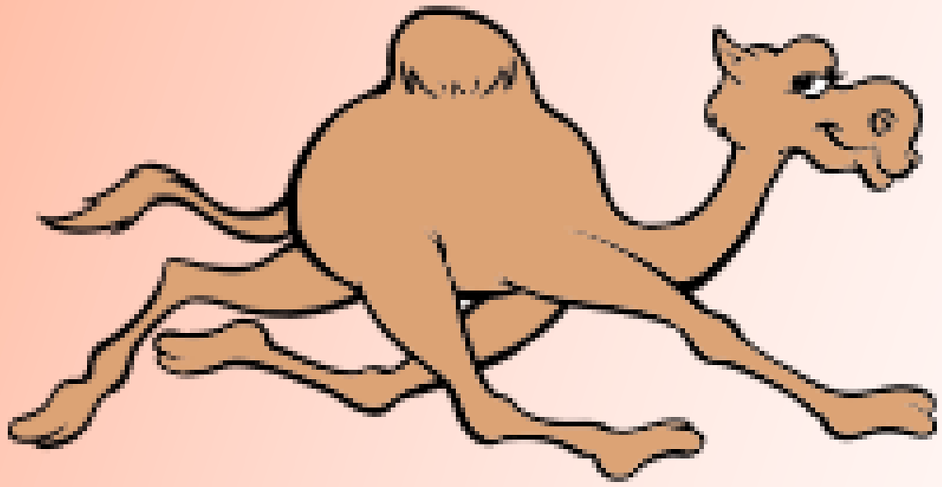
CamelOne 2013

June 10-11 2013

Boston, MA



Blueprint Services

Guillaume Nodet

Author

- Guillaume Nodet
- ASF Member, PMC member of various Apache projects (Aries, Karaf, ServiceMix, Felix …)
- Consulting Engineer at Red Hat

Agenda

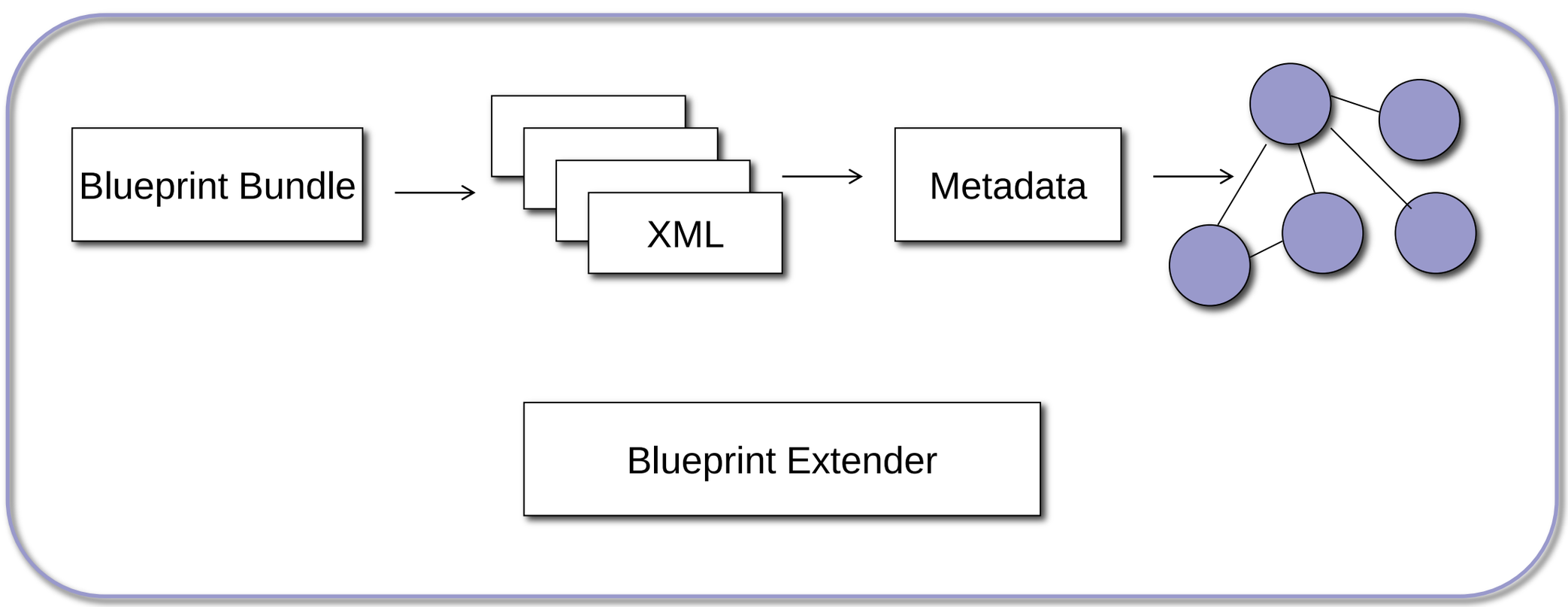
- Introduction
- Configuration
- Beans
- Service references
- Service registrations
- Advanced uses
- Custom namespaces
- Running outside OSGi
- Conclusion

Introduction

- Dependency injection / IOC
- Handle legacy code
- Handle the OSGi dynamics
- Hide the OSGi API

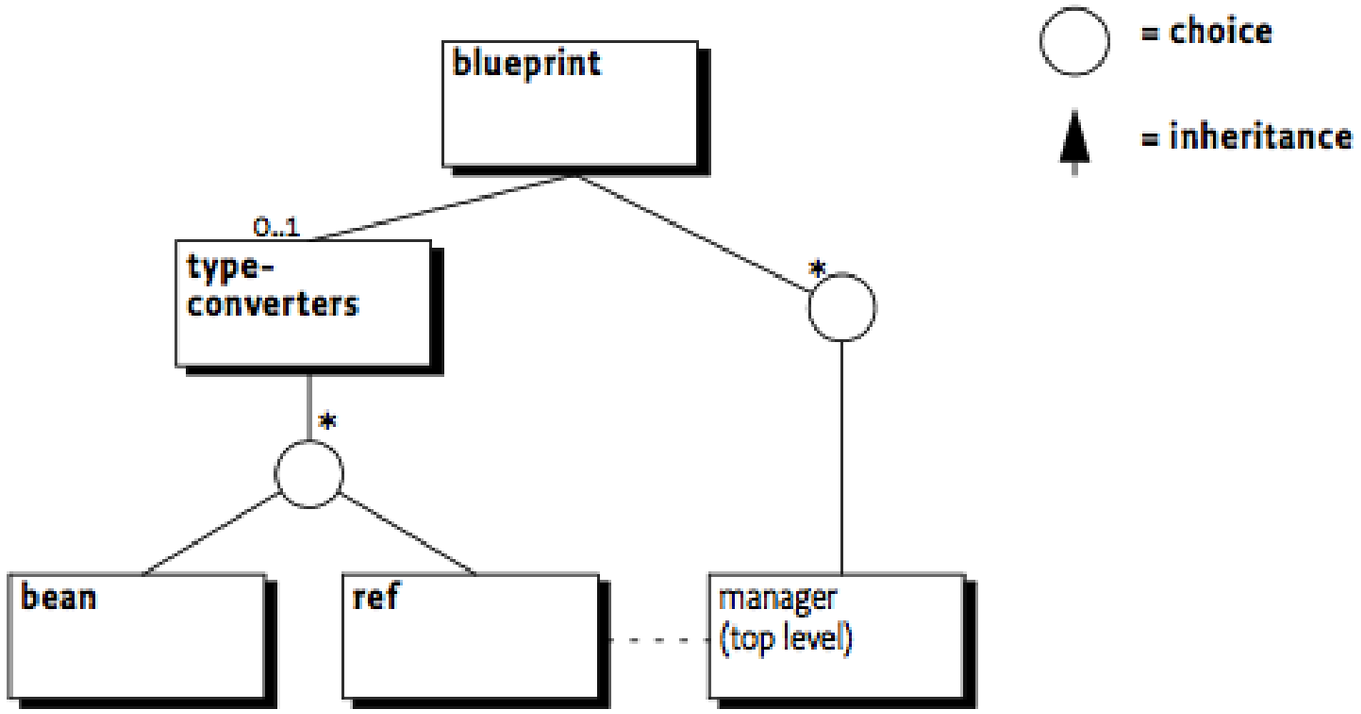
Configuration

- Extender pattern
- XML resources
- Metadata



Blueprint XML definition

blueprint ::= <type-converters> manager *
manager ::= <bean> | <service> | service-reference
service-reference ::= <reference> | <ref-list>
type-converter ::= <bean> | <ref>



Simple example

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0”>
    <service interface=“osgi.devcon.User”>
        <bean class=“osgi.devcon.impl.UserImpl”>
            <argument value=“gnodet” />
        </bean>
    </service>
</blueprint>
```

```
bundleContext.registerService(
    osgi.devcon.User.class.getName(),
    new osgi.devcon.impl.UserImpl(“gnodet”),
    new Hashtable());
```

Simple example

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0”>
    <service interface=“osgi.devcon.User”>
        <bean class=“osgi.devcon.impl.UserImpl”>
            <argument value=“gnodet” />
        </bean>
    </service>
</blueprint>
```

Top level element

```
bundleContext.registerService(
    osgi.devcon.User.class.getName(),
    new osgi.devcon.impl.UserImpl(“gnodet”),
    new Hashtable());
```

Simple example

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0”>
    <service interface=“osgi.devcon.User”>
        <bean class=“osgi.devcon.impl.UserImpl”>
            <argument value=“gnodet” />
        </bean>
    </service>
</blueprint>
```

Blueprint namespace

```
bundleContext.registerService(
    osgi.devcon.User.class.getName(),
    new osgi.devcon.impl.UserImpl(“gnodet”),
    new Hashtable());
```

Simple example

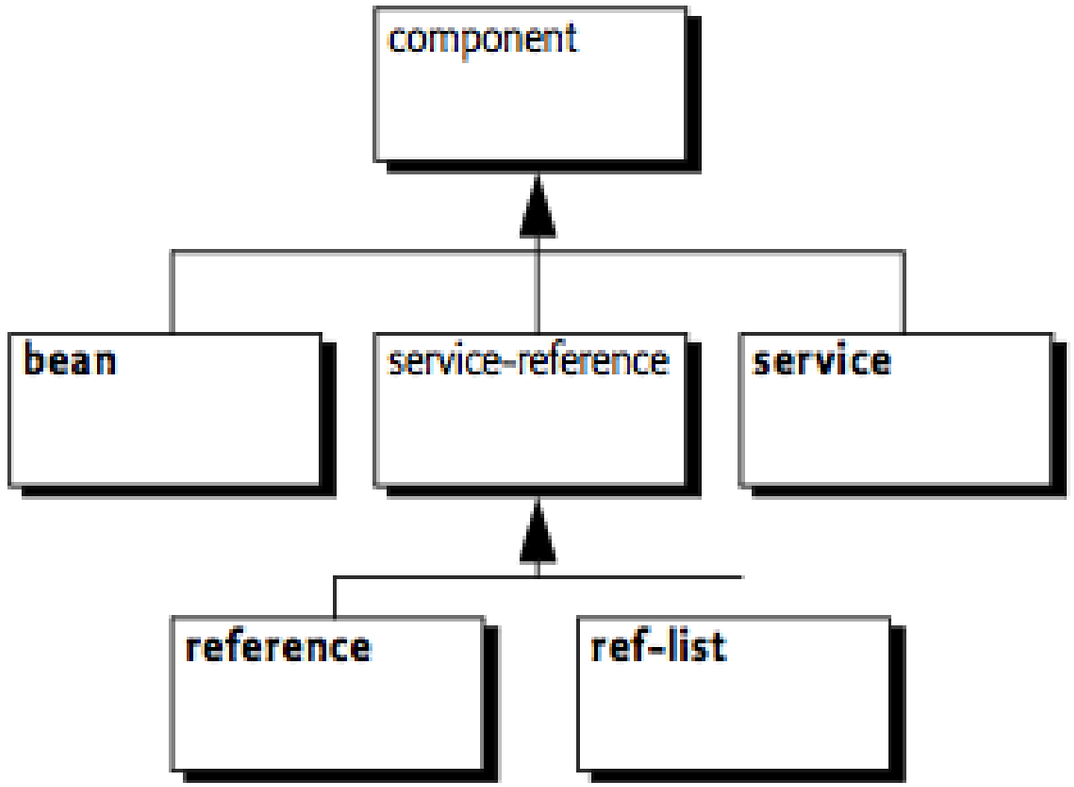
```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0”>
    <service interface=“osgi.devcon.User”>
        <bean class=“osgi.devcon.impl.UserImpl”>
            <argument value=“gnodet” />
        </bean>
    </service>
</blueprint>
```

Manager definition

```
bundleContext.registerService(
    osgi.devcon.User.class.getName(),
    new osgi.devcon.impl.UserImpl(“gnodet”),
    new Hashtable());
```

Manager types

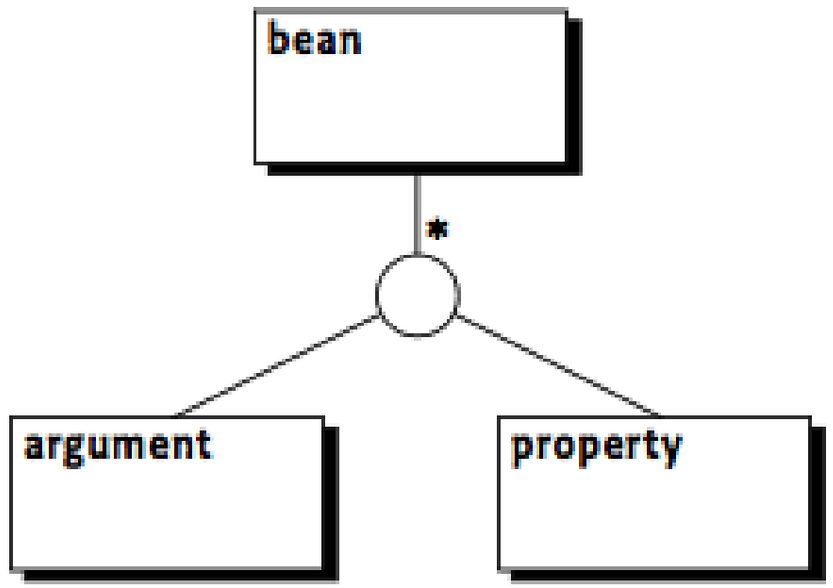
- Bean
- Single Service Reference
- Multiple Service References
- Service Registration



Managers

- Described by some metadata
- Provide objects
- Activation / deactivation
- Dependencies (implicit or explicit)
- Initialization
  - Eager
  - Lazy
- Id
- Inlined managers

Bean manager




```
<bean class="osgi.devcon.impl.UserImpl">
    <argument value="gnodet" />
    <property name="arrival" value="22/06/09" />
</bean>
```

Bean creation

- Constructor creation

```
<bean class="osgi.devcon.impl.UserImpl" />
```

```
new osgi.devcon.impl.UserImpl()
```

- Constructor creation with arguments

```
<bean class="osgi.devcon.impl.UserImpl">
    <argument value="gnodet" />
</bean>
```

```
new osgi.devcon.impl.UserImpl("gnodet")
```

Bean creation

- Static factory

```
<bean class="osgi.devcon.impl.UserFactory"
      factory-method="createUser">
    <argument value="gnodet" />
</bean>
```

```
osgi.devcon.impl.UserFactory
        .createUser( "gnodet")
```

Bean creation

- Instance factory

```
<bean factory-ref="userFactory"
      factory-method="createUser">
    <argument value="gnodet" />
</bean>
```

```
osgi.devcon.impl.UserFactory userFactory = …
userFactory.createUser( "gnodet")
```

Bean arguments

```
<argument value="gnodet"/>

<argument ref="[refid]"/>

<argument>
     [any value]
</argument>
```

Bean arguments

```
<argument value="gnodet"/>

<argument ref="[refid]"/>

<argument>
     [any value]
</argument>
```

Plain value

Bean arguments

```
<argument value="gnodet"/>

<argument ref="[refid]"/>

<argument>
     [any value]
</argument>
```

Reference to a top level manager

Bean arguments

```
<argument value="gnodet"/>

<argument ref="[refid]"/>

<argument>
     [any value]
</argument>
```

Complex value

Bean properties

```
<bean class="osgi.devcon.impl.UserImpl">
    <property name="userId" value="gnodet" />
</bean>
```

```
UserImpl user = new osgi.devcon.impl.UserImpl();
user.setUserId("gnodet");
```

Bean properties

```
<bean class="osgi.devcon.impl.UserImpl">
    <property name="userId" value="gnodet" />
</bean>
```

```
UserImpl user = new osgi.devcon.impl.UserImpl();
user.setUserId("gnodet");
```

Property name

Bean properties

```
<bean class="osgi.devcon.impl.UserImpl">
    <property name="userId" value="gnodet" />
</bean>
```

```
UserImpl user = new osgi.devcon.impl.UserImpl();
user.setUserId("gnodet");
```

Property value

Bean properties

```
<property name="userId" value="gnodet"/>

<property name="userId" ref="[refid]"/>

<property name="userId">
    [any value]
</property>
```

Bean scope

- Singleton
  - a single instance will be reused
- Prototype
  - a new instance will be created each time it is injected

Values

- `<null/>`
- `<bean>`
- `<reference>`
- `<ref-list>`
- `<service>`
- `<ref>`
- `<idref>`
- `<value>`
- `<list>`
- `<set>`
- `<map>`
- `<array>`
- `<props>`

<ref />

- Injects an object provided by the manager with the given id

```
<bean id="regId" … />

<bean class="osgi.devcon.impl.UserImpl">
    <property name="registration">
        <ref component-id="regId" />
    </property>
</bean>
```

```
<bean class="osgi.devcon.impl.UserImpl">
    <property name="registration" ref="regId" />
</bean>
```

<ref />

- Injects an object provided by the manager with the given id

```
<bean id="regId" … />

<bean class="osgi.devcon.impl.UserImpl">
    <property name="registration">
        <ref component-id="regId" />
    </property>
</bean>
```

Property value

```
<bean class="osgi.devcon.impl.UserImpl">
    <property name="registration" ref="regId" />
</bean>
```

<idref />

- Injects the id of an existing object

```
<bean id="regId" … />

<bean class="osgi.devcon.impl.UserImpl">
    <property name="registrationId">
        <idref component-id="regId" />
    </property>
</bean>
```

<value>

- Insert the content of the element text

```
<bean class="osgi.devcon.impl.UserImpl">
    <property name="userId">
        <value>gnodet</value>
    </property>
</bean>
```

```
<bean class="osgi.devcon.impl.UserImpl">
    <property name="userId" value="gnodet" />
</bean>
```

<list>, <set> and <array>

- Inserts a collection of objects

```
<list>
    <list>
        <value>2</value>
        <value>7</value>
    </list>
    <list value-type="int">
        <value>9</value>
        <value>5</value>
    </list>
</list>
```

```
Arrays.asList(
    Arrays.asList("2","7"),
    Arrays.asList(9,5)
)
```

`<list>`, `<set>` and `<array>`

- Inserts a collection of objects

```
<list>
    <list>
        <value>2</value>
        <value>7</value>
    </list>
    <list value-type="int">
        <value>9</value>
        <value>5</value>
    </list>
</list>
```

```
Arrays.asList(
    Arrays.asList("2","7"),
    Arrays.asList(9,5)
)
```

Type of the values

<map>

- Inserts a map of objects

```
<map>
    <entry key="cheese" value="cheddar"/>
    <entry key="fruit" value="orange"/>
</map>
```

```
<map>
    <entry key-ref="keyId" value="cheddar"/>
    <entry key="fruit" value-ref="valueId"/>
</map>
```

```
<map key-type=”...” value-type="... ">
    <entry ...>
</map>
```

<map>

- Inserts a map of objects

```
<map>
    <entry>
        <key>
            <value type="org.osgi.framework.Version">
                3.2.1
            </value>
        </key>
        <bean ... />
    </entry>
</map>
```

<props>

- Inserts a `java.util.Properties` object

```
<props>
    <prop key="1">one</prop>
    <prop key="2" value="two" />
</props>
```

Components as values

- Instances provided by managers can be injected

```
<list>
     <bean class="com.acme.FooImpl"/>
</list>
```

Service References

- Single service: **<reference>**

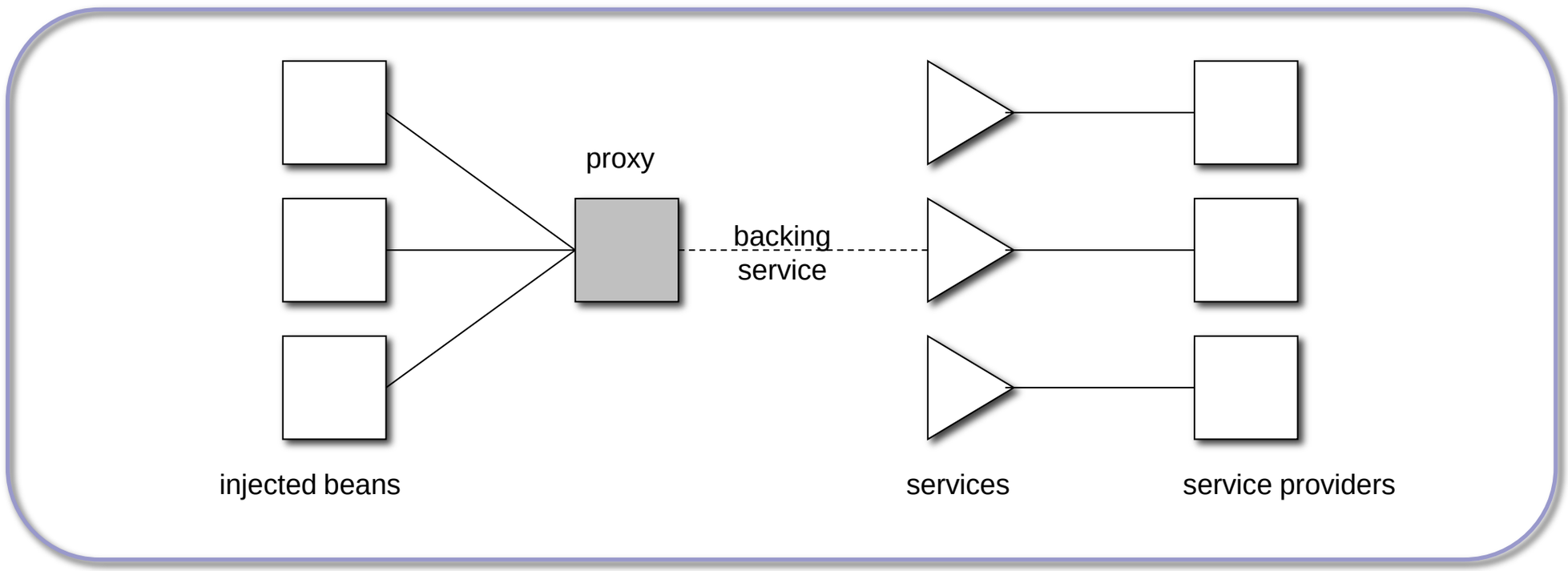
```
<reference id="user1"
           interface="osgi.devcon.User"
           filter="(name=gnodet)" />
```

- Multiple services: **<ref-list>**

```
<ref-list id=”all-users"
          interface="osgi.devcon.User” />
```

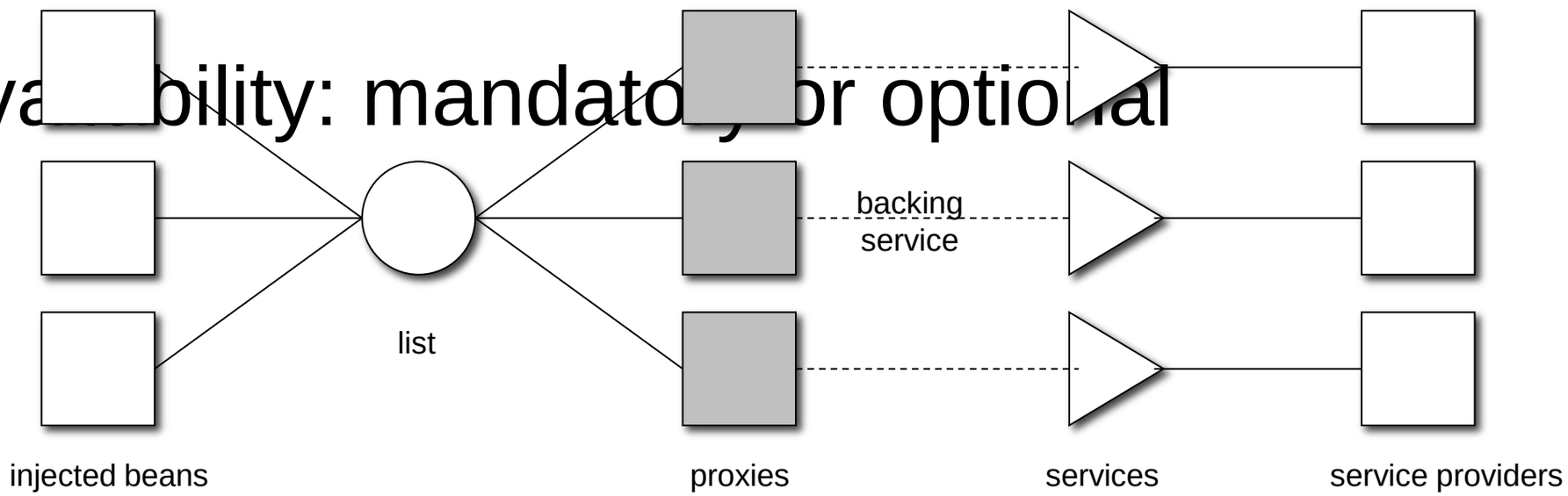
<reference>

- Provides a proxy to an OSGi service
- Availability: mandatory or optional
- Timeout
- Damping



<ref-list>

- Provides a read-only and dynamic list of OSGi service
- Availability: mandatory or optional



Service References Listeners

```
<bean id="listener" ... />

<ref-list interface="osgi.devcon.User">
    <reference-listener ref="listener"
                        bind-method="bind"
                        unbind-method="unbind" />
</ref-list>
```

```
public class Listener {
    public void bind(User user) { }
    public void unbind(User user) { }
}
```

```
public void (T)
public void (T, Map)
public void (ServiceReference)
```

Service References Listeners

```
<bean id="listener" ... />

<ref-list interface="osgi.devcon.User">
    <reference-listener ref="listener"
                        bind-method="bind"
                        unbind-method="unbind" />
</ref-list>
```

```
public class Listener {
    public void bind(User user) { }
    public void unbind(User user) { }
}
```

```
public void (T)
public void (T, Map)
public void (ServiceReference)
```

Bind method

Service registrations

- Expose an object as an OSGi service
- Register a ServiceFactory
- Dependencies on service references

```
<service ref="user"
         interface="osgi.devcon.User" />
```

```
<service auto-export="interfaces">
    <bean class="osgi.devcon.impl.UserImpl" />
</service>
```

Service properties

- Expose an object as an OSGi service

```
<service ref="fooImpl" interface="osgi.devcon.User">
    <service-properties>
        <entry key="name" value="gnodet"/>
    </service-properties>
</service>
```

Service Registration Listeners

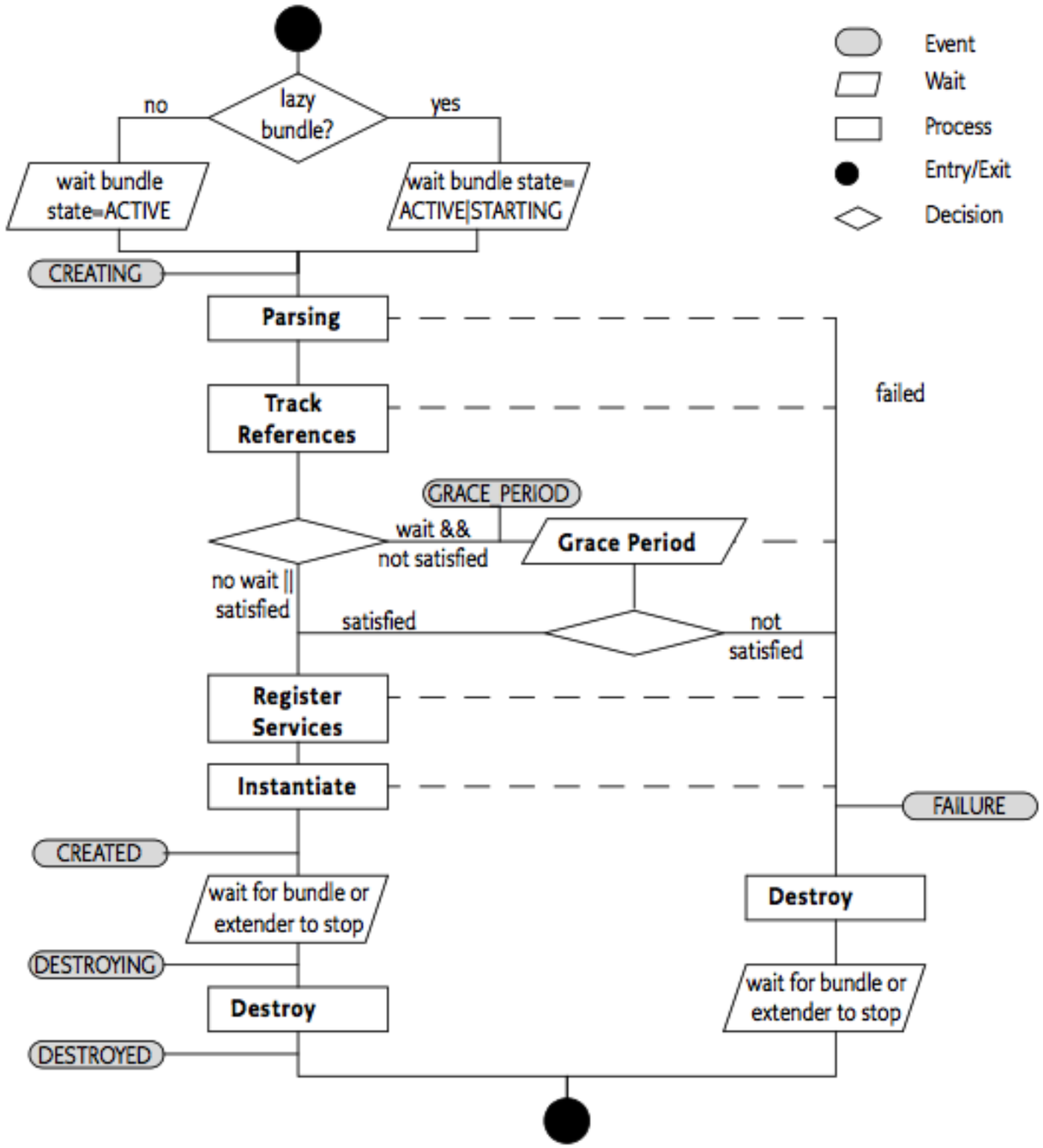
```
<bean id="listener" ... />

<service ref="..." interface="osgi.devcon.User">
    <registration-listener
        ref="listener"
        registration-method="register"
        unregistration-method="unregister" />
</ref-list>
```

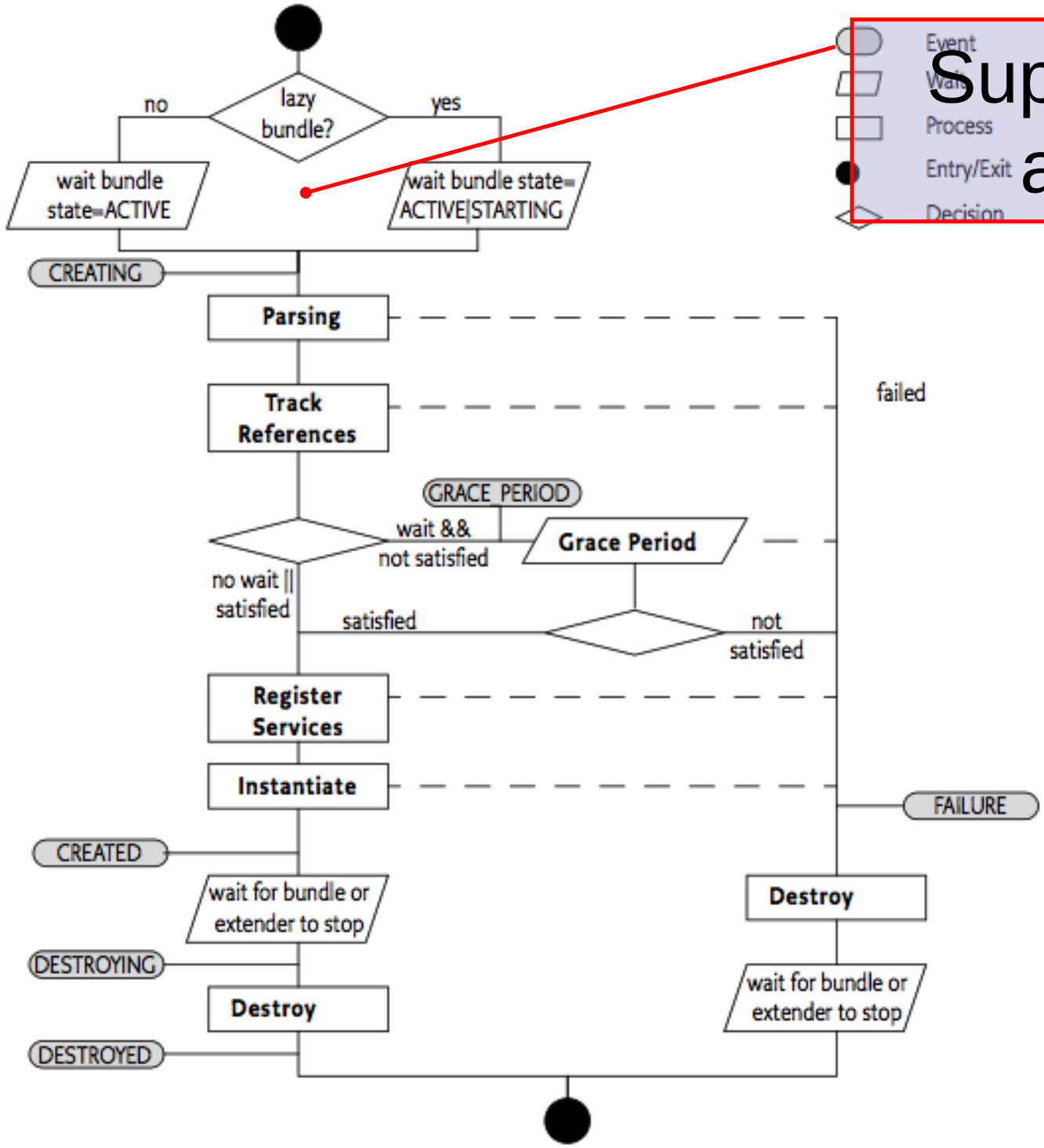
```
public class Listener {
    public void register(User user, Map props) { }
    public void unregister(User user, Map props) { }
}
```

```
public void (T, Map)
```

Lifecycle

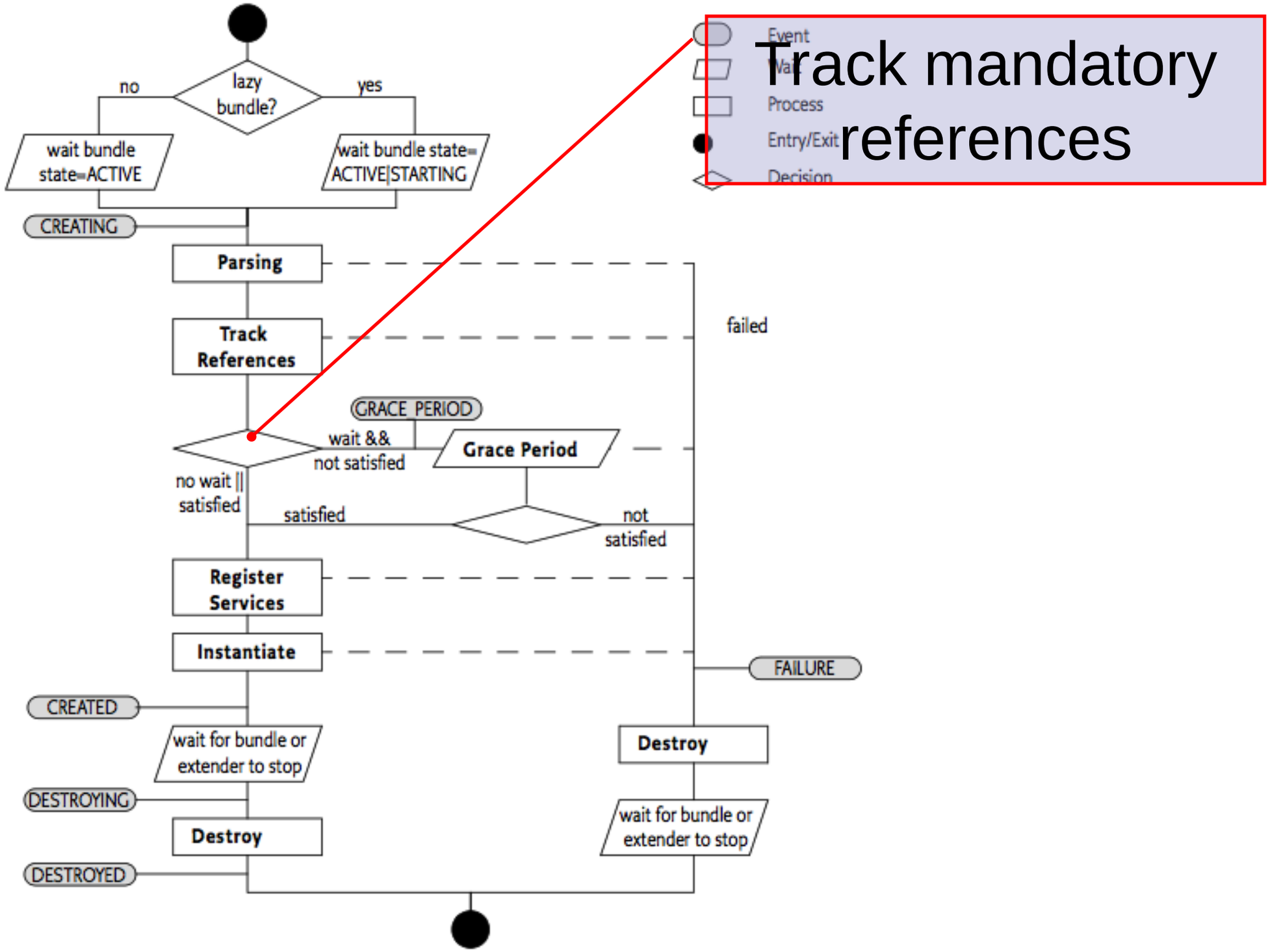

Event
Wait
Process
Entry/Exit
Decision
lazy bundle?
no
yes
wait bundle state=ACTIVE
wait bundle state=ACTIVE|STARTING
CREATING
Parsing
Track References
failed
GRACE_PERIOD
wait && not satisfied
Grace Period
no wait || satisfied
satisfied
not satisfied
Register Services
Instantiate
FAILURE
CREATED
wait for bundle or extender to stop
Destroy
DESTROYING
Destroy
wait for bundle or extender to stop
DESTROYED

Lifecycle



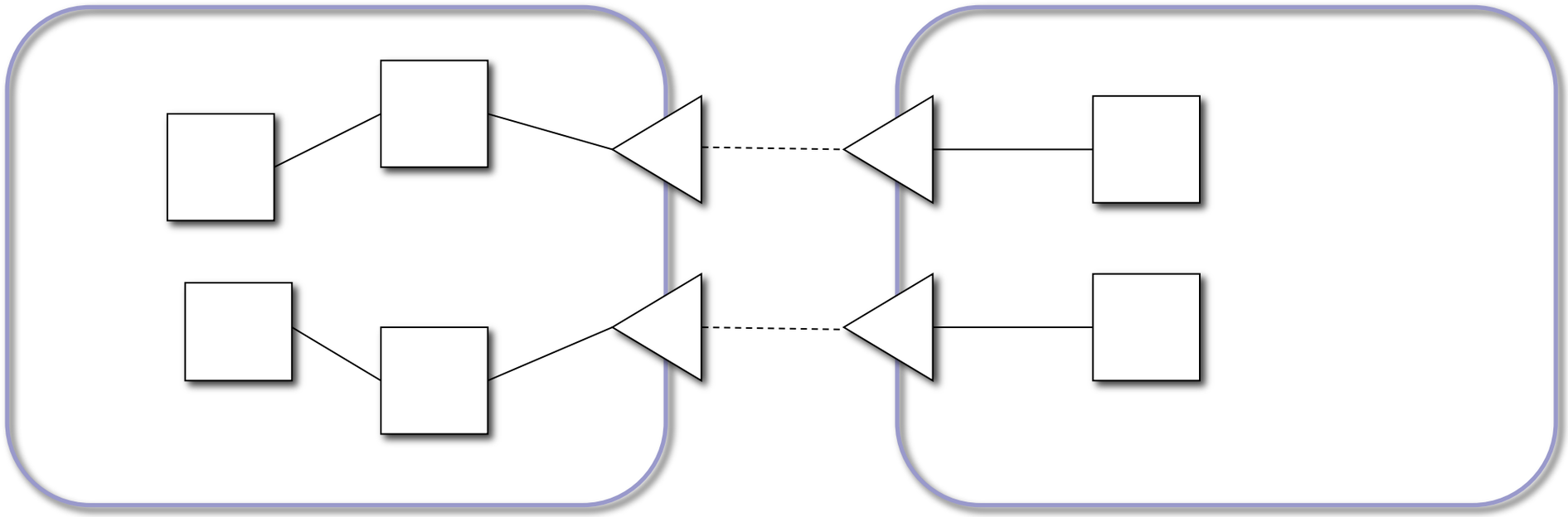
Support for lazy activattion

Lifecycle



Track mandatory references

Lifecycle



Advanced use

- Conversions
- Disambiguation
- Creating custom converters
- Use of **`<idref>`**
- **`<ref-list>`** can be injected as **`List<ServiceReference>`**
- Use of the ranking attribute on **`<service>`**
- Blueprint Events

Conversions

- Arrays, collections, maps
- Primitives / wrapped primitives
- Simple types with a String constructor
- Locale, Pattern, Properties, Class
- JDK 5 Generics
- Custom converters

Disambiguation

- Constructors / factory methods can have multiple overloads

```
public class Bar {
    public Bar(File file) { ... }
    public Bar(URI uri) { ... }
}
```

```
<bean class="foo.Bar">
    <argument type="java.net.URI"
              value="file://hello.txt"/>
</bean>
```

Custom converters

```
<type-converters>
    <bean id="converter1" class=”foo.DateTypeConverter">
        <property name="format" value="yyyy.MM.dd"/>
    </bean>
</type-converters>

<bean class=“...”>
    <property name=“date” value=“2009.06.22” />
</bean>
```

```
public class DateTypeConverter {
    public boolean canConvert(Object fromValue,
                              CollapsedType toType) {
        ...
    }
    public Object convert(Object fromValue,
                          CollapsedType toType) throws Exception {
        ...
    }
}
```

<ref-list> as List<ServiceReference>

```
<ref-list interface="osgi.devcon.User"
          member-type="service-reference" />
```

Other advanced uses

- **`<ref-list>`** as **`List<ServiceReference>`**

```
<ref-list interface="osgi.devcon.User"
          member-type="service-reference" />
```

- Use of **`ranking`** attribute on **`<service>`**

```
<service ref="foo"
         interface="osgi.devcon.User"
         ranking="5" />
```

Use of <idref>

- Prototypes
- Use of the Blueprint API

```
<bean id="bar" class="foo.Bar" scope="prototype">
    <property name="prop" value="val" />
</bean>

<bean class="foo.BarCreator">
    <property name="blueprintContainer"
              ref="blueprintContainer" />
    <property name="id">
        <idref component-id="bar" />
    </property>
</bean>
```

```
Bar bar = (Bar) blueprintContainer.getComponent(id)
```

Blueprint events

- Register listeners
- Blueprint events
  - **`CREATING`**
  - **`CREATED`**
  - **`DESTROYING`**
  - **`DESTROYED`**
  - **`FAILURE`**
  - **`GRACE_PERIOD`**
  - **`WAITING`**

```
public interface BlueprintListener {
    void blueprintEvent(BlueprintEvent event);
}
```

Custom namespaces

- Custom namespace handlers
- Extended namespace
- Config Admin support
- Transaction support

Extended namespace

```
<ext:property-placeholder system-properties="override">
   <ext:default-properties>
       <ext:property name="name" value="value" />
   </ext:default-properties>
</ext:property>

<bean ...>
   <property name="prop" value="${name}" />
</bean>

<reference interface="foo.Bar"
           ext:proxy-method="classes" />
```

Config Admin

- Injection of values from a Configuration

```
<cm:property-placeholder persistent-id="foo.bar" />
   <cm:default-properties>
       <cm:property name="name" value="value" />
   </cm:default-properties>
</cm:property>

<bean ...>
   <property name="prop" value="${name}" />
</bean>
```

Config Admin

- Support for managed properties

```
<bean class="foo.Bar">
   <cm:managed-properties
        persistent-id="foo.bar"
        updated-strategy="component-managed"
        update-method="update" />
</bean>
```

Config Admin

- Support for managed service factories

```
<cm:managed-service-factory
        factory-pid="foo.bar"
        interface="foo.Bar">
   <service-properties>
      <entry key="key1" value="value1" />
   </service-properties>
   <cm:managed-component class="foo.BarImpl" />
</bean>
```

Transactions

- Proxy beans for declarative transactions support

```
<bean class="org.apache.aries.transaction.TestTxBean">
    <tx:transaction method="*" value="Required"/>
</bean>
```

Running outside OSGi

- NoOSGi / Web
- PojoSR

NoOSGi

- No OSGi api / framework needed at all
- No discovery of custom namespace handlers
- Fail when parsing an OSGi related element
- < 300 Ko
- Web module

NoOSGi

```
ClassLoader loader = …
URL[] urls = { … }
BlueprintContainerImpl container =
            new BlueprintContainerImpl(loader, urls);

Foo foo = (Foo) container.getComponentInstance("foo");

container.destroy();
```

NoOSGi Web

```
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">

    <context-param>
        <param-name>blueprintLocation</param-name>
        <param-value>META-INF/test.xml</param-value>
    </context-param>

    <context-param>
        <param-name>blueprintProperties</param-name>
        <param-value>test.properties</param-value>
    </context-param>

    <listener>
        <listener-class>
           org.apache.aries.blueprint.web.BlueprintContextListener
        </listener-class>
    </listener>


</web-app>
```

PojoSR

https://code.google.com/p/pojosr/

- Service Registry
- No module layer
- Flat class loader

Alternatives

DS

- Declarative descriptor
- Set of annotations at build time
- Lightweight

CDI

- Annotations based
- Spec being written in the OSGi Alliance
- Seems promising

Implementations

- Eclipse Gemini Blueprint (RI)
  - Requires the Spring Framework
  - > 2 Mo but provide more features
- Aries Blueprint
  - Clean implementation of Blueprint
  - Much lighter
  - Integrated in Apache Karaf

Conclusion

- Existing alternatives
  - DS, iPojo, Peaberry, CDI
- Strengths of blueprint
  - Familiarity with Spring
  - More powerful Dependency Injection
  - Easily extensible through namespaces