OSGi Best Practices

Karl Pauls & Carsten Ziegeler | Adobe

Bē
Craig Ward

Carsten Ziegeler

- Principal Scientist @ Adobe Research Switzerland
- Member of the Apache Software Foundation
- VP of Apache Felix and Sling
- OSGi Expert Groups and Board member

Karl Pauls

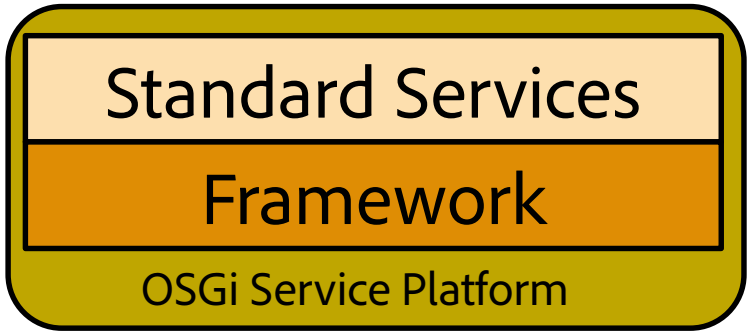
- Computer Scientist @ Adobe
- Member of the Apache Software Foundation
- PMC of Apache Felix and Sling
- Co-Author OSGi in Action

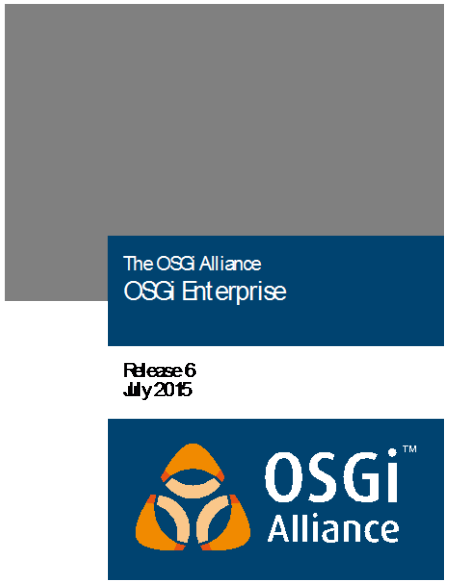
Adobe

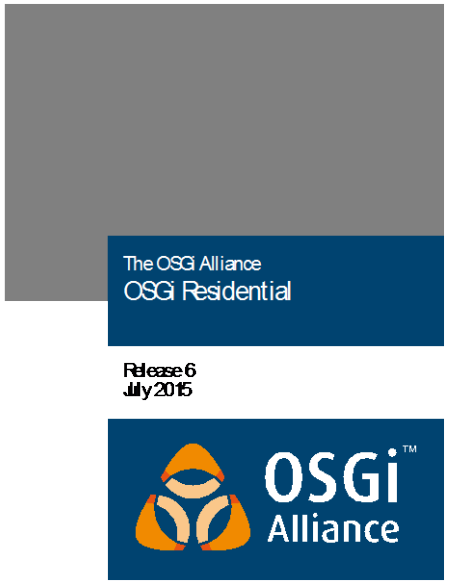
Preface

Craig Ward

OSGi Service Platform



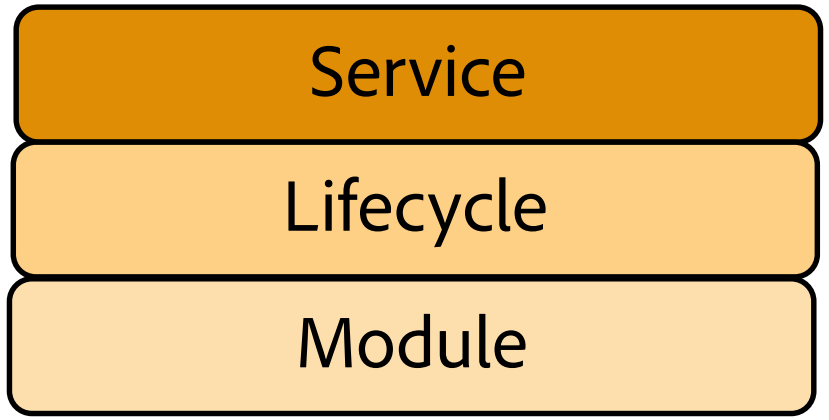


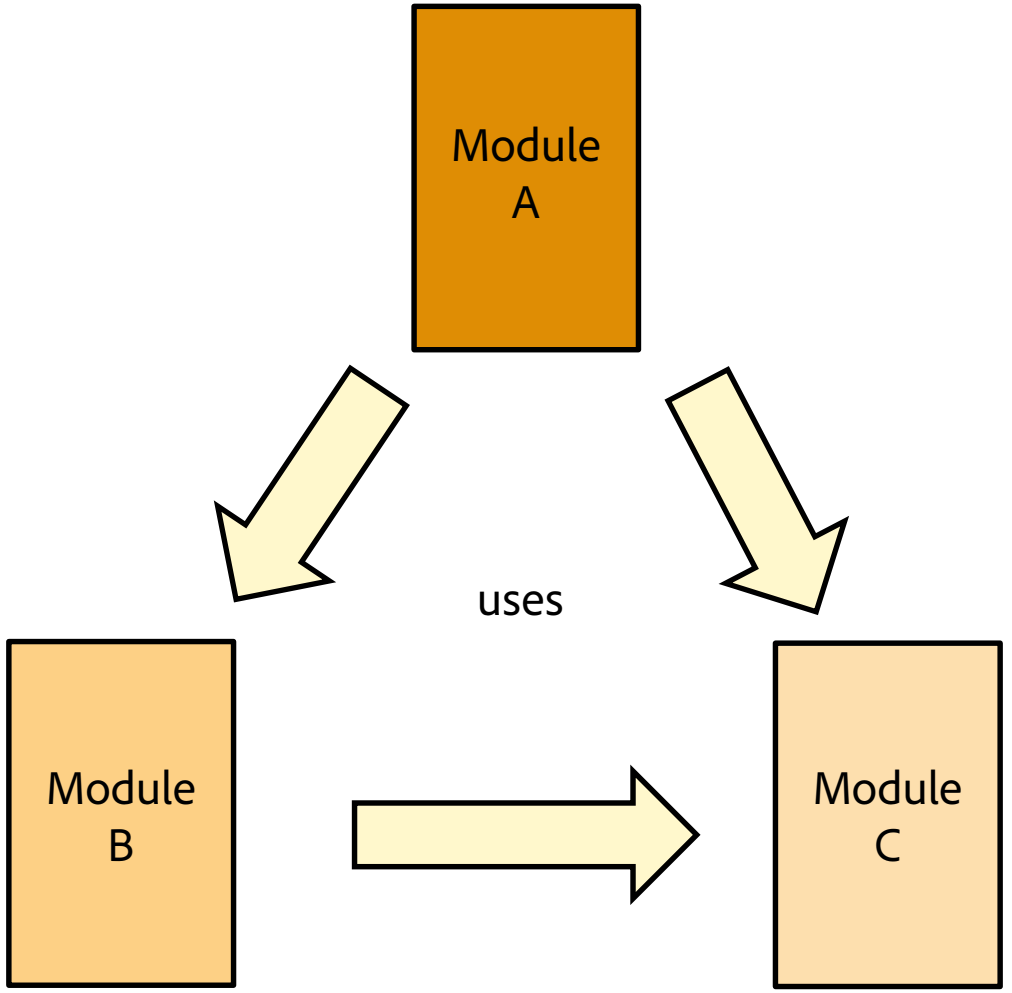


Adobe

OSGi Framework

- Dynamic Module Layer for Java
  - Low-Level Code Visibility Control
  - Side-by-Side Versioning
  - Deployment and Management Support
- Layered Architecture



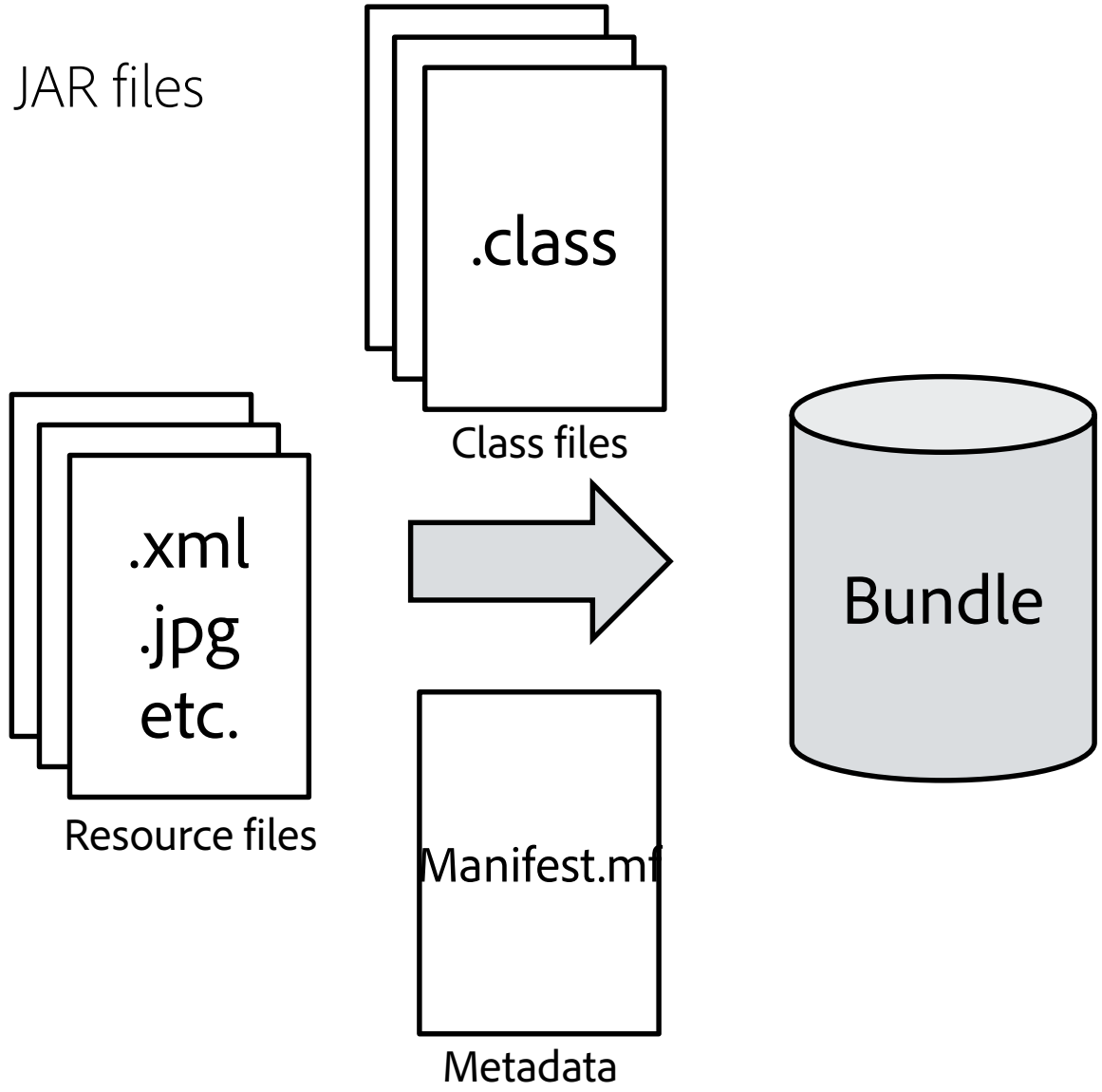


Module Layer

- Assemble applications from logically independent JAR files

```
Bundle-ManifestVersion: 2
Bundle-Name: API Provider
Bundle-SymbolicName: org.foo.api
Bundle-Version: 1.0
Export-Package: org.foo.api;version="1.0"
```

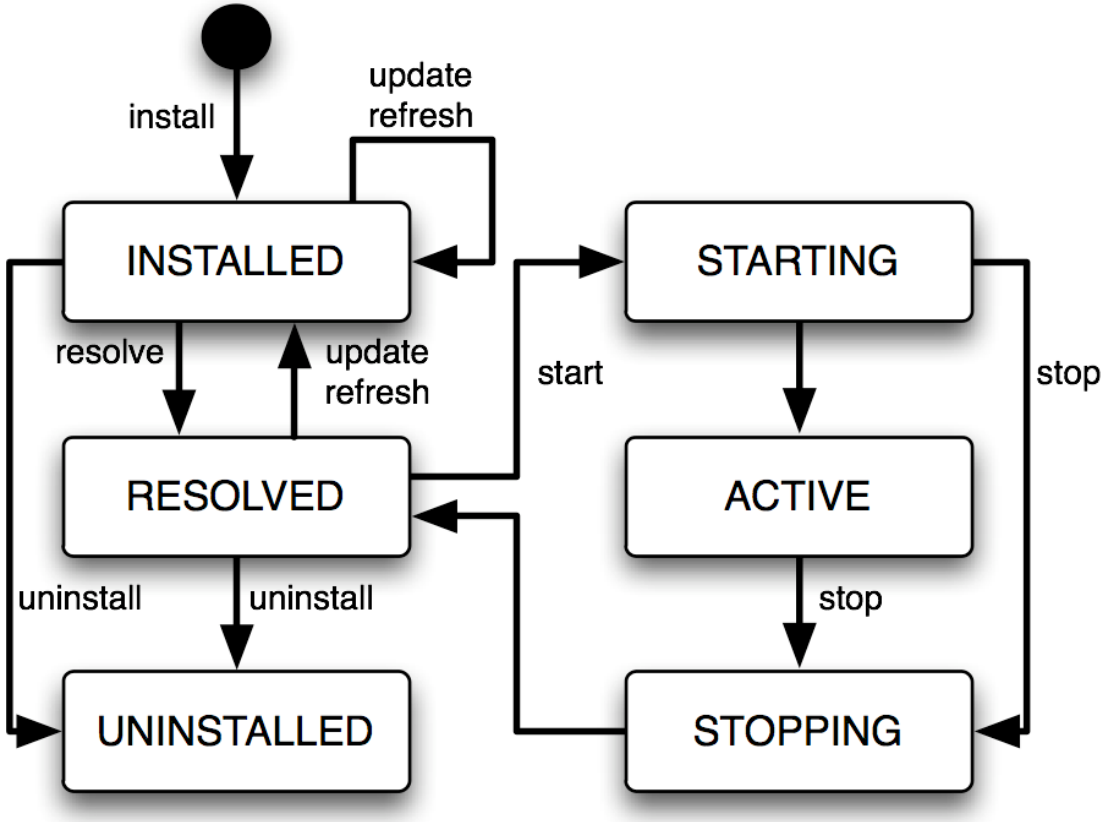
```
Bundle-ManifestVersion: 2
Bundle-Name: API Client
Bundle-SymbolicName: org.foo.api.client
Bundle-Version: 1.0
Import-Package: org.foo.api;version="[1.0,2.0)"
```



Lifecycle Layer

- Powerful extensibility mechanism with execution-time dynamism

```
package org.foo.api.provider;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {
    public void start(BundleContext ctx) {}

    public void stop(BundleContext ctx) {}
}
```

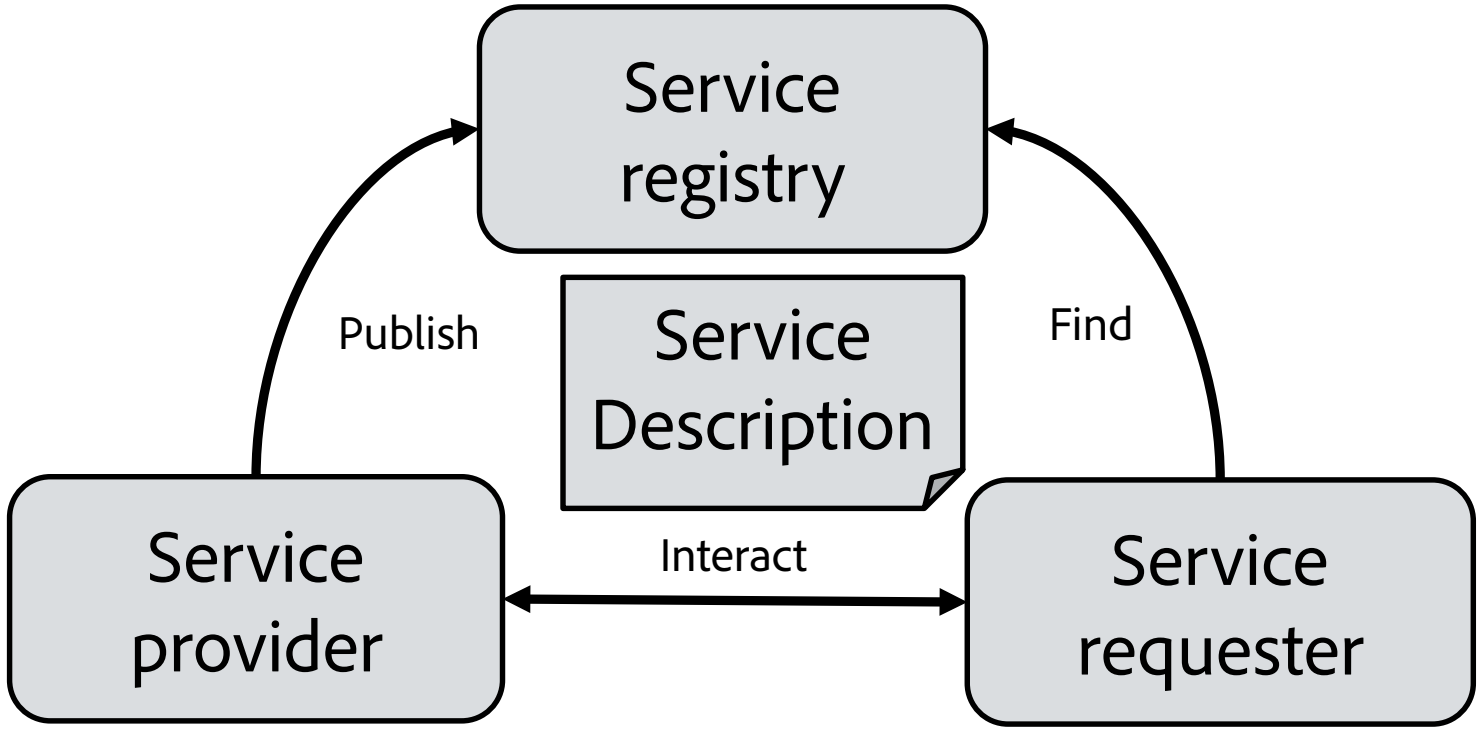


Service Layer

- Flexible application programming model incorporating service-oriented computing concepts

```
public class Activator implements BundleActivator {
   public void start(BundleContext ctx) {
      ctx.registerService(FooService.class.getName(),
         new FooServiceImpl(),
         getServiceProperties());
   }....
}
```

```
public class Activator implements BundleActivator {
   public void start(BundleContext ctx) {
      ServiceReference<FooService> ref =
         ctx.getServiceReference(FooService.class,
         getServiceFilter()
      );
      ctx.getService(ref).foo();
   }...
```



Part I

OSGi Bundle Development

OSGi Bundle Development

- Building: Apache Felix Maven Bundle Plugin / BND
- Deployment (during development): Apache Sling Maven Sling Plugin / Webconsole
- Semantic Versioning

Adobe

Apache Felix Maven Bundle Plugin / BND

- Provides ways to develop bundles with maven.

```
<plugin>
<groupId>org.apache.felix</groupId>
<artifactId>maven-bundle-plugin</artifactId>
<extensions>true</extensions>
<configuration>
<instructions>
<Export-Package>org.osgi.service.log</Export-Package>
<Private-Package>org.apache.felix.log.impl</Private-Package>
<Bundle-SymbolicName>${pom.artifactId}</Bundle-SymbolicName>
<Bundle-Activator>....impl.Activator</Bundle-Activator>
<!-- inline all non-pom dependencies, except scope runtime -->
<Embed-Dependency>
*;scope=!runtime;type=!pom;inline=true
</Embed-Dependency>
 </instructions> </configuration> </plugin>
```

```
Manifest-Version: 1
Bundle-ManifestVersion: 2
Import-Package: org.osgi.framework;version=1.3,
org.osgi.service.log;version=1.3
Export-Package:
org.osgi.service.log;uses:=org.osgi.framework;version
= 1.3
Bundle-Version: 0.8.0.SNAPSHOT
Bundle-Name: Apache Felix Log Service
Private-Package: org.apache.felix.log.impl
Bundle-Activator: org.apache.felix.log.impl.Activator
Bundle-SymbolicName: org.apache.felix.log
```

Apache Sling Maven Sling Plugin

- The Maven Sling Plugin provides a number of goals which may be of help while developing bundles for Sling.

```
<project> ... <build>
<!-- To define the plugin version in your parent POM -->
<pluginManagement> <plugins> <plugin>
        <groupId>org.apache.sling</groupId>
        <artifactId>maven-sling-plugin</artifactId>
        <version>2.3.0</version> </plugin> ... </plugins>
</pluginManagement>
<!-- To use the plugin goals in your POM or parent POM -->
<plugins> <plugin>
        <groupId>org.apache.sling</groupId>
        <artifactId>maven-sling-plugin</artifactId>
        <version>2.3.0</version>
        </plugin> ...
</plugins> </build> ... </project>
```

sling:install

Install an OSGi bundle to a running Sling instance. The plugin places an HTTP POST request to Felix Web Console.

sling:uninstall

Uninstall an OSGi bundle from a running Sling instance. The plugin an HTTP POST request to Felix Web Console to uninstall the bundle.

Semantic Versioning

- <major>.<minor>.<patch>.<qualifier>
  - Incompatible.Compatible.Patch.Internal

- Used for Import and Export packages
- Defined via annotations:

  package-info.java:

  ```
  @Version("1.2.0")
  package com.example;
  ```

1.1.1.qualifier

1.1.1.qualifier > 1.1.1

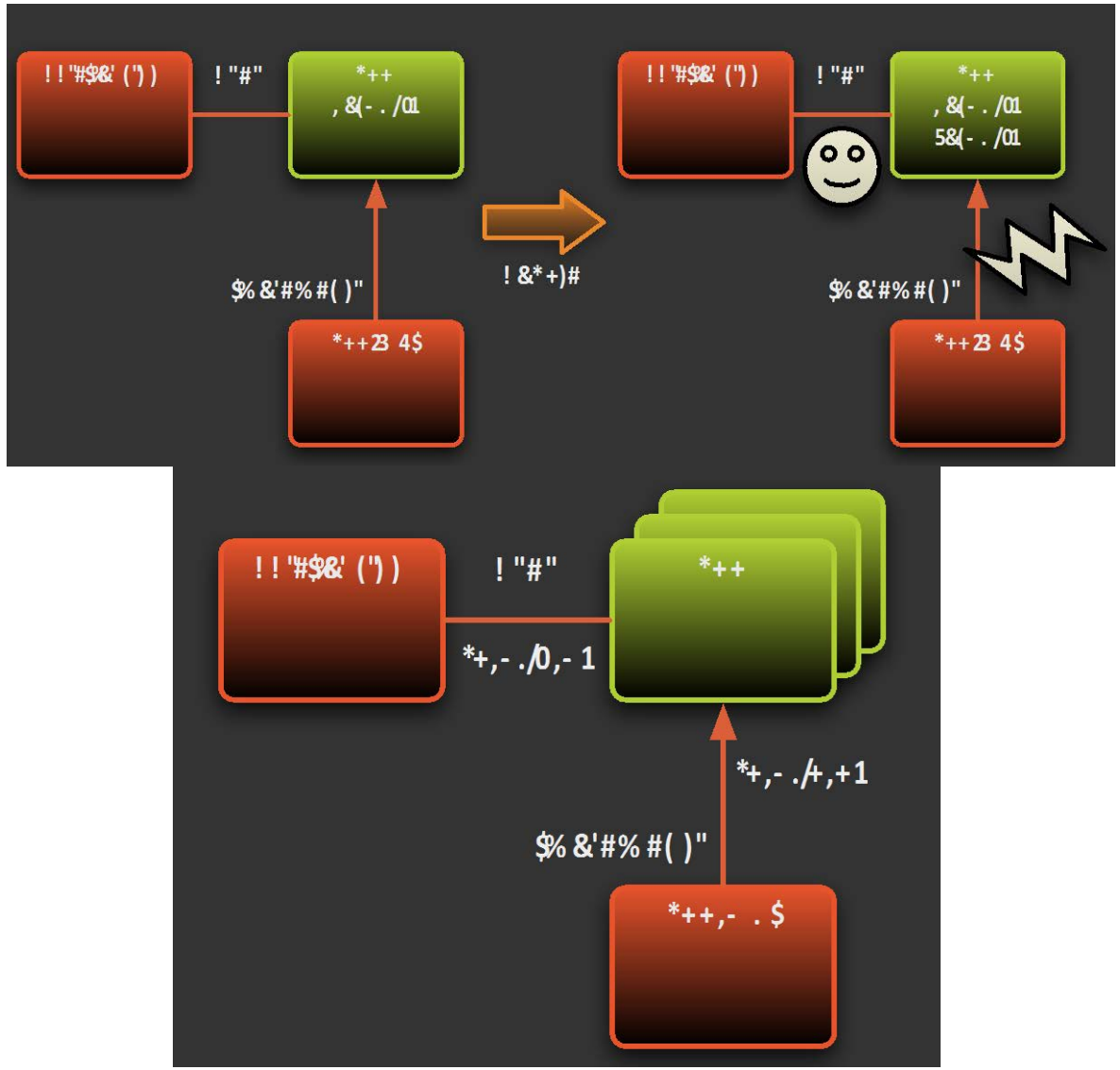
```
Export-Package: com.acme.foo; version=1.0.2
Import-Package: com.acme.bar; version="[1,2)"
```

Semantic Versioning

- Different kinds of "compatible"
  - Consumer import version=[<major>,<major>+1)
  - Provider import version=[<major>.<minor>,<major>.<minor>+1)
- Handled by bnd via annotations:

```
@ProviderType public interface SomeInterface
```

```
@ConsumerType public interface SomeInterface
```



Adobe

Baseline

- When configured with baseline, bnd will do version increases for you (according to annotations)
  - Add: <execution><id>baseline</id><goals><goal>baseline</goal></goals></execution>

```
[INFO]  PACKAGE_NAME                                DELTA     CUR_VER    BASE_VER   REC_VER    WARNINGS
[INFO] = ========================================== ========== ========== ========== ========== ==========
[INFO] ~ org.apache.sling.commons.compiler          changed   2.1.1      2.1.0      2.1.1      -
[INFO]     ~ class org.apache.sling.commons.compiler.CompilerMessage
[INFO]         + annotated org.osgi.annotation.versioning.ProviderType
[INFO]     ~ interface org.apache.sling.commons.compiler.CompilationResult
[INFO]         + annotated org.osgi.annotation.versioning.ProviderType
[INFO]     ~ interface org.apache.sling.commons.compiler.CompilationUnit
[INFO]         + annotated org.osgi.annotation.versioning.ConsumerType
[INFO]     ~ interface org.apache.sling.commons.compiler.CompilationUnitWithSource
[INFO]         + annotated org.osgi.annotation.versioning.ConsumerType
[INFO]     ~ interface org.apache.sling.commons.compiler.JavaCompiler
[INFO]         + annotated org.osgi.annotation.versioning.ProviderType
[INFO]     - version 2.1.0
[INFO]     + version 2.1.1
[INFO] ------------------------------------------------------------------------------------------------------------
[INFO] Baseline analysis complete, 0 error(s), 0 warning(s)
```


Adobe
Part II
OSGi Service Development
Be
Craig Ward

OSGi Service Development

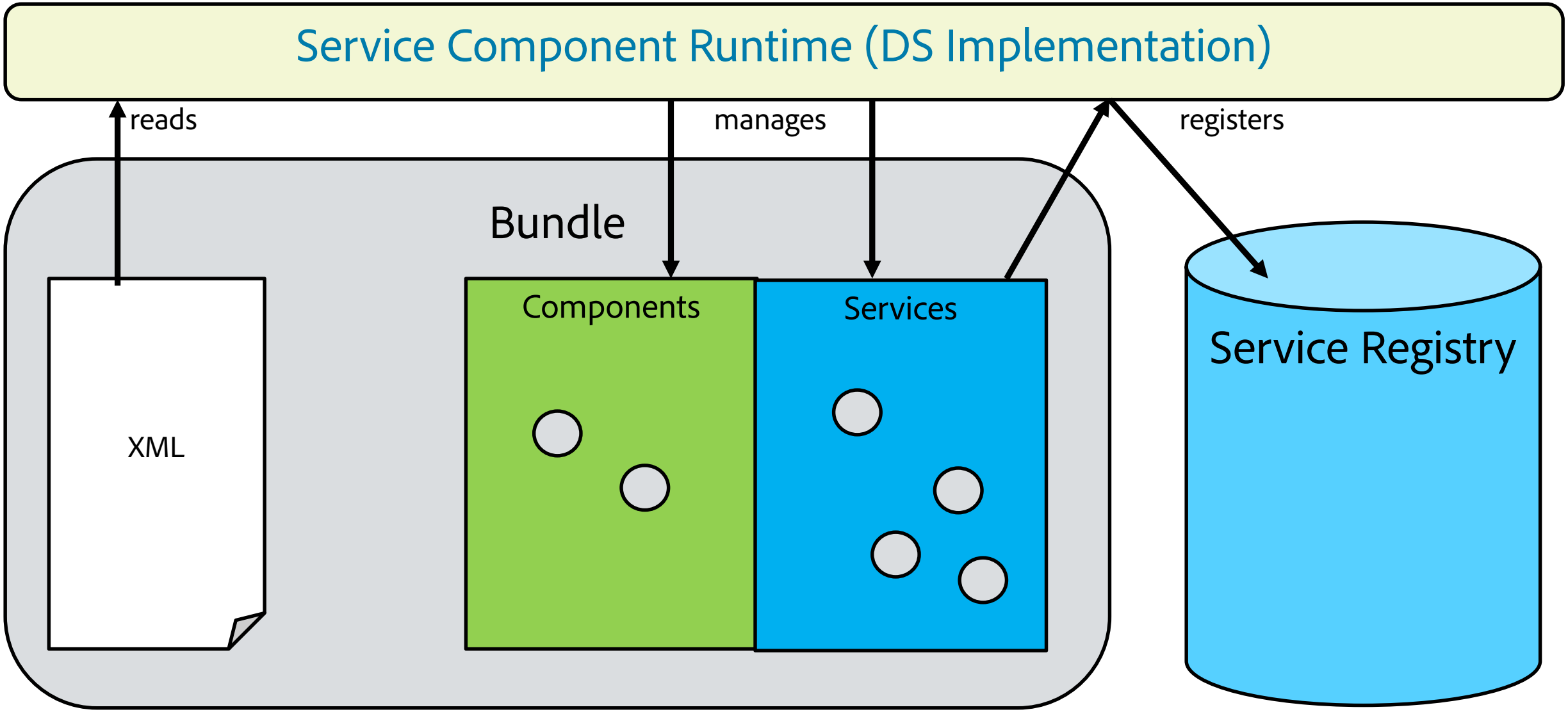
- OSGi Declarative Services Specification
- OSGi Configuration Admin Specification
- OSGi Metatype Specification

Simply Simple

- POJOs
- Annotations based
  - **org.osgi.service.component.annotations.***
  - **org.osgi.service.metatype.annotations.***
- Declarative Services does the hard work

- Based on OSGi R6 Release (AEM 6.2+)
- (Additional improvements with upcoming R7 Release – AEM 6.4+)

Declarative Services



Components

```
@Component(service = {})
public class MyComponentImpl {

    ...
```


Adobe

Lifecycle

```
@Component(service = {})
public class MyComponentImpl {

   @Activate
   protected void activate() {
      ...
   }

   @Deactivate
   private void deactivate() {
      ...
   }
```

Services

```
@Component(service = {MyComponent.class},
   property = {
      Constants.SERVICE_DESCRIPTION + "=The best service in the world",
      "service.ranking:Integer=5",
      "tag=foo",
      "tag=bar"
   }
)
public class MyComponentImpl implements MyComponent {


   ...
```

Using Services : References

- Reference cardinality
  - 0..1 : Optional service — e.g. EventAdmin
  - 1..1 : Mandatory service — e.g. ResourceResolverFactory
  - 0..n : Multiple services, optional — e.g. servlet Filter
  - 1..n : At least one service — e.g. ResourceProvider
  - X..n : At least X services — e.g. extension services

Mandatory Unary References

```
@Component(service = {MyComponent.class})
public class MyComponentImpl implements MyComponent {

   @Reference(policyOption=ReferencePolicyOption.GREEDY)
   private ResourceResovlerFactory factory;

   ... {
      factory.getResourcResolver();
   }
```

Optional Unary References

```
@Component(service = {MyComponent.class})
public class MyComponentImpl implements MyComponent {

   @Reference(policy=ReferencePolicy.DYNAMIC,
         cardinality=ReferenceCardinality.OPTIONAL,
         policyOption=ReferencePolicyOption.GREEDY)
   private volatile EventAdmin eventAdmin;

... {
      final EventAdmin localEA = eventAdmin;
      if ( localEA != null ) {
         localEA.sendEvent(...);
      }
    }
```

Multiple Cardinality Reference I

```
@Reference(cardinality=ReferenceCardinality.MULTIPLE)
private volatile List<Filter> filters;

... {
      final List<Filter> localFilters = filters;
      for(final Filter f : localFilters) {
          ...
      }
   }
```

Multiple Cardinality Reference II

```
@Reference(cardinality=ReferenceCardinality.MULTIPLE)
private final Set<Filter> filters = new ConcurrentSkipListSet<Filter>();


… {
    for(final Filter f : filters ) {
        …
    }
}
```

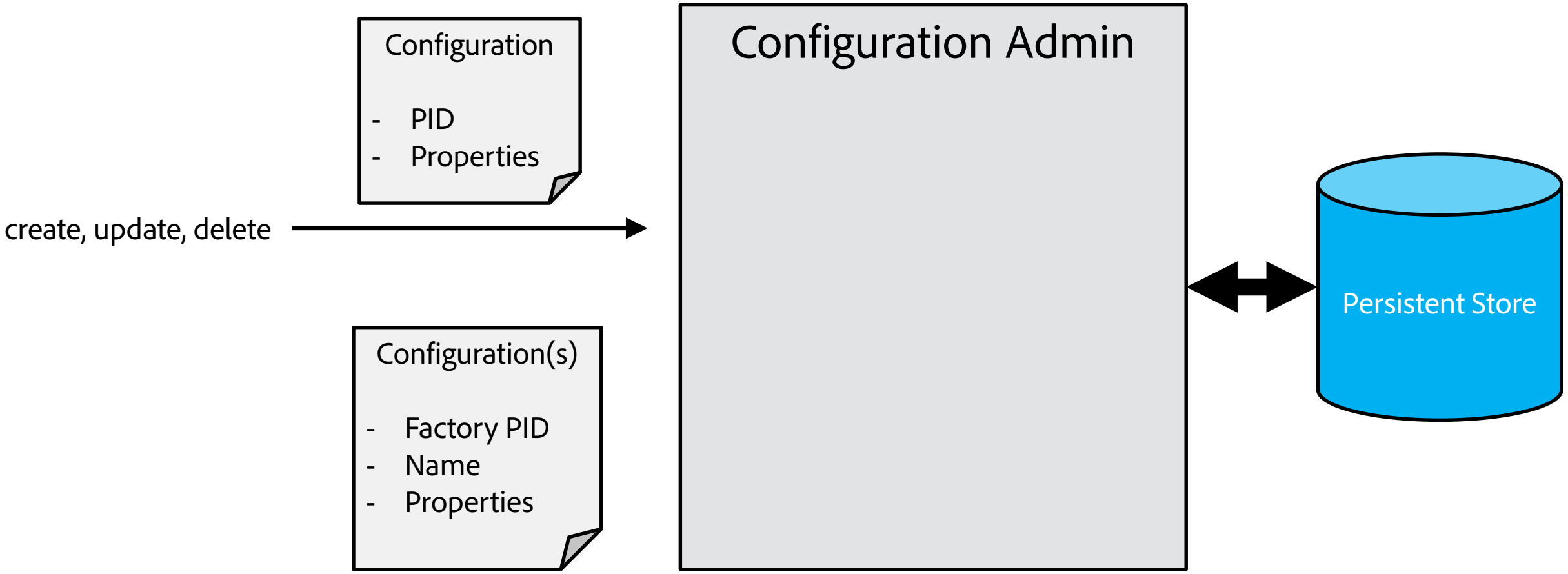
Advanced Reference Handling I

```
@Reference(service = ResourceDecorator.class,
        cardinality = ReferenceCardinality.MULTIPLE,
        policy = ReferencePolicy.DYNAMIC)
protected void bindResourceDecorator(final ResourceDecorator decorator,
                        final Map<String, Object> props) {
    ...
}

private void unbindResourceDecorator(final ResourceDecorator decorator) {
    ...
}
```

Advanced Reference Handling II

- Mandatory references are set *before* activate() is called
- References are set in alphabetical order of their name
  - Name can be specified
  - Defaults to service interface
  - Only useful for implementing bind methods
- References are unset in reverse order

Configuration Admin



Define Configuration Property Type….

- Example configuration
  - Boolean for enabled
  - String array with topics
  - String user name
  - Service ranking

```
@interface MyConfig {

        boolean enabled() default true;
        String[] topic() default {"topicA", "topicB"};
        String userName();

        int service_ranking() default 15;
}
```

..and use in lifecycle method

```
@Component(service = {})
public class MyComponentImpl {

   private MyConfig configuration;

   @Activate
   protected void activate(final MyConfig config) {
      // note: annotation MyConfig used as interface
      if ( config.enabled() ) {
        this.configuration = config;
      }
   }
}
```

Advanced Configuration Handling

- Require Configuration

```
@Component(configurationPolicy=ConfigurationPolicy.REQUIRE)
```

- Specify PID

```
@Component(name=...)
```

- For multiple service registrations (factory) use REQUIRE and name and **metatype**

Metatype

- Define type information of data
- ObjectClassDefinition with Attributes

- Common use case: Configuration properties for a component
- But not limited to this!

Configuration Metatype

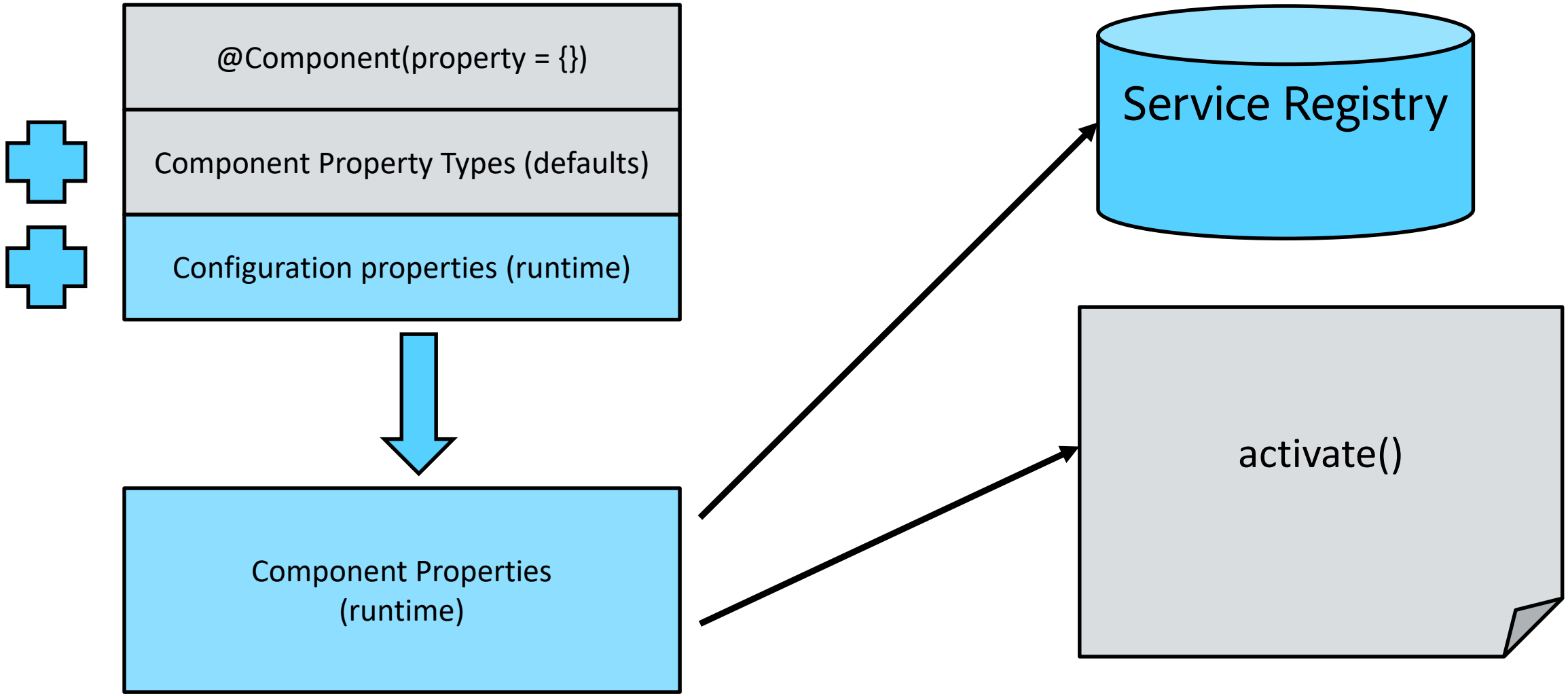
```
@ObjectClassDefinition(label="My Component",
                       description="Coolest component in the world.")
@interface MyConfig {

  @AttributeDefinition(label="Enabled",
                       description="Topic and user name are used if enabled")
  boolean enabled() default true;

  @AttributeDefinition(...)
  String[] topic() default {"topicA", "topicB"};

  @AttributeDefinition(...)
  String userName();

}
```

Connect Component Configuration with Metatype

```
@Component ( service = {})
@Designate( ocd = MyConfig.class )
public class MyComponentImpl {
```

```
@Component ( service = {Logger.class})
@Designate( ocd = MyConfig.class , factory = true)
public class LoggerImpl implements Logger {
```

Properties, Properties, Properties



Lifecycle Methods

- Activate, deactivate **and** modified (!)
- Argument Types
  - Component Property Types
  - Map (with component properties)
  - BundleContext (try to avoid)
  - ComponentContext (really try to avoid)
  - Deactivate: int with reason

Lazy vs Immediate

- Components are **immediate** by default
- Services are **lazy** by default
- Only activated if satisfied (references, configuration policy)
  - **Cascading effect**
  - **Try to use modified for configuration changes**
- Usually no need to change that behaviour (-> DON'T use immediate flag on `@Component`)
- If an implementation mixes component and service parts
  - **Split or**
  - **Use immediate flag**
  - **But only for these use cases!**

Declarative Services

- Developing OSGi Services made easy
- Configuration support
- Reference management
- Dealing with dynamics

Summary and Outlook

Best Practices

Summary

- Maven Bundle Plugin for building
- Sling Plugin / Webconsole for deployment during dev
- (Package) Versioning via Annotations
- Baseline bnd
- DS for easy annotation driven service component development incl.
  - **Lifecycle**
  - **Dependencies**
- Configuration Admin with annotations for service configuration
- Metatype for annotation driven typed properties

Adobe

Outlook

- Upcoming Apache Felix Maven OSGi best practices plugin
- Java9
- Cloud
- R7

References

- https://www.osgi.org
- https://felix.apache.org
- https://sling.apache.org
- https://www.manning.com/hall
- http://bnd.bndtools.org
- http://www.osgi.org/wp-content/uploads/SemanticVersioning1.pdf
- https://blog.osoco.de/2015/08/osgi-components-simply-simple-part-i/
- http://aries.apache.org


Adobe