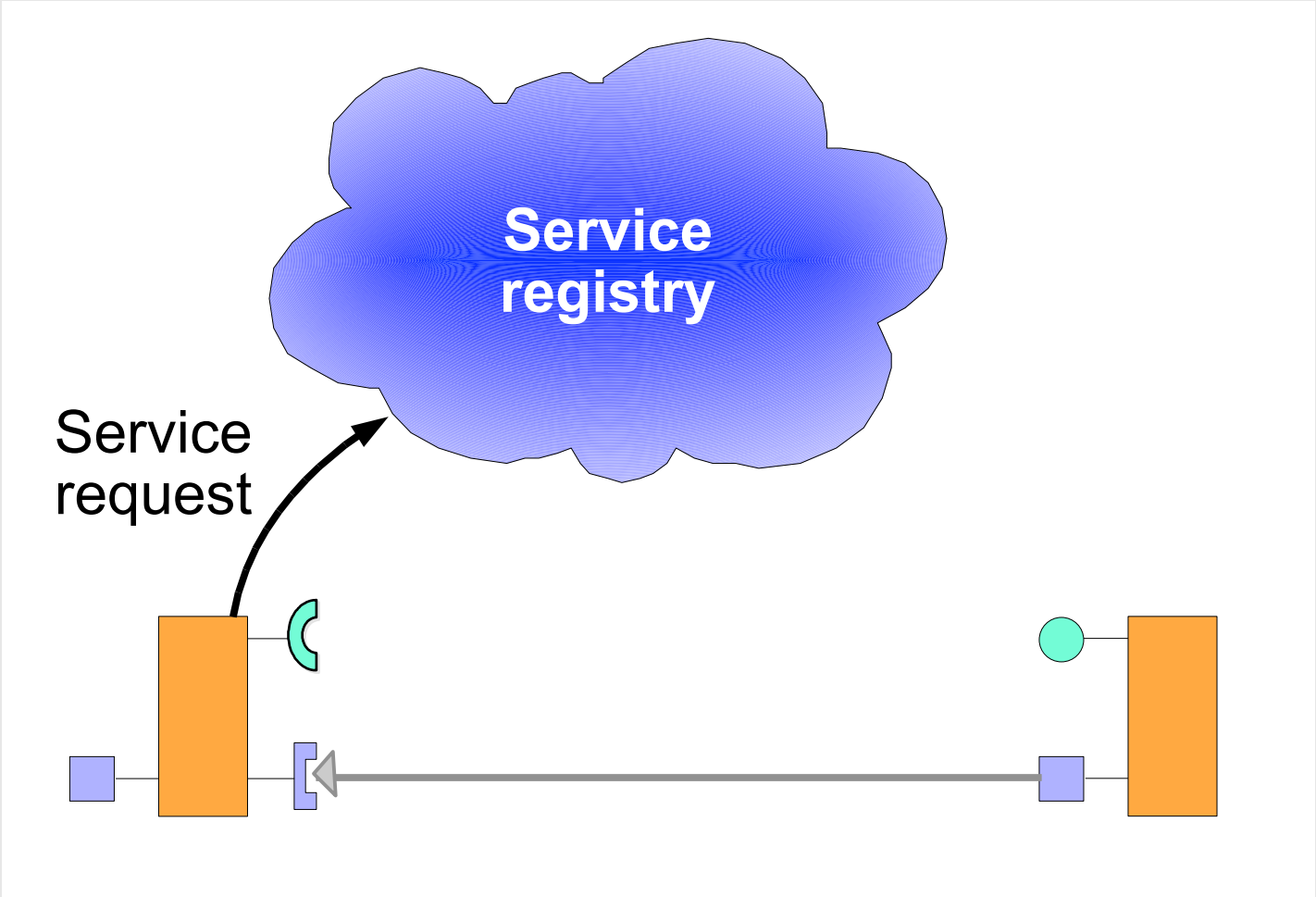
Service Model



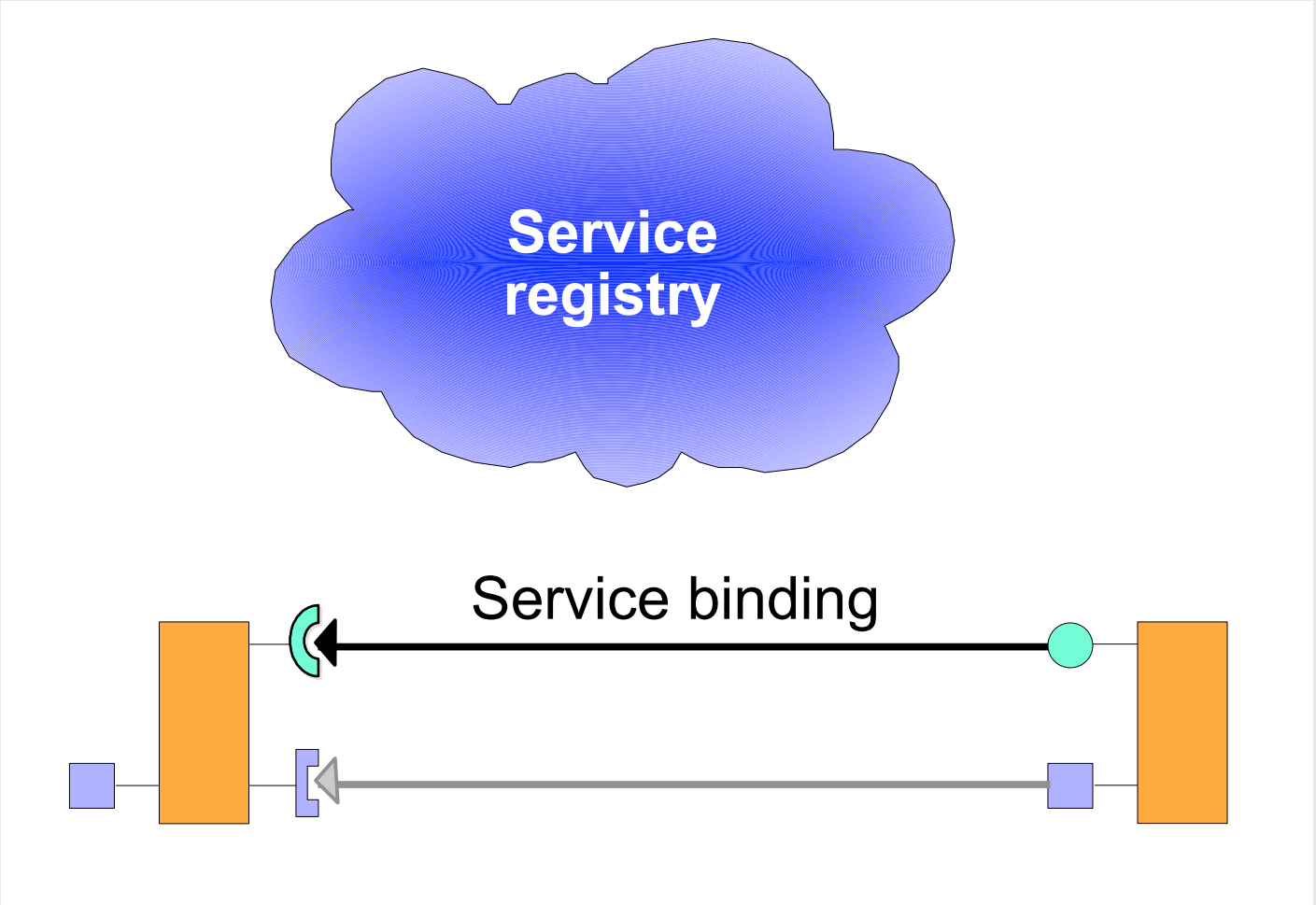
Service Model



Service Model



Service Model



Service Whiteboard Pattern

- 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
- 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.
     * @param g2 The graphics object used for painting.
     * @param p The position to paint the triangle.
    **/
    public void draw(Graphics2D g2, Point p);
}
```

Service-Based Paint Program

- Shape service bundles have an activator to register their service

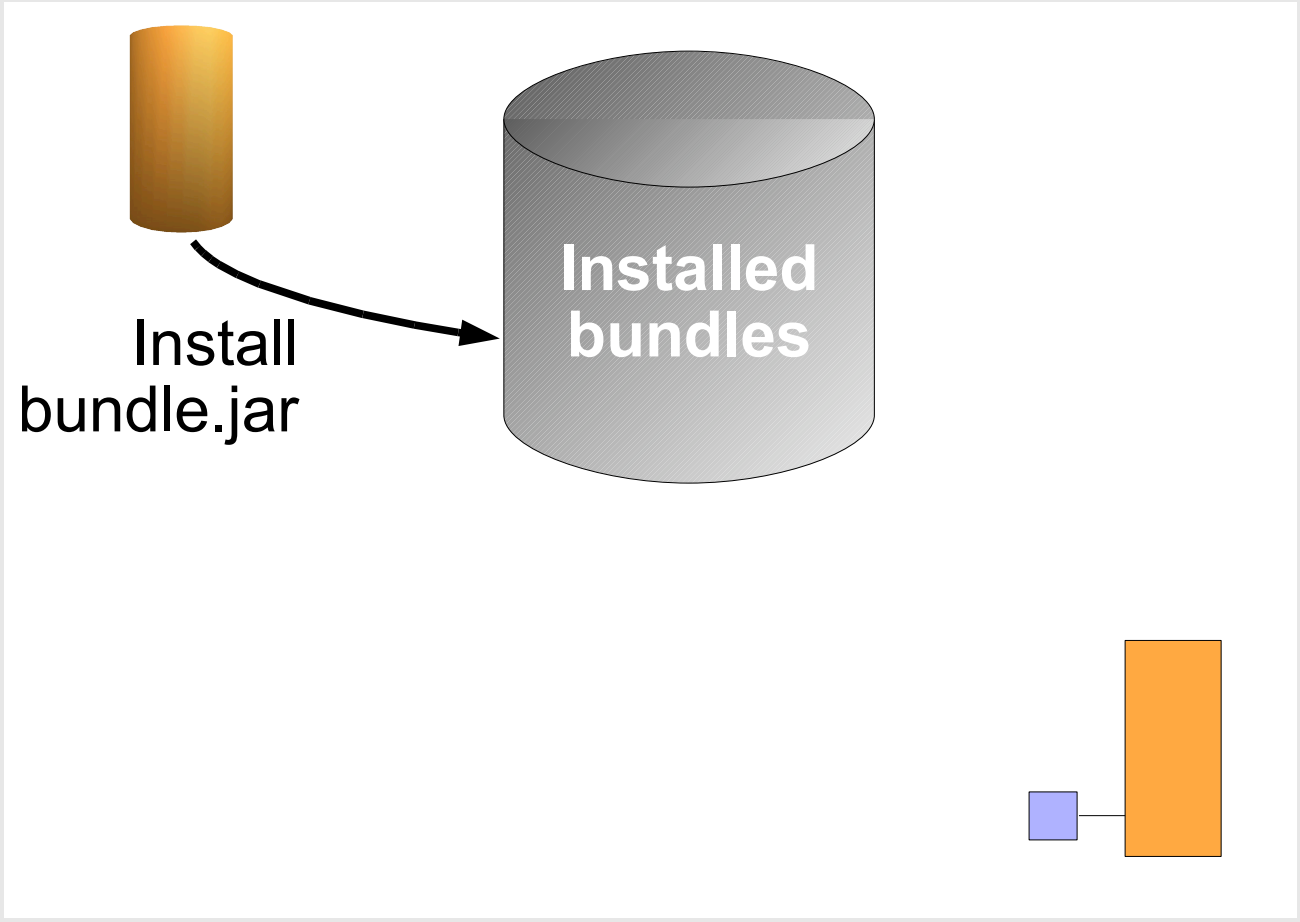
```
/**
 * Implements the <tt>BundleActivator.start()</tt> method, which
 * registers the circle <tt>SimpleShape</tt> service.
 * @param context The context for the bundle.
**/
public void start(BundleContext context)
{
    m_context = context;
    Hashtable dict = new Hashtable();
    dict.put(SimpleShape.NAME_PROPERTY, "Circle");
    dict.put(SimpleShape.ICON_PROPERTY,
        new ImageIcon(this.getClass().getResource("circle.png")));
    m_context.registerService(
        SimpleShape.class.getName(), new Circle(), dict);
}
```

Service-Based Shape Tracker

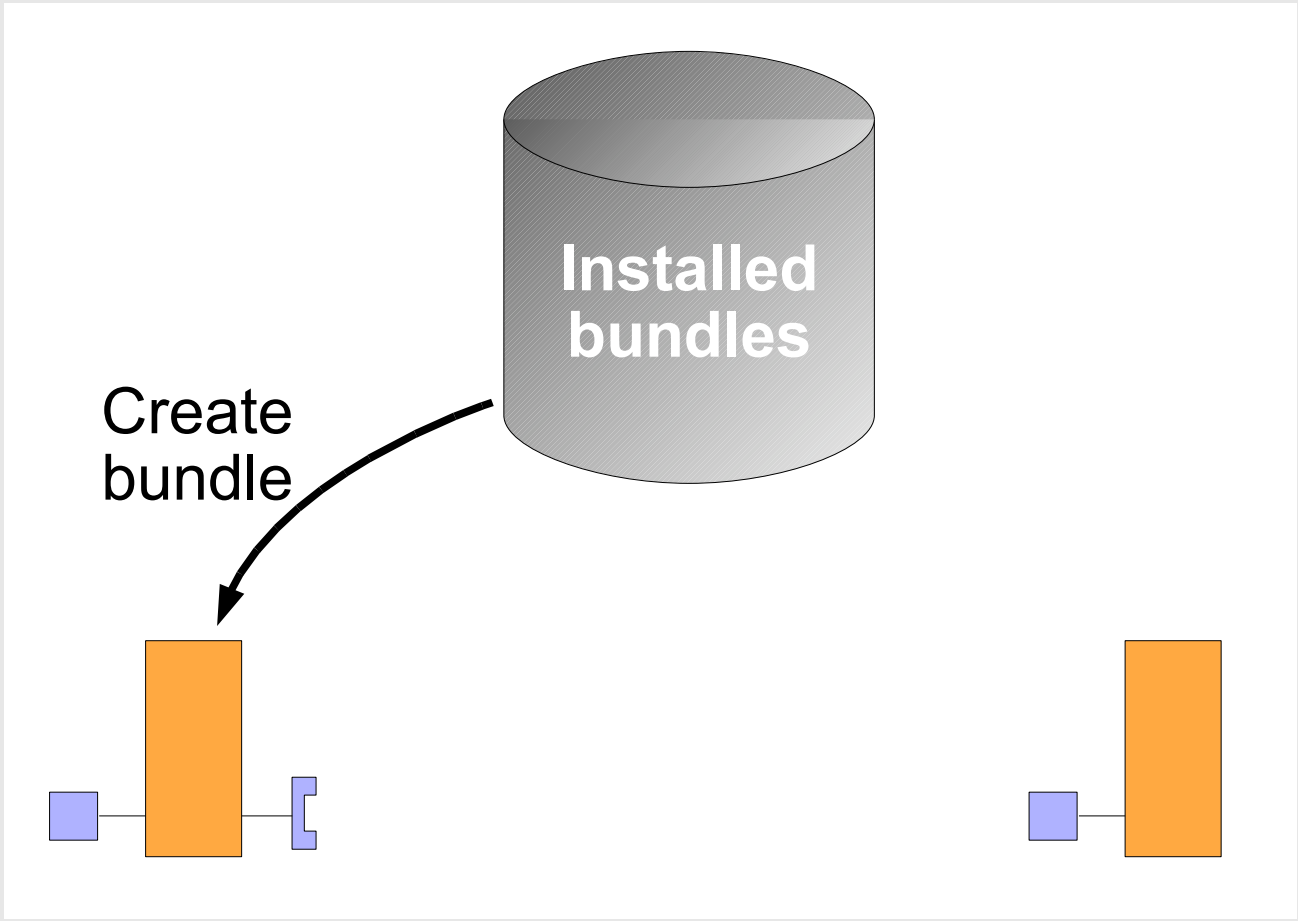
- Use Inversion of Control and inject shapes
  - Puts tracking logic in one place
  - Isolates application from OSGi API
- Implemented as OSGi Service Tracker subclass
  - Uses whiteboard pattern for services
  - Listen for SimpleShape service events

Extender Model

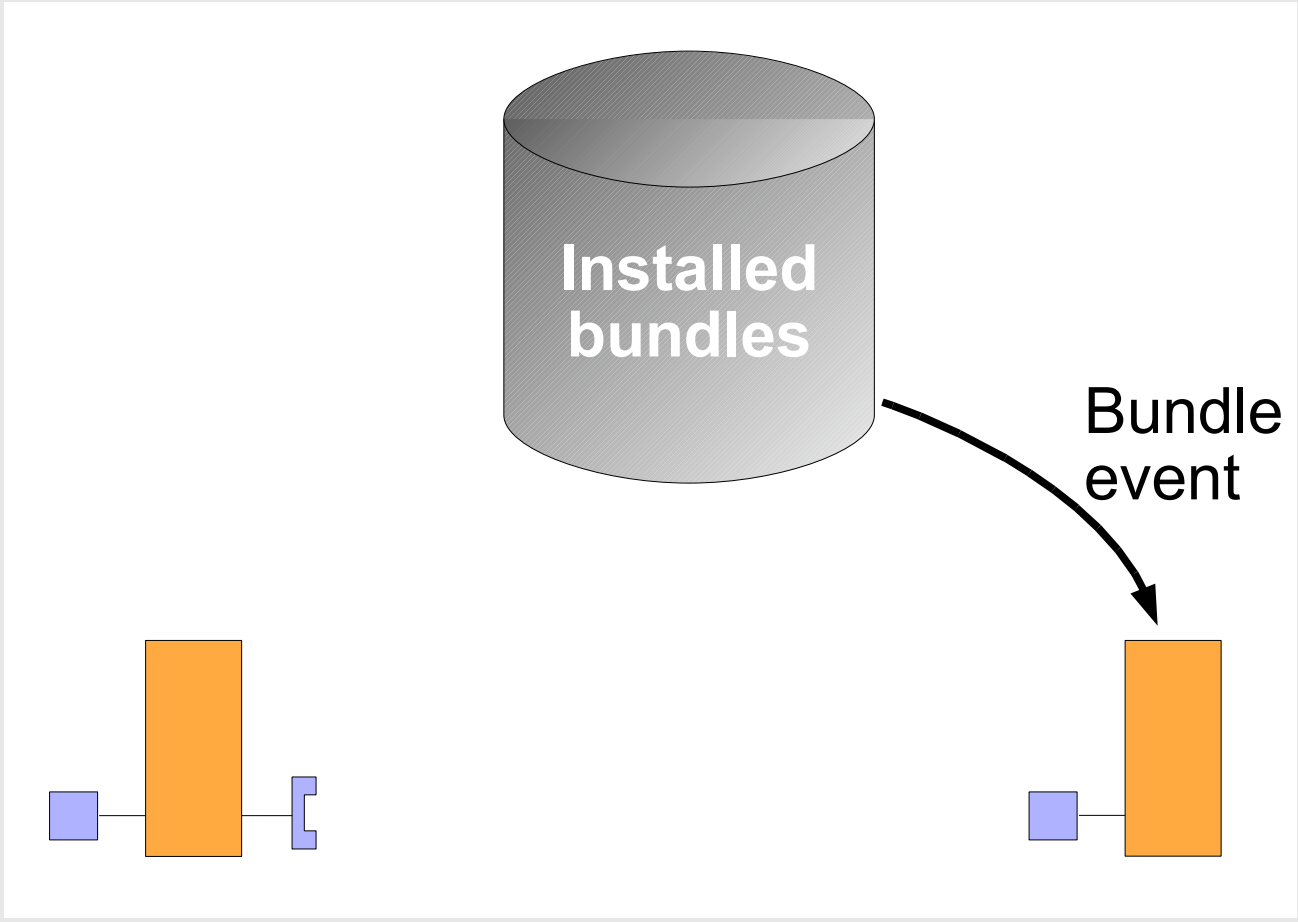
Extender Model



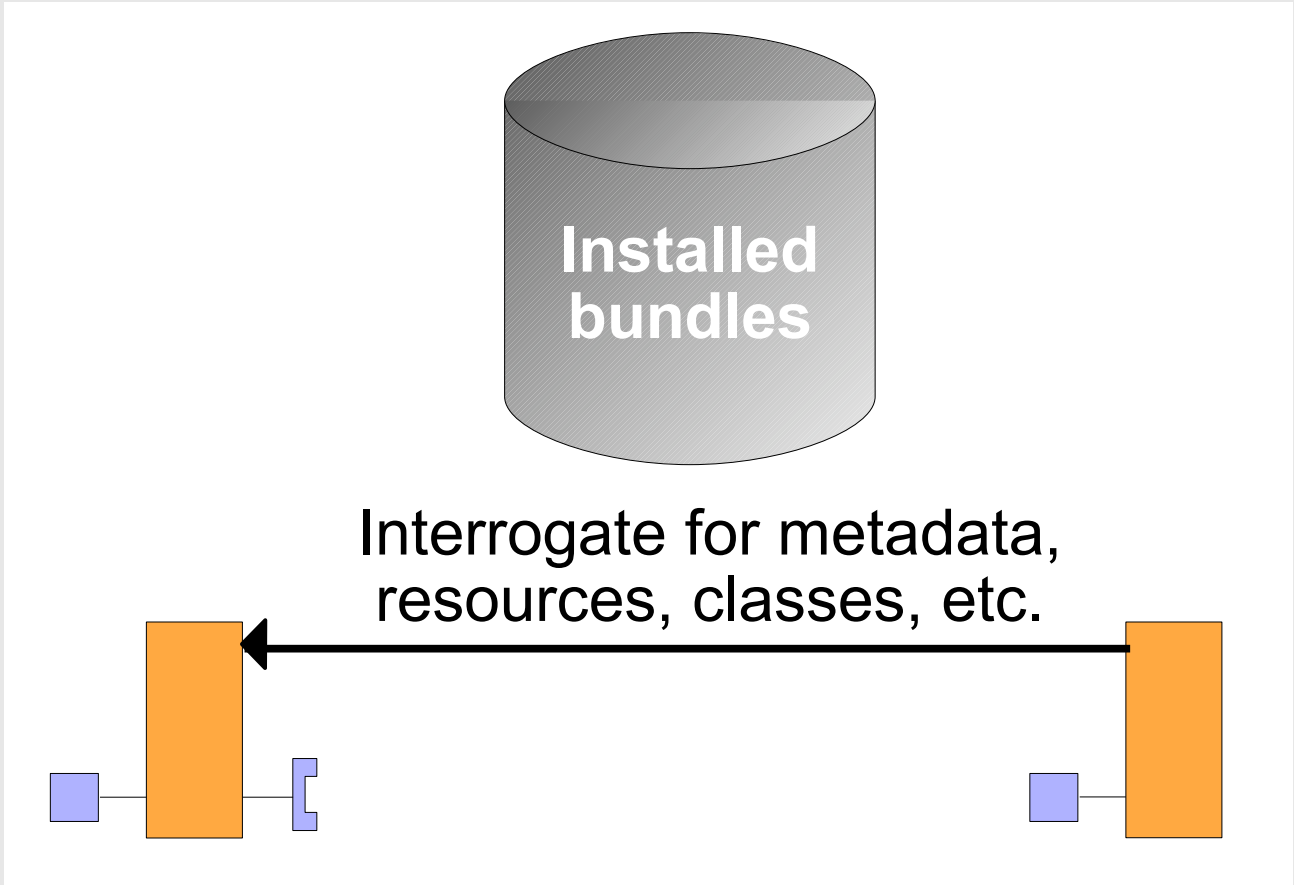
Extender Model



Extender Model



Extender Model



Extension-Based Paint

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

Extension-Based Paint

- Shape extension bundles have extension-related metadata in their Jar manifest

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

Extender-Based Tracker

- Use Inversion of Control and inject shapes
  - Puts tracking logic in one place
  - Isolates application from OSGi API
- Implemented as custom „bundle tracker“
  - Uses pattern similar to whiteboard, but for installed bundles instead of services
  - Listens for bundle events
    - Probes bundle manifest to see if shape extensions provided

Packaging the Paint Program

- Implementations are packaged in a similar fashion
  - As a bundle JAR file with metadata
- Separate public API into separate packages
  - org.apache.felix.examples.servicebased.host.service
- Only export public API packages in your metadata

Show example!

Agenda

- History of OSGi
- The Framework
- The Compendium
- OSGi Application Approaches
- Embedding
- Managing Service Dependencies
- Development Environment
- Open Source Frameworks

Hosted Framework

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

Hosted Framework

- Felix tries to simplify hosted framework scenarios
- Configuration data is passed into framework constructor
- Felix framework is 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

```
// Create a list for custom framework activators and
// add an instance of the auto-activator it for processing
// auto-install and auto-start properties.
List list = new ArrayList();
list.add(new AutoActivator());

// Create a case-insensitive property map.
Map configMap = new StringMap(false);

try
{
    // Create an instance of the framework.
    Felix felix = new Felix(configMap, list);

    // Start the framework instance
    felix.start();
    ...
    // Stop the framework instance
    felix.stop();
}
catch (Exception ex) { ... }
```

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

- 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

Agenda

- History of OSGi
- The Framework
- The Compendium
- OSGi Application Approaches
- Embedding
- Managing Service Dependencies
- Development Environment
- Open Source Frameworks

Managing Dependencies

- Declarative Services
- Dependency Manager
- iPOJO
- Spring

Declarative Services

- Service Component Runtime, part of the spec
- Declared in a header in the bundle manifest
- XML descriptor for dependencies
- Maven SCR plugin
- Felix shell command for managing SCR bundles

Declarative Services Example

```
public class SampleComparator implements Comparator {
    private volatile LogService m_log;
    public int compare(Object o1, Object o2) {
        return o1.equals(o2) ? 0 : -1;
    }

    protected void activate(ComponentContext context) {
        LogService log = m_log;
        if (log != null) {
            log.log(LogService.LOG_INFO, "Hello Components!");
        }
    }
    protected void deactivate(ComponentContext context) {
    }

    protected void bindLog(LogService log) {
        m_log = log;
    }
    protected void unbindLog(LogService log) {
        m_log = null;
    }
}
```

```
<component name="sample.component" immediate="true">
  <implementation class="sample.SampleComparator" />
  <property name="service.description" value="Comparator" />
  <property name="service.vendor" value="ASF" />
  <service>
    <provide interface="java.util.Comparator" />
  </service>
  <reference name="log"
         interface="org.osgi.service.log.LogService"
         cardinality="0..1" policy="dynamic"
         bind="bindLog" unbind="unbindLog" />
</component>
```

Dependency Manager

- API based dependency management
- supports extensible types of dependencies:
  - service dependency
  - configuration dependency
- change dependencies dynamically at runtime

Dependency Manager Example

```
public class SampleComparator implements Comparator {
    private volatile LogService m_log;

    public int compare(Object o1, Object o2) {
        return o1.equals(o2) ? 0 : -1;
    }

    void start() {
        m_log.log(LogService.LOG_INFO, "Hello there!");
    }
}
```

```
public class Activator extends DependencyActivatorBase {
    public void init(BundleContext context, DependencyManager manager) throws Exception {
        manager.createService()
            .setInterface(Comparator.class.getName(), null)
            .setImplementation(SampleComparator.class)
            .add(createServiceDependency()
                .setService(LogService.class)
                .setRequired(false));
    }

    public void destroy(BundleContext context, DependencyManager manager) throws Exception {
    }
}
```

iPOJO

- iPOJO is an evolution of Service Binder (which inspired Declarative Services).
- Further simplify the OSGi programming model
  - Byte code inspection and instrumentation is used to simplify metadata
- Implement a composite component concept
  - Further embrace of the Factory concept
  - A composite has two levels: what is on the outside and what is on the inside.

iPOJO

- Component definitions do not automatically create instances
  - purely define a component type that is reified as „factory“ service in the service registry
  - To get an instance it is necessary to create them
    - programmatically or via metadata
- Conceptually, a composite is a service registry nested inside of a parent service registry.
  - Global OSGi service registry at the root
- Furthermore, a composite can offer and require services from the outside

iPOJO

- Component that provides a „org.foo.TextEditor“ service
  - 0-to-n dynamic dependency on „org.foo.Plugin“ services
  - field=“plugins“ refers to a field in the component class where Plugin services will be injected

```
<component classname=“org.foo.MyEditor“>
    <provides/>
    <requires field=“plugins“/>
</component>
<instance component=“org.foo.MyEditor“/>
```

iPOJO

- Composite defines a text editor service that is tailored to editing Java text files
  - singular text editor subservice
  - aggregate subservice for all Java-related plugins

```
<composite name="org.foo.JavaEditorFactory">
  <subservice action="instantiate" specification="org.foo.Plugin"
    filter="(mime.type=text/java)" aggregate="true"/>
  <subservice action="instantiate" specification="org.foo.TextEditor"
    binding-policy="static"/>
</composite>
<instance component="org.foo.JavaEditorFactory"/>
```

Spring OSGi

- The Spring Dynamic Modules for OSGi(tm) Service Platforms project makes it easy to build Spring applications that run in an OSGi framework
- Exposing beans as OSGi services is fairly simple
- OSGi services can be used as well
  - Services are injected using a proxy
  - Dynamism is limited because the proxy will stay regardless of an available service
  - Throws ServiceUnavailableException

Spring OSGi

- Exposing beans as OSGi services

```
<bean name="reverseBean"
      class="nl.luminis.demo.reversestring.ReverseStringImpl"/>
 <osgi:service ref="reverseBean"
              interface="nl.luminis.demo.string.ReverseString"/>
```

- Using OSGi services inside of beans

```
<bean name="reverseBean"
      class="nl.luminis.demo.reversestring.ReverseStringBean">
  <property name="auditService" ref="externalAuditService"/>
</bean>
```

Spring OSGi

- Dynamic services

```
<osgi:reference id="externalAuditService"
                interface="nl.luminis.demo.audit.AuditService"
                cardinality="0..1">
  <osgi:listener ref="reverseBean"
                 bind-method="serviceAdded"
                 unbind-method="serviceRemoved"/>
</osgi:reference>
```

Agenda

- History of OSGi
- The Framework
- The Compendium
- OSGi Application Approaches
- Embedding
- Managing Service Dependencies
- Development Environment
- Open Source Frameworks

Development

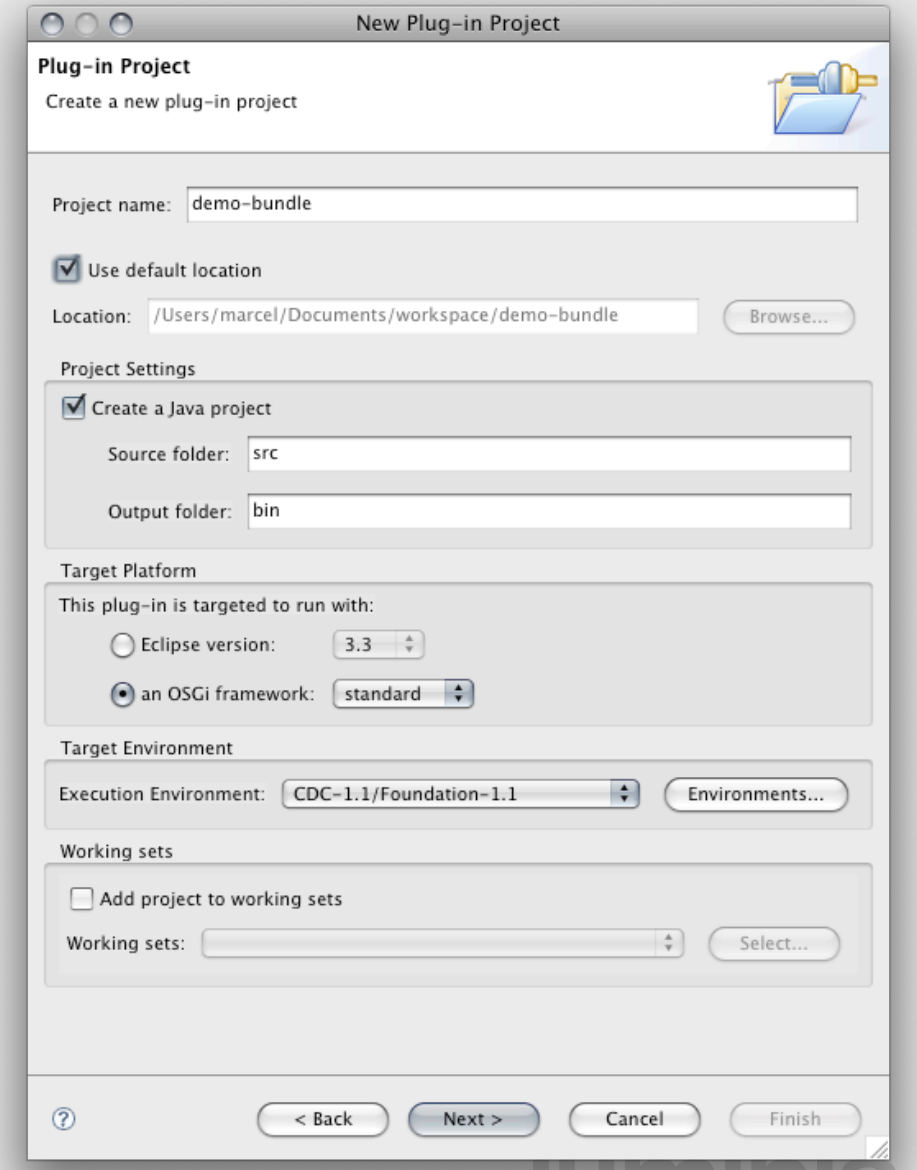
- Eclipse
- Build systems
  - Maven
  - Ant

Eclipse

PLUG-IN == BUNDLE

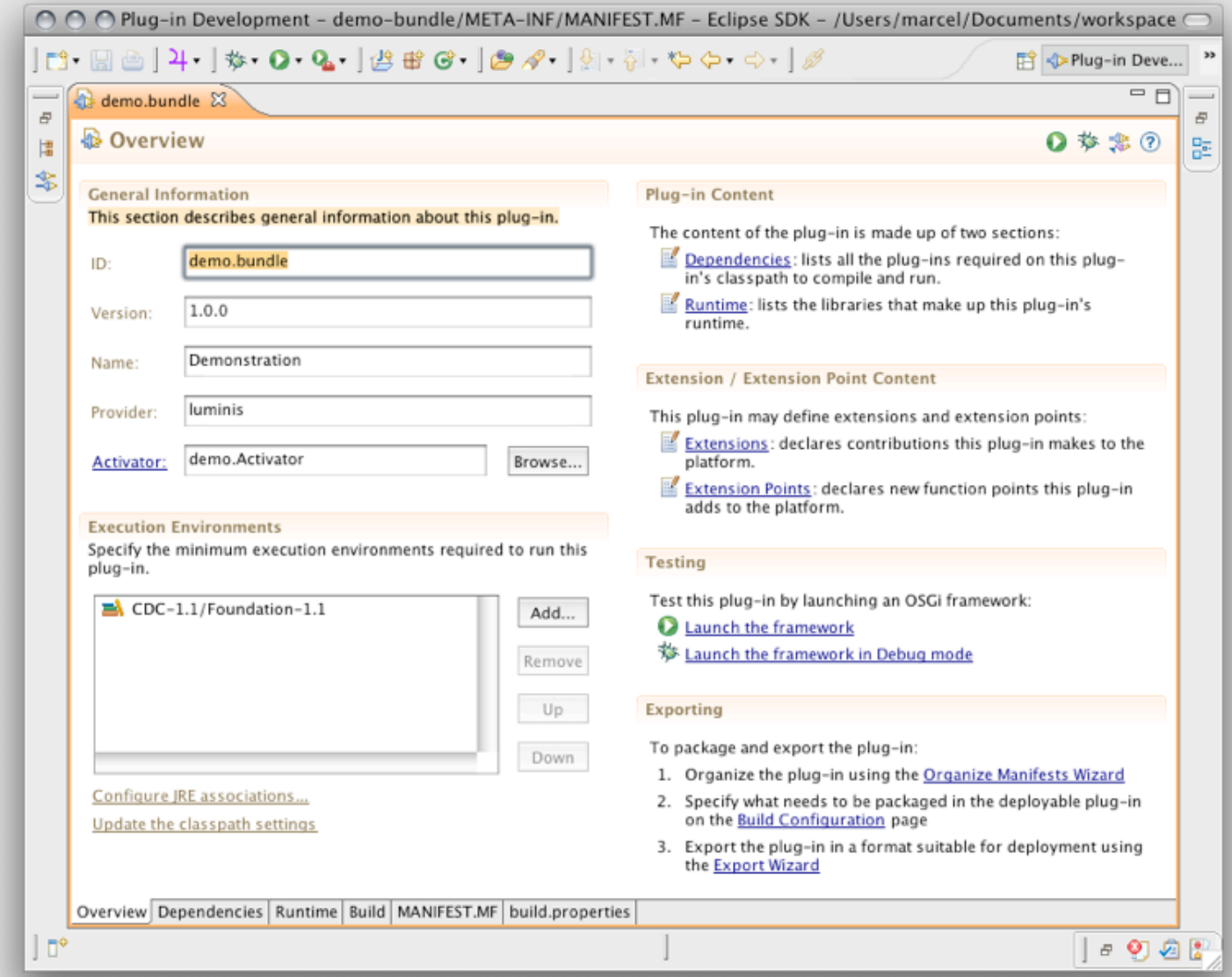
Eclipse

- Create plug-in project for each bundle
- Make sure to select “standard” OSGi framework
- Select your Execution Environment



Eclipse

- Manage dependencies
- Run inside Eclipse
- Export to bundle file



Build System

- Eclipse
  - harder to run outside of Eclipse
  - one bundle per project
- Maven
  - used and developed within Felix
  - one bundle per “project” model
- Ant
  - use a Bnd based bundle task
  - more flexible project models possible

Maven

- Two plugins in Felix:
  - Bnd based bundle plugin
    - Supports publishing to an OBR
    - Has Eclipse/PDE integration
  - SCR plugin
- Pax Eclipse plugin at OPS4J
  - synchronizes Maven and Eclipse projects

Ant

- More flexibility and control
- Use macros
- OSGi Bundle Ant Task

  https://opensource.luminis.net/confluence/x/AgAX

Agenda

- History of OSGi
- The Framework
- The Compendium
- OSGi Application Approaches
- Embedding
- Managing Service Dependencies
- Development Environment
- Open Source Frameworks

Open Source Implementations

- Apache Felix: http://felix.apache.org/
  - R4, originally called Oscar
- Knopflerfish 2: http://www.knopflerfish.org/
  - R4, open source version of UbiServ by Makewave
- Equinox: http://www.eclipse.org/equinox/
  - R4, initially developed for Eclipse and the RCP
- Concierge: http://concierge.sourceforge.net/
  - R3, optimized for resource constrained environments

Drinks!

? & !