Component Deployment on OSGi: The Gravity Case

Fractal Workshop (January 29, 2003)

Richard S. Hall
Humberto Cervantes

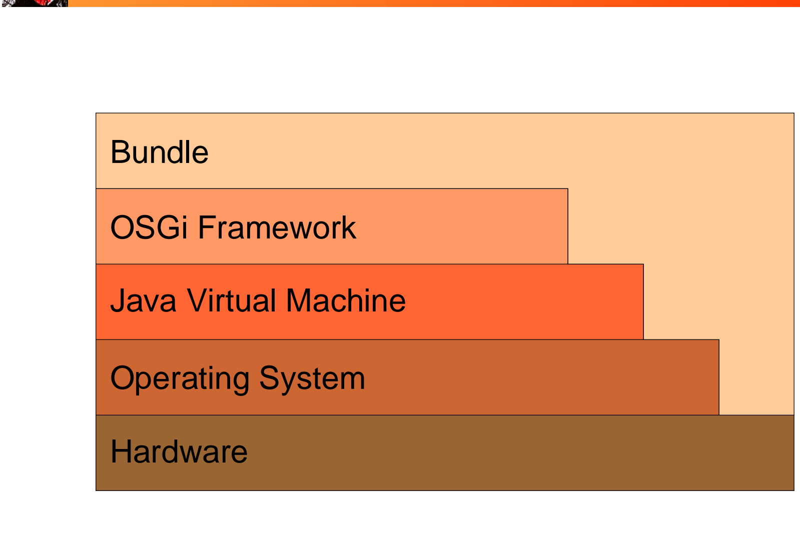
Presentation's Purpose

- Introduce and characterize the Open Services Gateway Initiative (OSGi) framework
  - Highlight deployment capabilities
- Describe our research agenda and how we use OSGi as a foundation for a service-oriented component model
- Illustrate the relationship to Fractal

Introducing OSGi
(Open Services Gateway Initiative)

- High-level characterization
  - Programmable environment for hosting dynamically downloadable services
  - Provides
    - Simple component model
    - Component life cycle management
    - Service-orientation
      - Clean separation between specification and implementation
- Low-level characterization
  - Class loader extension
  - Centralized service registry

OSGi Technology Stack



OSGi Terminology

- Framework
  - The actual service “container” that provides deployed services their environment and manages them
- Service
  - Java interface with defined semantics
- Bundle
  - Service implementation and deployment unit
    - More precisely, a JAR file that may contain Java classes, libraries, native code, and a manifest file
  - Bundles register services with the framework
  - Bundles interact only via services

Bundle Manifest

- A file called `META-INF/MANIFEST.MF` in the bundle's JAR file
- Contains meta-data about the bundle
  - The bundle's class path
  - The Java packages imported by the bundle
  - The Java packages exported by the bundle
  - The native libraries required by the bundle
- The manifest file format is attribute-value pairs (i.e., Java properties)

Bundle Manifest Example

```
Bundle-Activator: org.foo.SimpleBundle
Bundle-ClassPath: .,org/foo/embedded.jar
Bundle-NativeCode:
 libfoo.so; osname=Linux; processor=x86,
 foo.dll; osname=Windows 98; processor=x86
Import-Package:
 javax.servlet;
 specification-version=2.3
Export-Package:
 org.foo.service;
 specification-version=1.1
```

Bundle Activator

- A simple interface used by bundles to get their container context

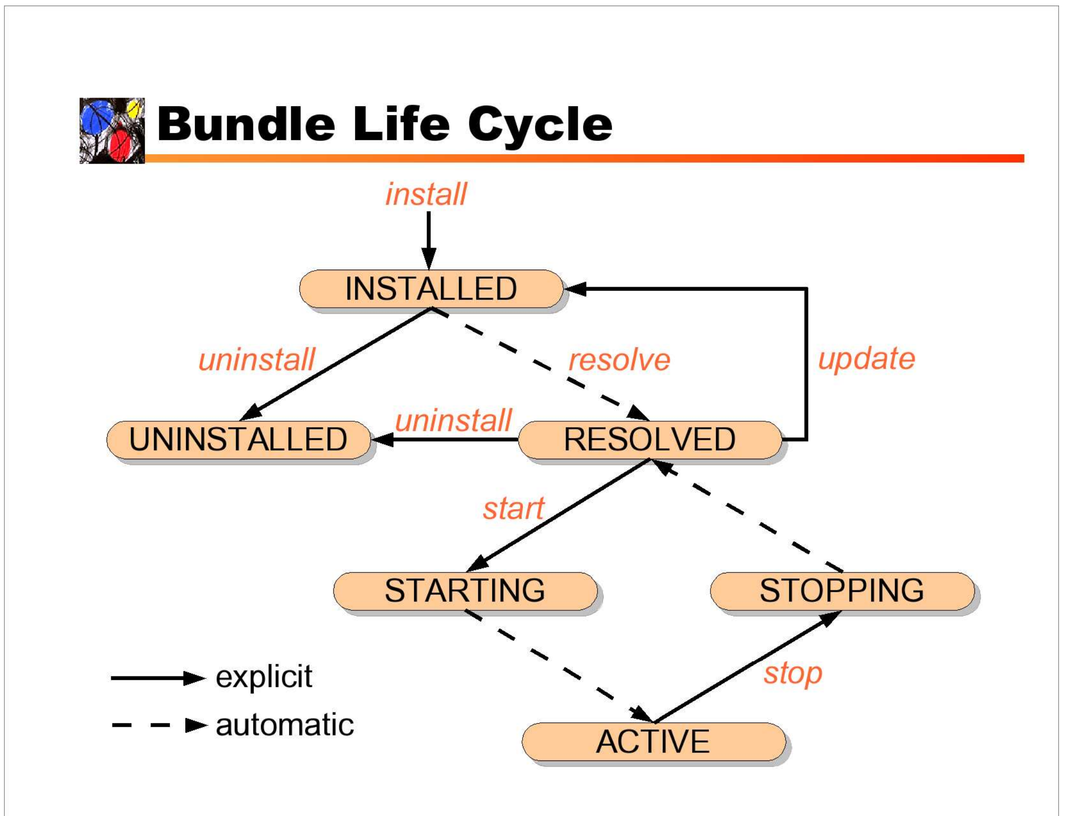
```
public interface BundleActivator
{
    public void start(BundleContext context)
        throws Exception;
    public void stop(BundleContext context)
        throws Exception;
}
```

Bundle Context

- A `BundleContext` interface represents the execution environment of a single bundle within the OSGi environment and acts as a proxy to the underlying framework and a bundle can use it to
  - Install new bundles
  - Interrogate other bundles
  - Obtain persistent storage
  - Lookup and retrieve services
  - Register services
  - Subscribe for various events

Bundle Interface

- Each bundle is represented in memory as an implementation of the interface `Bundle`
  - Used to manage the bundle, it has
    - A unique identifier
    - The bundle location (URL where retrieved)
    - A state value
    - Methods for starting, stopping, updating, and uninstalling the bundle
    - Methods for viewing registered and used services



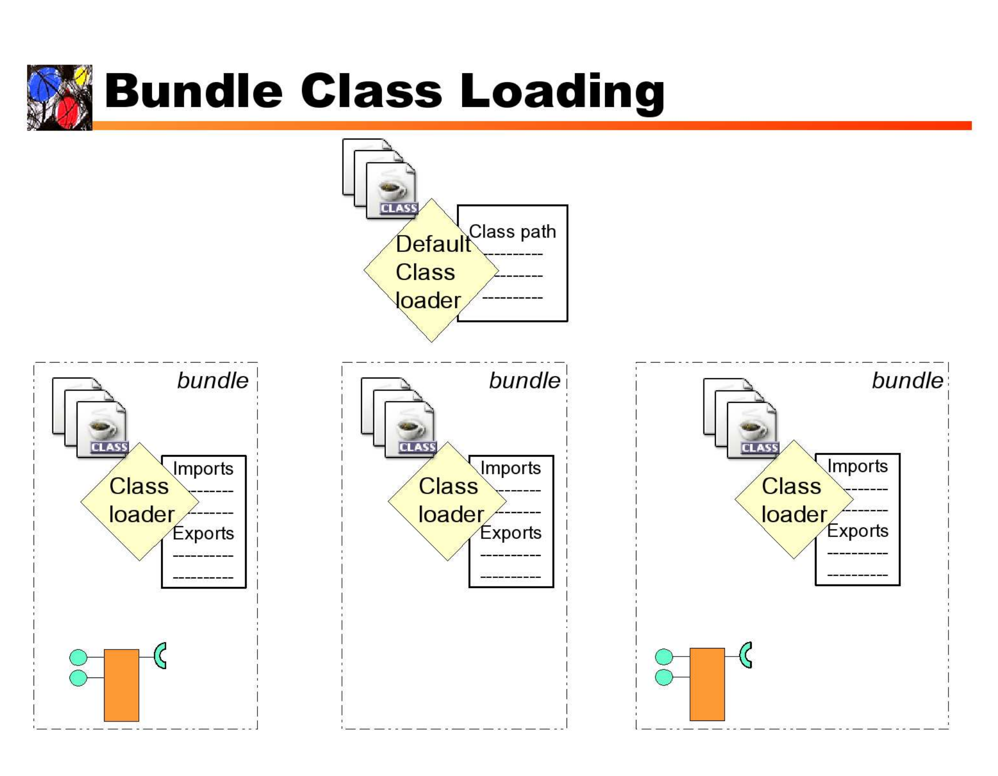
Deployment and the Bundle Life Cycle

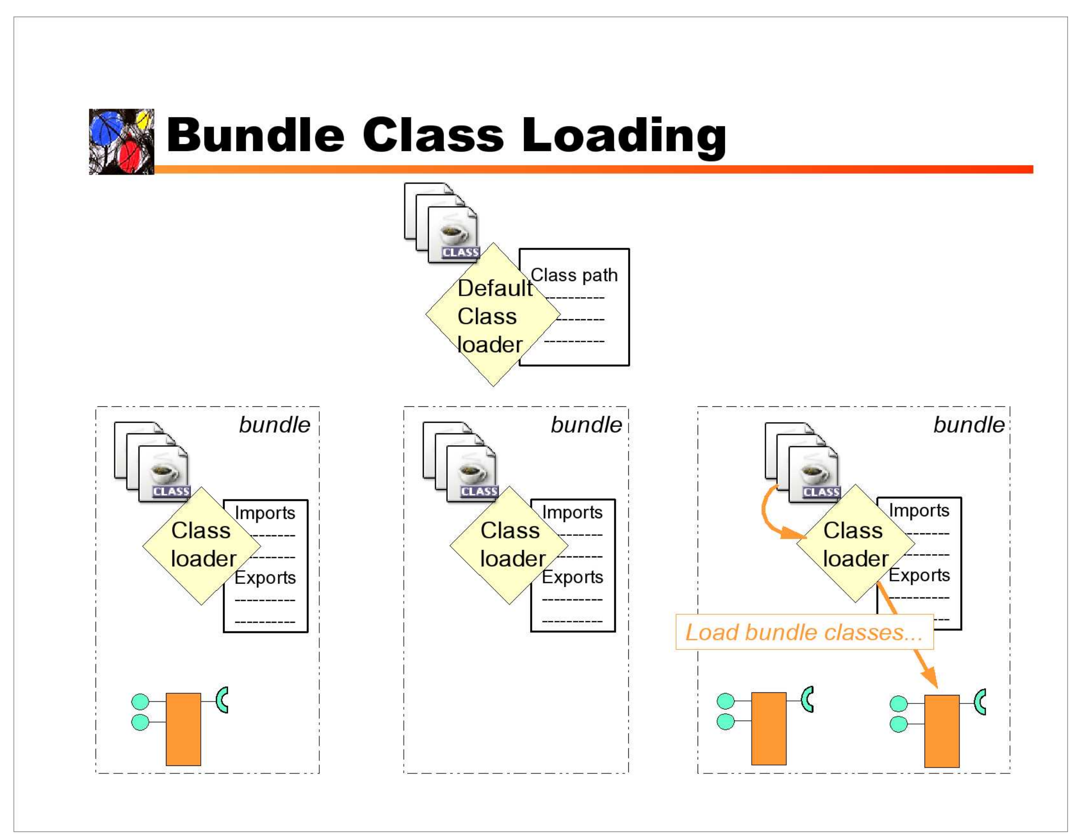
- *install* – retrieve bundle JAR file into framework, generally from a URL
- *resolve* – satisfy all package import dependencies, which enables export packages (implicit)
- *start* / *stop* – life cycle methods used to create and initialize components contained in bundle
- *update* – retrieve a new bundle JAR file, generally from a URL (deferred)
- *uninstall* – remove a bundle JAR file from framework (deferred)

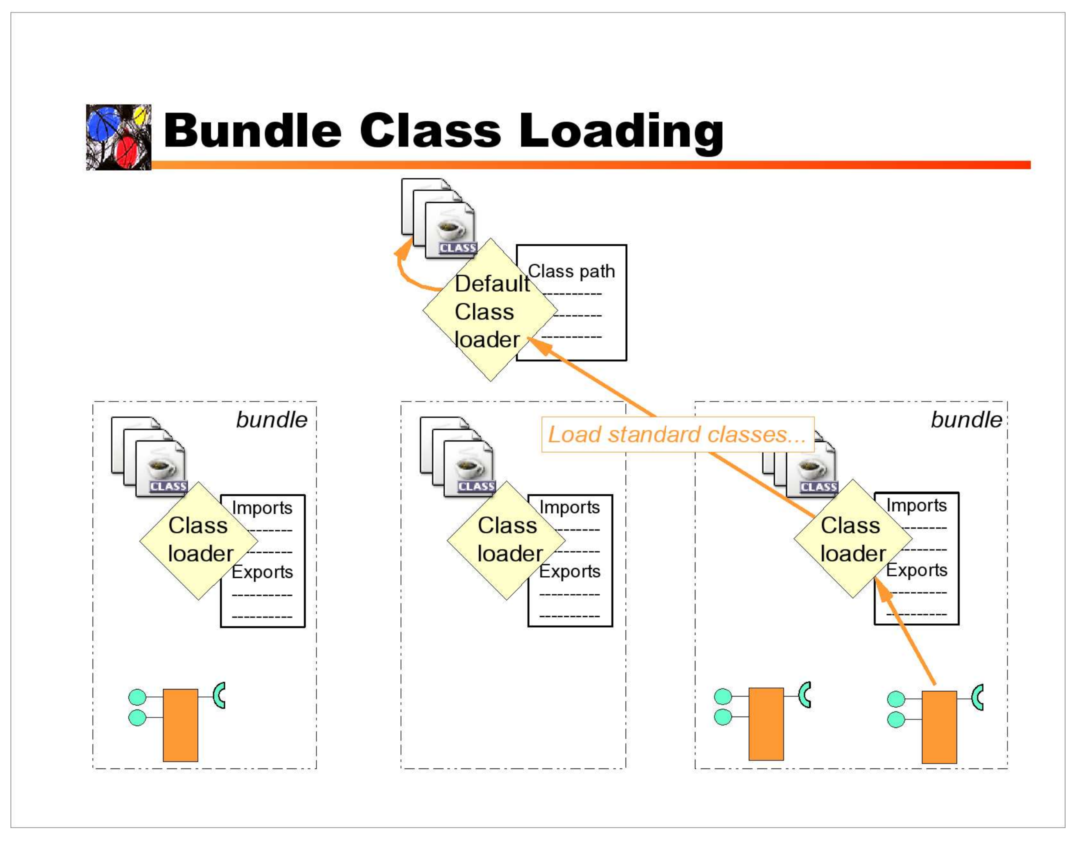
Bundle Namespace

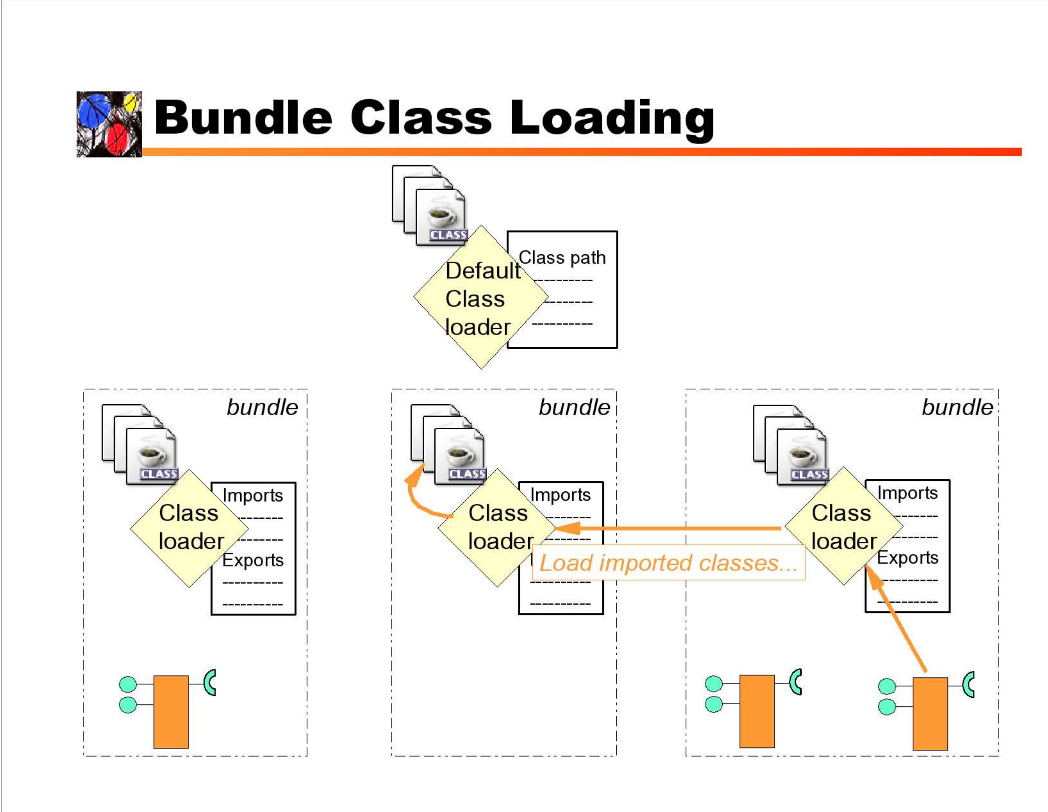
- Each bundle is loaded via its own class loader, therefore each bundle defines its own namespace
  - Avoids naming conflicts
  - Enables sharing of packages
- The bundle's class path is defined by the class path manifest entry and is “.” by default
- The bundle's class path is extended by the “import package” manifest attribute
- A bundle shares classes using the “export package” manifest attribute
  - Many bundles can export a given package, but only one instance of package is in use at any given time
- All bundles implicitly import every package that they export

Bundle Class Loading




Bundle Class Loading
Class path
Default Class loader
bundle
Imports
Class loader
Exports
bundle
Imports
Class loader
Exports
bundle
Imports
Class loader
Exports
Load bundle classes...


Bundle Class Loading
Class path
Default Class loader
bundle
Imports
Class loader
Exports
bundle
Imports
Class loader
Exports
Load standard classes...
bundle
Imports
Class loader
Exports



Services in OSGi

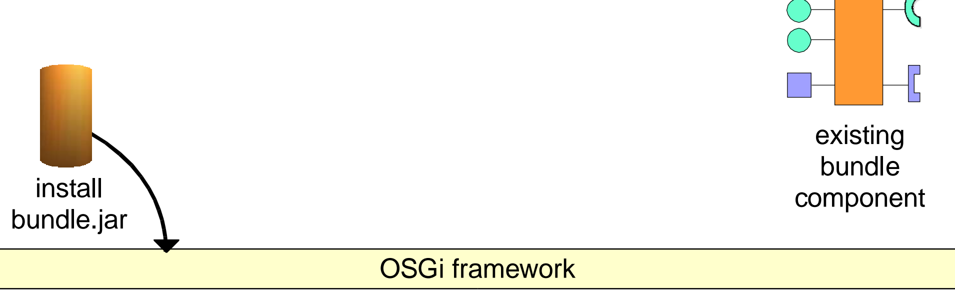
- The framework provides a shared service registry for bundles
- A service is defined by its *service interface*, a realization of a service is referred to as a *service object*
- The semantics of the service object are defined by the service interface
- Service objects may implement multiple service interfaces
- A service object is owned by a bundle and must be registered using the bundle context
- A simple LDAP query mechanism is used to look up services

Security

- Security paradigm is based on Java2
- Defines standard permissions
  - `AdminPermission` – controls access to the administrative functions of the framework
  - `ServicePermission` – controls service object registration and access
    - Grants register/get permission to named service interfaces (with wildcards)
  - `PackagePermission` – controls importing and exporting packages
    - Grants import/export permission to a named package (with wildcards)
- Services may define custom permissions

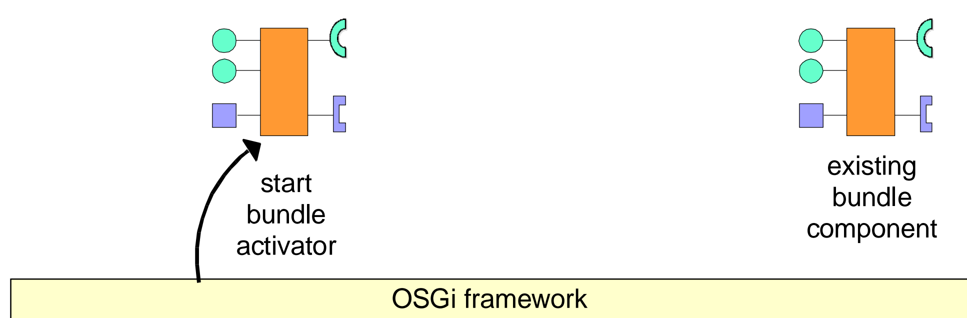
OSGi Component Model (1)

- By default, the `BundleActivator` is the single component delivered in the bundle JAR file



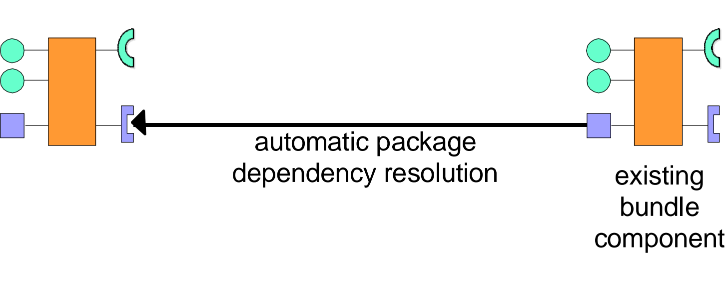
OSGi Component Model (1)

- By default, the `BundleActivator` is the single component delivered in the bundle JAR file

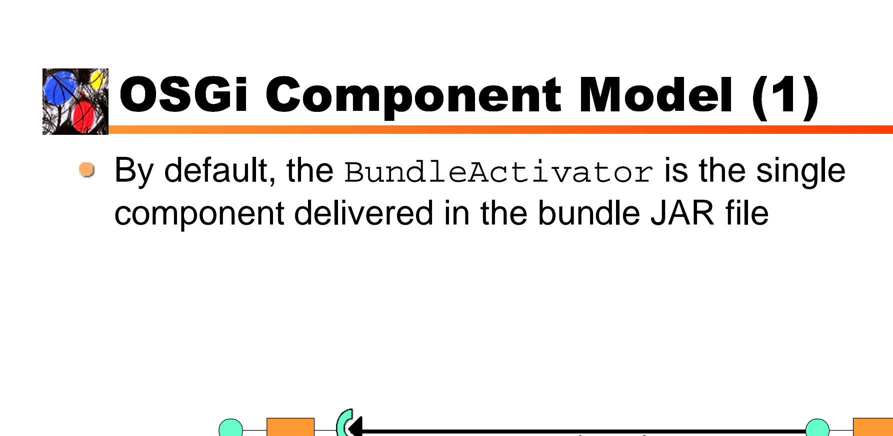


OSGi Component Model (1)

- By default, the `BundleActivator` is the single component delivered in the bundle JAR file



OSGi framework



OSGi Component Model (1)

- By default, the `BundleActivator` is the single component delivered in the bundle JAR file

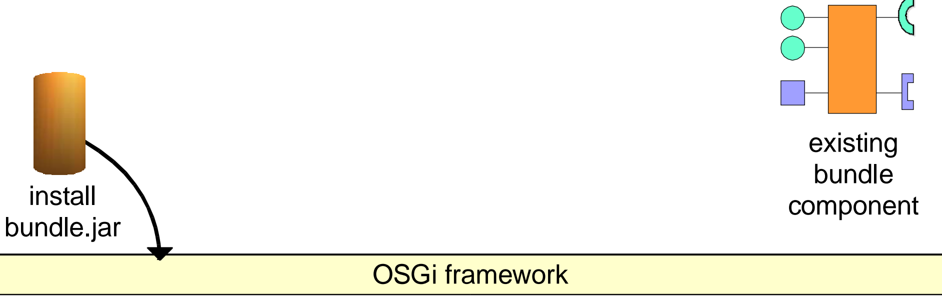
manual service dependency resolution

existing bundle component

OSGi framework

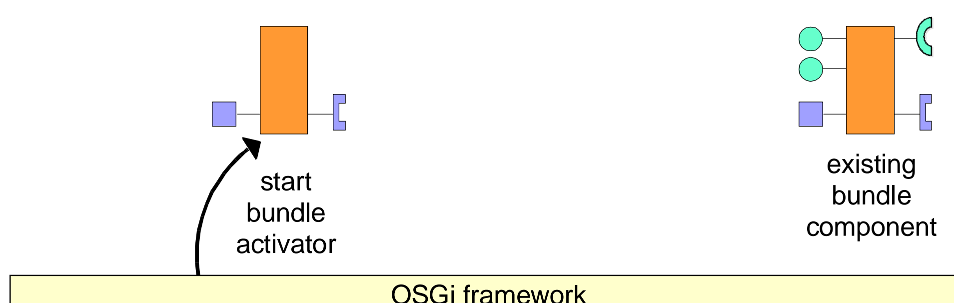
OSGi Component Model (2)

- The `start()` method of the `BundleActivator` component may create instances of other components contained in the JAR file



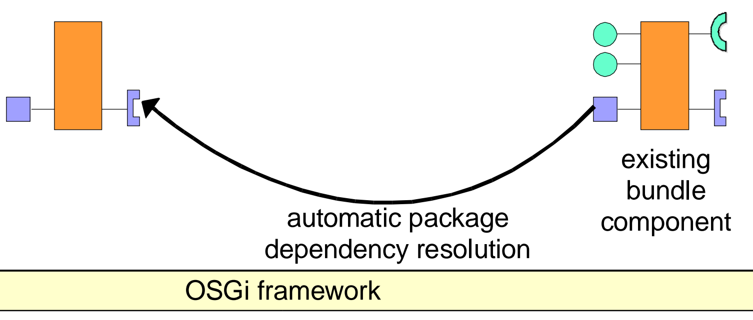
OSGi Component Model (2)

- The `start()` method of the `BundleActivator` component may create instances of other components contained in the JAR file



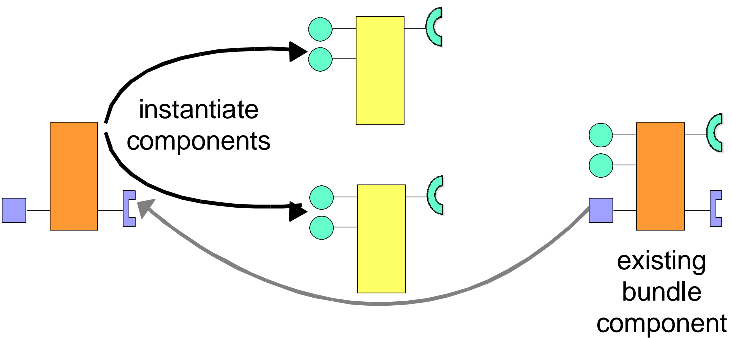
OSGi Component Model (2)

- The `start()` method of the `BundleActivator` component may create instances of other components contained in the JAR file



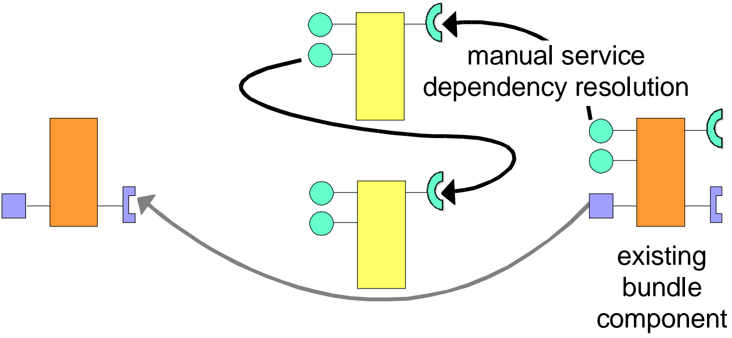
OSGi Component Model (2)

- The `start()` method of the `BundleActivator` component may create instances of other components contained in the JAR file



OSGi Component Model (2)

- The `start()` method of the `BundleActivator` component may create instances of other components contained in the JAR file



Our Research Agenda

- Investigate a service-oriented component model to support building blocks that exhibit dynamic availability
  - i.e., application building blocks may appear or disappear at any time
  - Building block availability is not under application control

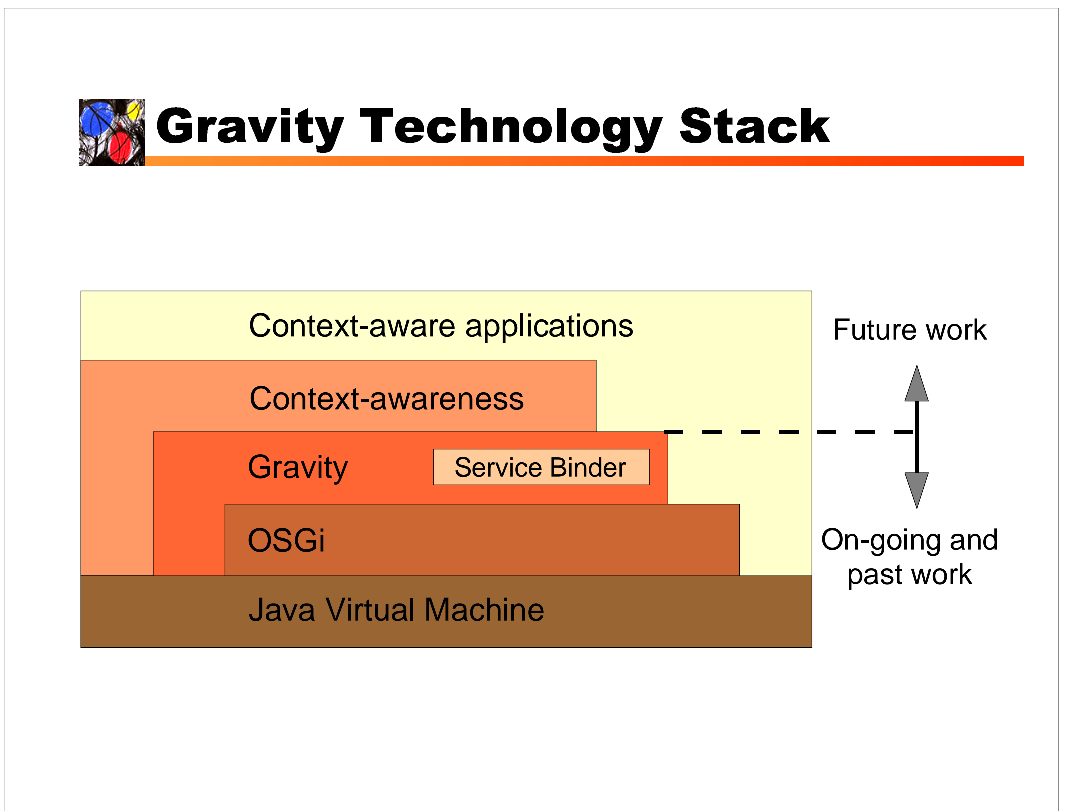
Gravity

- Gravity is a framework to support our service-oriented component and architecture models
  - Simplifies creating application out of dynamically available building blocks
  - Supports *dynamic assembly of applications*
    - Blurs the lines among application design, deployment, and usage
    - Automated architectural composition
  - Built on top of OSGi

Gravity Features

- Run-time mode
  - Application execution environment
- Design-time mode
  - Application composition modification
  - Application property modification
    - Behavioral and visual properties
  - Available at all times
- Deployment
  - Independently manageable components
    - Install, upgrade, and remove
  - Available at all times

Gravity Technology Stack



How Does OSGi Fit?

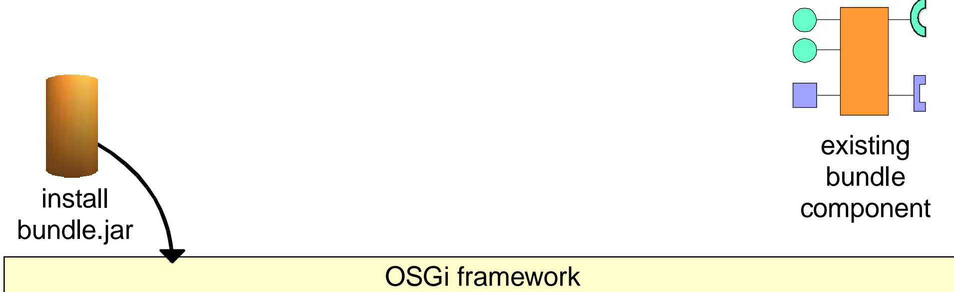
- Supports service-oriented concepts
  - Provides a service registry
- Provides deployment capabilities
  - Explicitly supports install, update, and uninstall processes
  - Automatically manages Java package/resource sharing and native libraries
  - Defines a deployment unit for component delivery

Issues Using OSGi Component Model in Gravity

- Component life cycle is subordinate to the bundle life cycle
- Component instantiation is opaque and non-standardized
  - Each bundle defines how its components are created
- Components cannot be created outside of the bundle activator
  - As above, it is non-standard
  - A `BundleContext` is required for registering component services
    - Cannot simply pass context outside of bundle, because of security concerns in OSGi

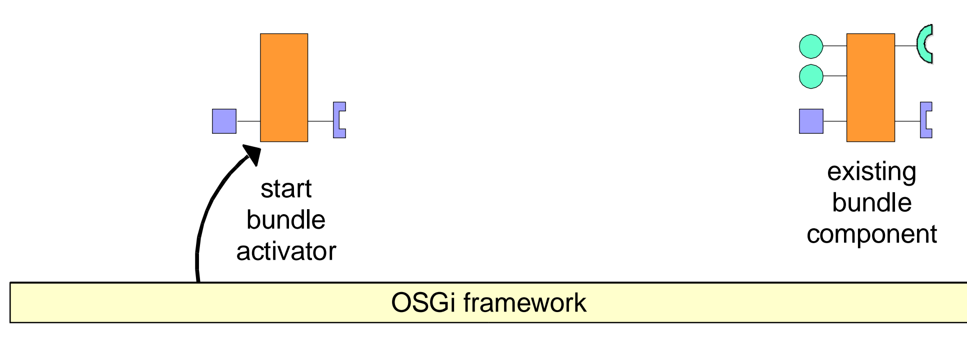
Extended OSGi Component Model for Gravity

- By introducing a `Factory` service concept, it is possible to standardize OSGi component creation



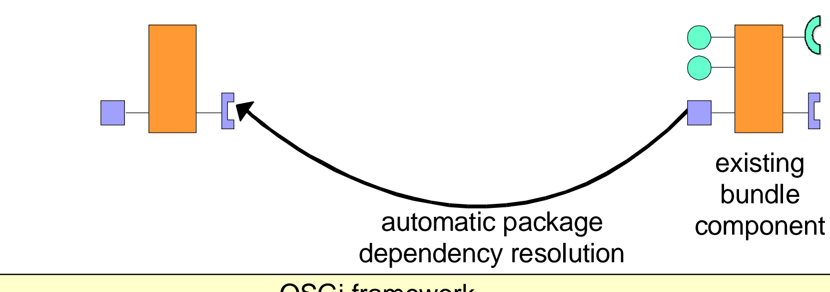
Extended OSGi Component Model for Gravity

- By introducing a `Factory` service concept, it is possible to standardize OSGi component creation



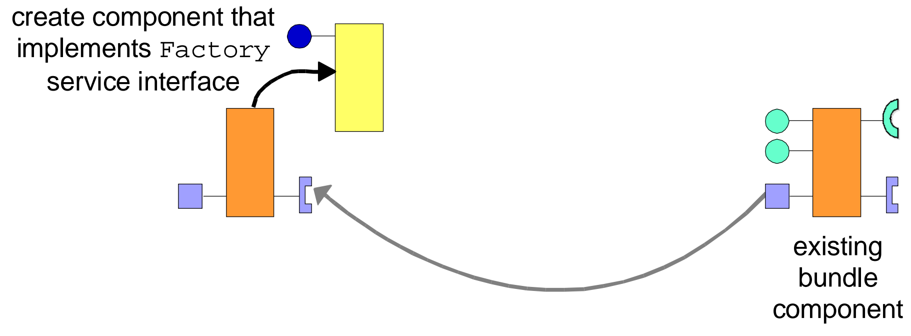
Extended OSGi Component Model for Gravity

- By introducing a `Factory` service concept, it is possible to standardize OSGi component creation



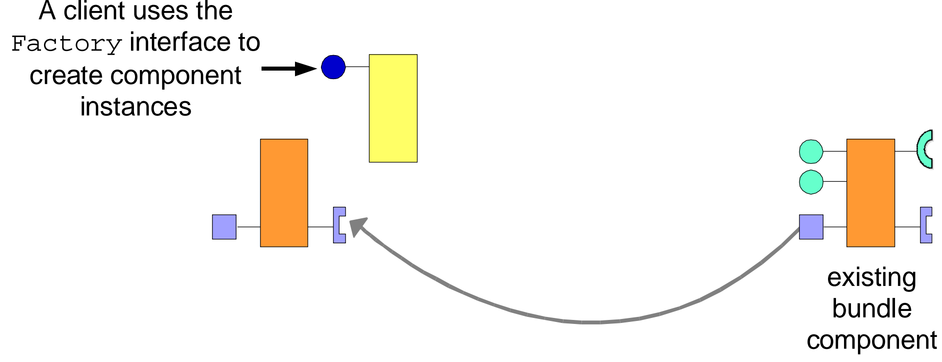
Extended OSGi Component Model for Gravity

- By introducing a `Factory` service concept, it is possible to standardize OSGi component creation



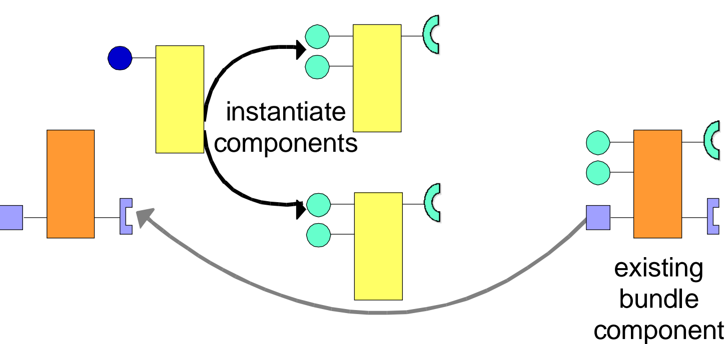
Extended OSGi Component Model for Gravity

- By introducing a `Factory` service concept, it is possible to standardize OSGi component creation



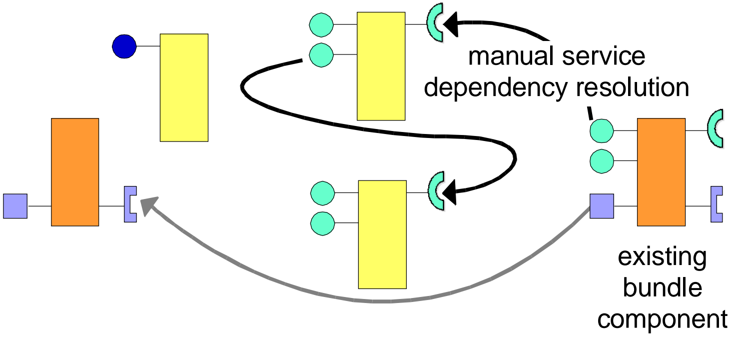
Extended OSGi Component Model for Gravity

- By introducing a `Factory` service concept, it is possible to standardize OSGi component creation



Extended OSGi Component Model for Gravity

- By introducing a `Factory` service concept, it is possible to standardize OSGi component creation



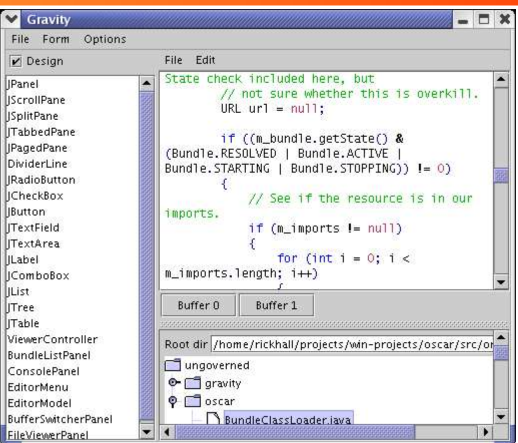
OSGi framework

Remaining Concerns

- OSGi and factories provide foundation, but
  - OSGi does not explicitly support factories
    - Only manages bundles, not factories
  - OSGi manages packages dependencies, not service dependencies
    - Must be done manually by application
      - Complex and error-prone
  - Dynamic building block availability is enabled, but not explicitly supported
  - Service-oriented architecture not supported

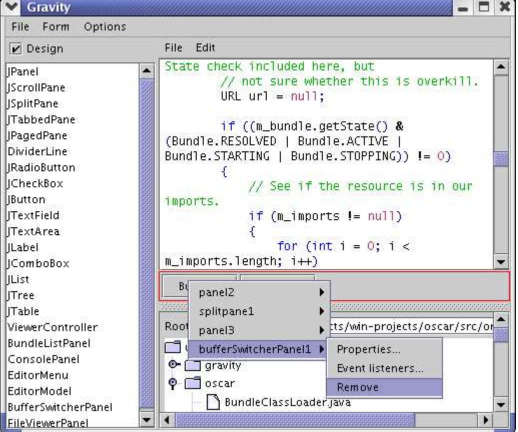
Our work attempts to address these concerns through our framework and service dependency automation...

Gravity Demonstration



- Implemented as a collection of OSGi bundles
- Provides design and execution environment with deployment capabilities always available

Gravity Demonstration



- Design mode is available at all times
  - Add, remove, modify application composition
  - Component composition is automated
- Not limited to GUI applications

Gravity Demonstration

Gravity
File Edit

```
String packageName = (name.lastIndexOf('/') >= 0)
    ? name.substring(0, name.lastIndexOf('/')) : name;
packageName = packageName.replace('/', '.');

boolean isExported = false;
for (int i = 0; !isExported && (i < m_exports.length); i++)
{
    if (m_oscar.isStrictOSGi() &&
        packageName.equals(m_exports[i].getName()))
    {
        isExported = true;
    }
    else if (!m_oscar.isStrictOSGi() &&
        packageName.startsWith(m_exports[i].getName()))
```

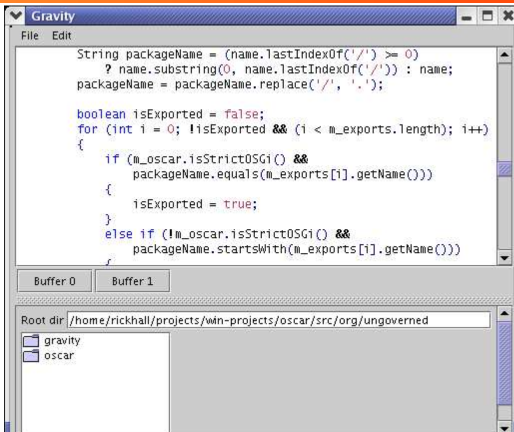
Buffer 0 Buffer 1

Root dir /home/rickhall/projects/win-projects/oscar/src/org/ungoverned

ungoverned
gravity
oscar
BundleClassLoader.java
BundleContextImpl.java

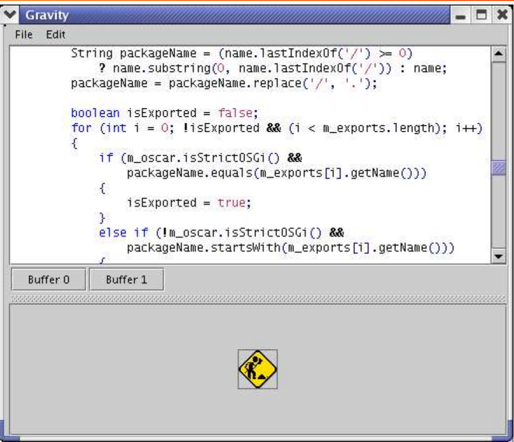
- Easily switch between design and execution
- Deployment still possible during execution

Gravity Demonstration



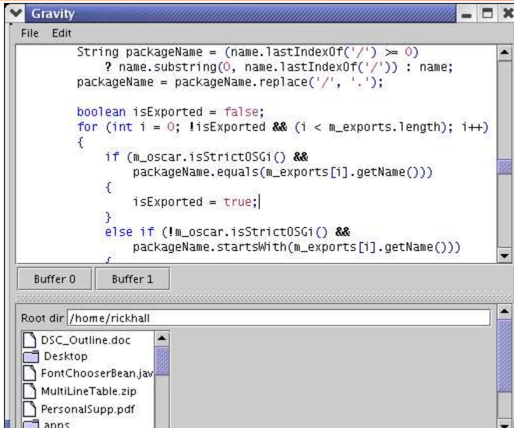
- Component service dependencies automatically maintained
  - Adapts to changes

Gravity Demonstration



- Resilient with respect to dynamic availability

Gravity Demonstration



- Always trying to maintain the “best” possible state

Fractal and OSGi?

- Related through the OSMOSE project
- OSGi “personality” on top of Fractal might not be the best approach
  - OSGi is mostly class loading
  - OSGi is intended to be a small platform
- Fractal must implement OSGi-like deployment capabilities
  - Proprietary versus standard?

OSGi “Personality” Approach

Applications
Bundles
OSGi Personality
Fractal
Java Virtual Machine

- Fractal still needs deployment capabilities
- OSGi does not need most Fractal capabilities
- OSGi is supposed to be small

OSGi Foundation Approach

Applications
Fractal
Bundles
OSGi Framework
Java Virtual Machine

- Leverages OSGi standard specification
- OSGi component model impacts Fractal component model

Deployment Layer Approach

Applications
Fractal
Bundles
OSGi API
Fractal Low-Level Deployment Layer
Java Virtual Machine

- Potentially a good approach, but adds an extra API layer (not intended to mean that Fractal is built on top of OSGi)

Conclusion

- OSGi is a lightweight, low-level framework
  - Handles dynamically loadable Java packages (i.e., super class loader)
  - Provides service-orientation
- Our project, called Gravity, effectively utilizes OSGi as the basis for its component framework
- Fractal requires deployment capabilities similar to what OSGi provides