OSGI DECLARATIVE SERVICES

Getting Started with OSGi Declarative Services
Speaker

Dirk Fauth
Research Engineer
Eclipse Committer

Robert Bosch GmbH
Motorstraße 28
70499 Stuttgart

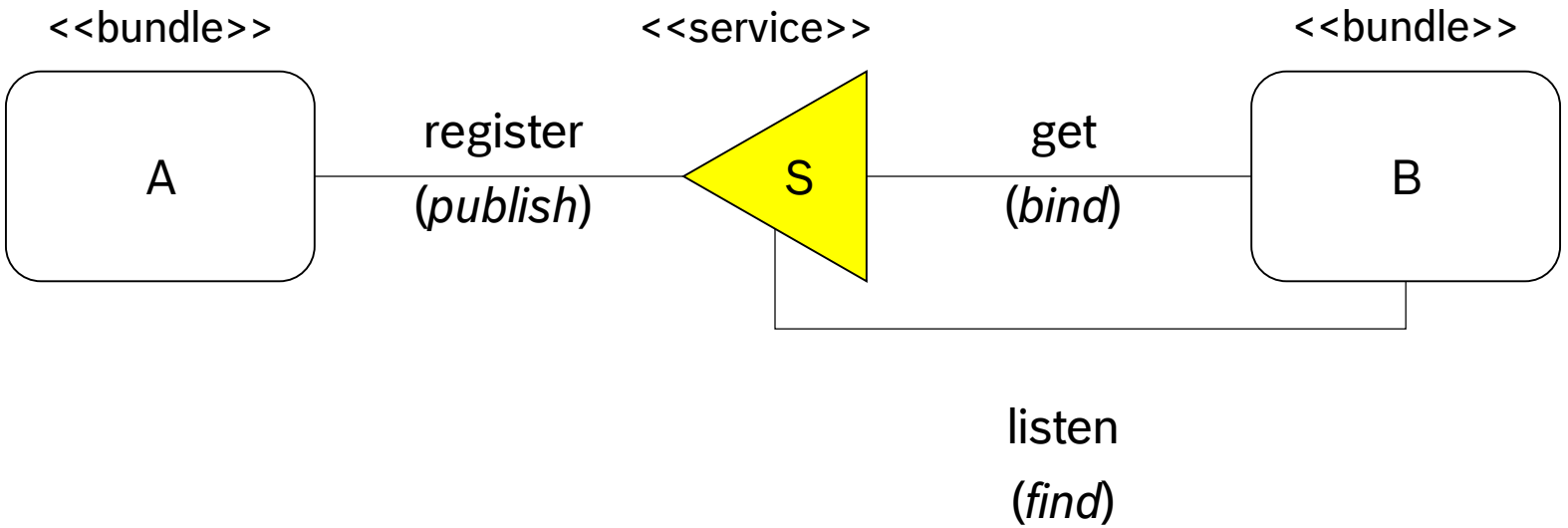
dirk.fauth@de.bosch.com
www.bosch.com
blog.vogella.com/author/fipro/
Twitter: fipro78

BOSCH

OVERVIEW

OSGi Declarative Services
Publish-Find-Bind

- Bundles **register (publish)** services
- Bundles **get (bind)** services
- Bundles **listen (find)** services



OSGi Declarative Services
Components

- **(Service) Component**
  - Java class contained in a bundle
  - Declared via *Component Description*
- **Component Description**
  - XML document to declare a *Service Component*
- **Component Configuration**
  - *Component Description* that is parameterized with component properties
  - Tracks the component dependencies and manages the component instance
- **Component Instance**
  - Instance of the component implementation class
  - Created when a *Component Configuration* is activated
  - Discarded when the *Component Configuration* is deactivated

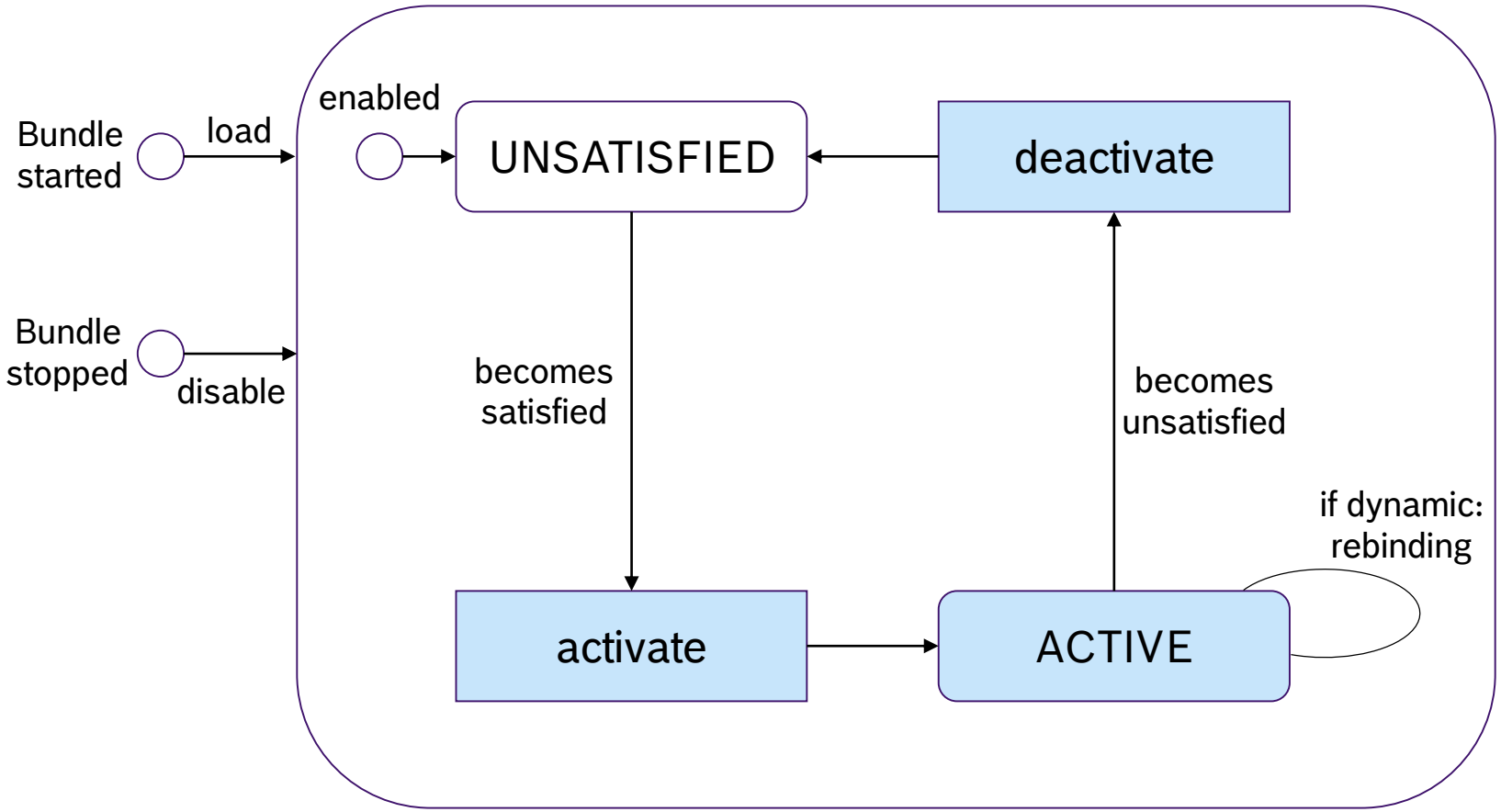
OSGi Declarative Services
References

- **References**
  - The definition of dependencies to other services.
- **Target Services**
  - The services that match the reference interface and target property filter.
- **Bound Services**
  - The services that are bound to a *Component Configuration*.
- **Access Strategies**
  - Method injection *(Event Strategy)*
  - Field injection *(Field Strategy)* (DS 1.3)
  - Constructor injection (DS 1.4)
  - Lookup Strategy

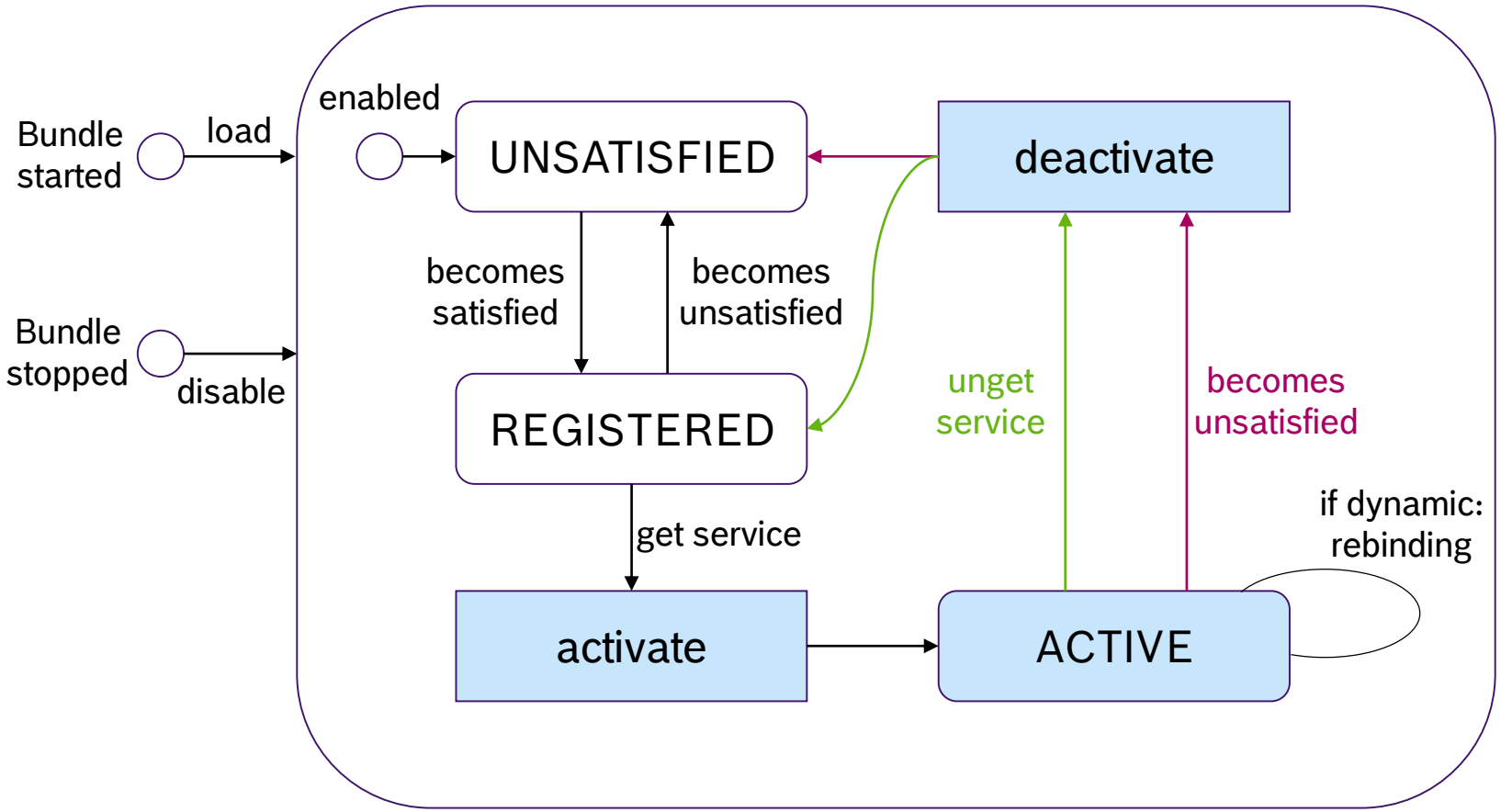
OSGi Declarative Services
Component Types

- **Delayed Component**
  - Activated when the service is requested
  - Needs to specify a service
- **Immediate Component**
  - Activated as soon as all dependencies are satisfied
  - Does not need to specify a service
- **Factory Component**
  - Creates and activates Component Configurations

OSGi Declarative Services
Component Lifecycle – Immediate



OSGi Declarative Services
Component Lifecycle – Delayed



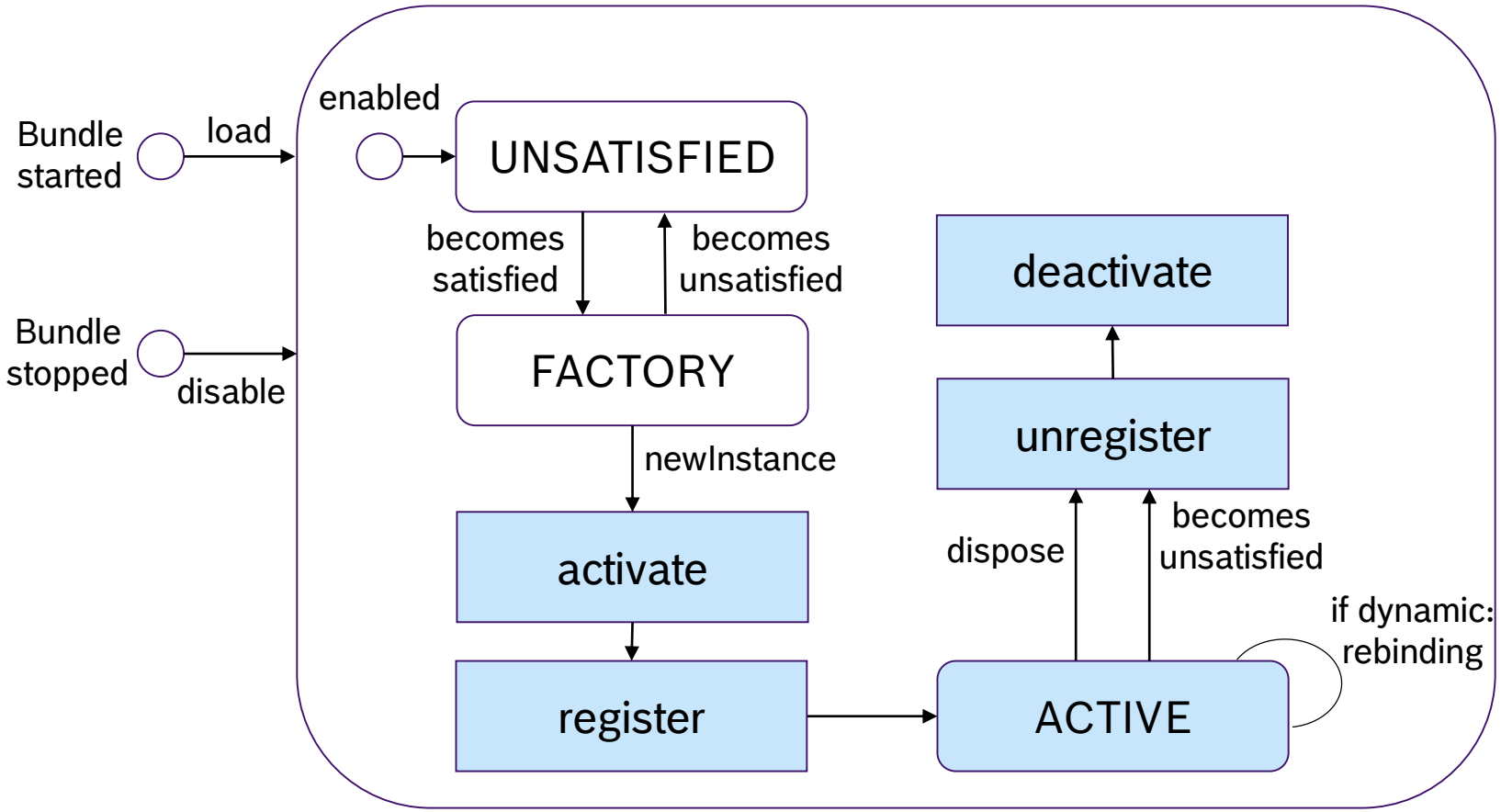
OSGi Declarative Services
Component Lifecycle – Activation

- Activation consists of the following steps:
  1. Load the component implementation class
  2. Create the component instance and component context
  3. Bind the target services
  4. Call the activate method if present

- For **Delayed Components** the load time is moved to the first request (including reference bindings)

(see Declarative Services Specification Version 1.4 – 112.5.6 Activation)

OSGi Declarative Services
Component Lifecycle – Factory



IMPLEMENT & PUBLISH

BOSCH

OSGi Declarative Services
Service API

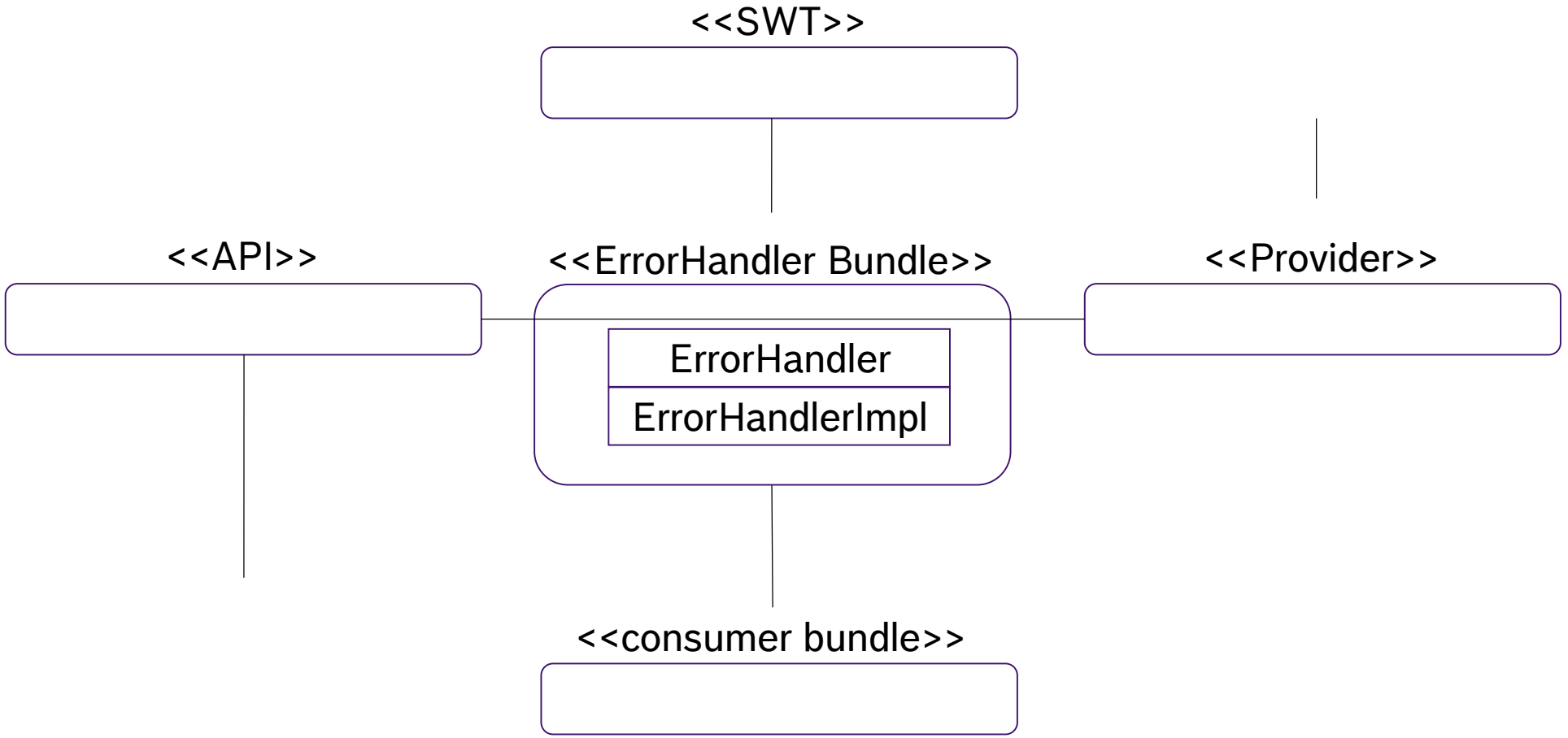
- Create a bundle for the service API (e.g. *org.fipro.modifier.api*)
- Create the service interface

```
public interface StringModifier {
    String modify(String input);
}
```

→ **Make the service implementation exchangeable**

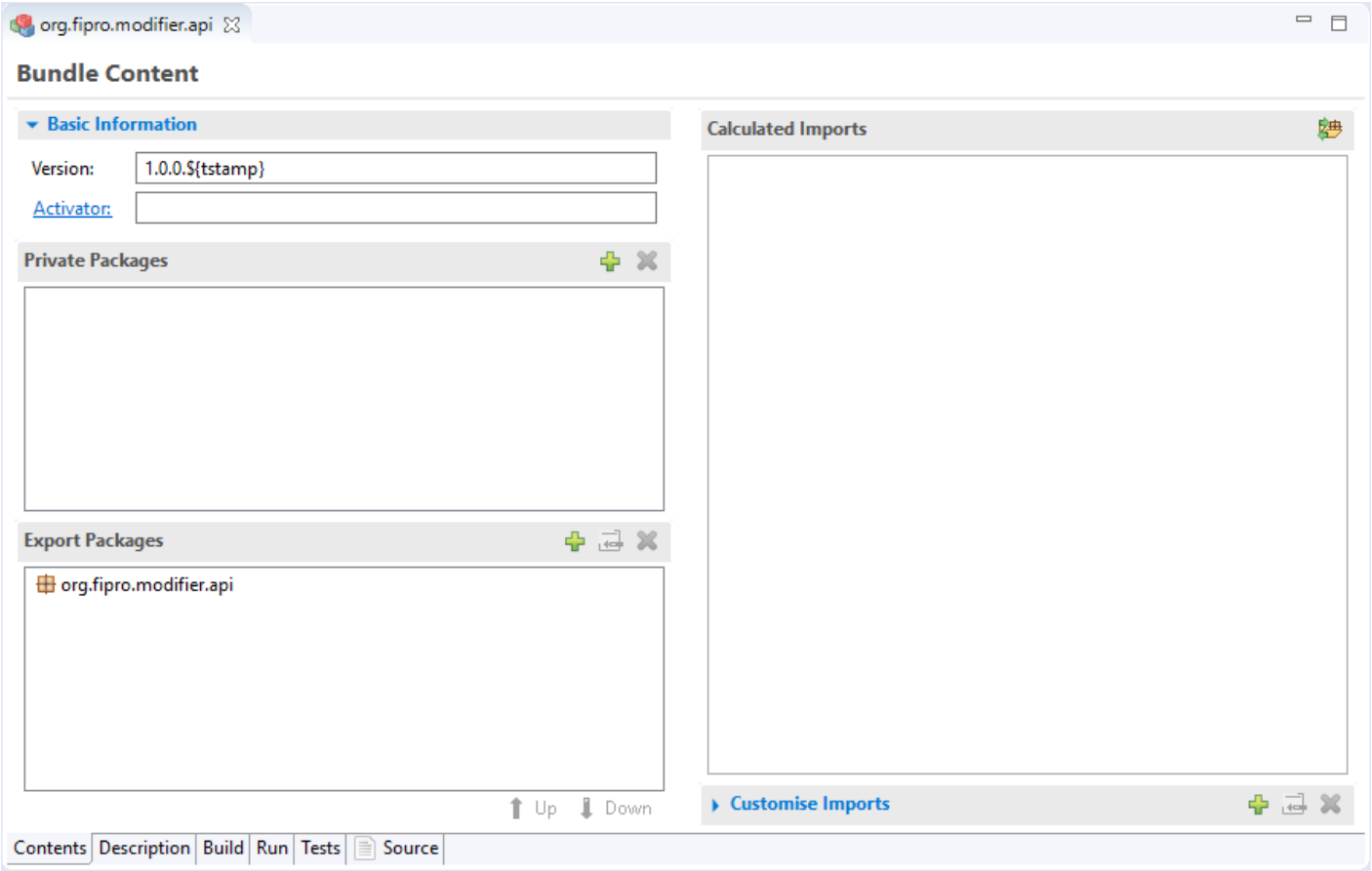
→ **Clean dependency hierarchy**

OSGi Declarative Services
Service API



OSGi Declarative Services
Metadata

- Create/Modify *.bnd* file
- Describe OSGi meta-data
- *MANIFEST.MF* is generated



OSGi Declarative Services
Service Provider

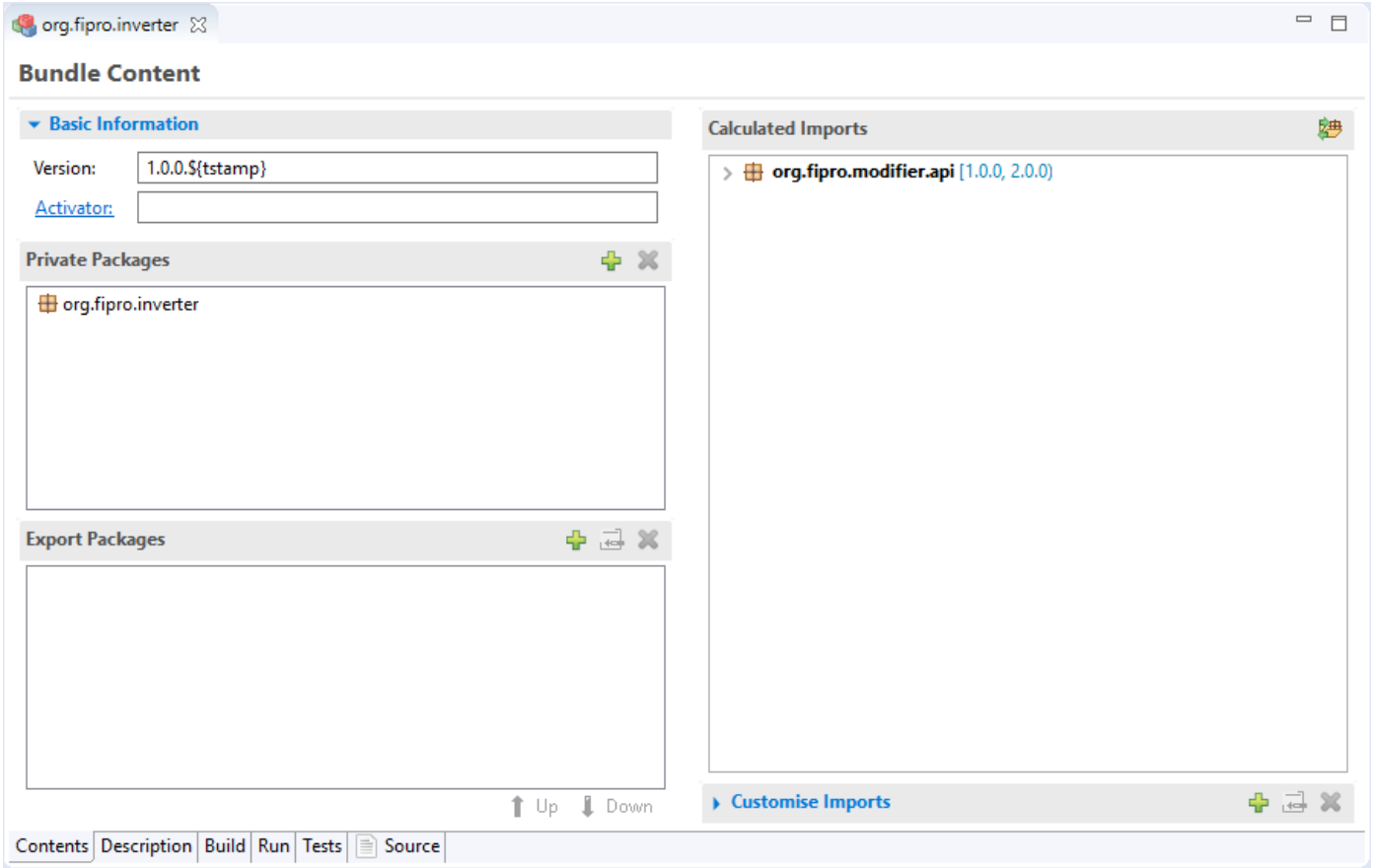
- Create a bundle for the service implementation (e.g. *org.fipro.inverter*)
- Create the service implementation

```
@Component
public class StringInverter implements StringModifier {

    @Override
    public String modify(String input) {
        return new StringBuilder(input).reverse().toString();
    }
}
```

OSGi Declarative Services
Metadata

- Add API bundle to the *Build Path*
- Add the package to *Private Packages*



OSGi Declarative Services
@Component

- **Annotation to mark a Java class as a *Service Component***
- **Generates**
  - *Component Description* XML file
  - Additional header in MANIFEST.MF
    - `Service-Component` header
    - `Provide-Capability` header for `osgi.service`
    - `Require-Capability` header for `osgi.extender` (*DS 1.3*)

OSGi Declarative Services
Capabilities

- Capability = non-code dependency
- `osgi.extender=osgi.component`

```
Require-Capability: osgi.extender;
 filter:="(&(osgi.extender=osgi.component)(version>=1.3)(!(version>=2.0)))"
```

  - added to spec with DS 1.3
  - Equinox DS adapted for DS 1.2 with Eclipse Neon

- `osgi.service`
  - specify the provided service implementations

```
Provide-Capability: osgi.service;
 objectClass:List<String>="org.fipro.inverter.StringInverter"
```

OSGi Declarative Services
@Component

Type Element	Default	Type Element	Default
configurationPid	full qualified class name of the component	**servicefactory** *(depr. in 1.3)*	`false`
configurationPolicy	optional	**xmlns**	lowest DS XML namespace that supports used features
enabled	`true`		
factory	empty String	**reference** *(since 1.3)*	empty
immediate	`false` if factory or service `true` otherwise	**scope** *(since 1.3)*	`ServiceScope.SINGLETON`
name	full qualified class name of the component		
properties	empty	**factoryProperties** *(since 1.4)*	empty
property	empty	**factoryProperty** *(since 1.4)*	empty
service	full qualified class names of all directly implemented interfaces		

OSGi Declarative Services
@Activate / @Modified / @Deactivate

- **Component life cycle methods**
- **Method parameters (*Activation Objects*)**
  - `ComponentContext`
  - `BundleContext`
  - `Map<String, ?>`
    Map containing component properties
  - *<Component Property Type> (DS 1.3)*
    Type safe access to component properties
  - `int / Integer` for (`@Deactivate`)
- DS 1.4 - `@Activate` on fields and constructors

```
@Activate
private void activate(
    ComponentContext c,
    BundleContext b,
    Map<String, ?> properties) {

    //do some initialization stuff
}
```

OSGi Declarative Services
Service Consumer

- Create a bundle for the service consumer (e.g. *org.fipro.modifier.command*)
- Create the service consumer implementation

```
@Component(property= {"osgi.command.scope:String=fipro",
                      "osgi.command.function:String=modify"},
    service=StringModifierCommand.class
)
public class StringModifierCommand {

    private StringModifier modifier;

    @Reference
    void bindStringModifier(StringModifier modifier) { this.modifier = modifier; }

    public void modify(String input) { System.out.println(modifier.modify(input)); }
}
```

OSGi Declarative Services
@Reference

- **Specify dependency on other services**
- **Resolving references is required to satisfy a component**
  (if the reference is not optional)
- **Different techniques for accessing services**
  - *Method injection (Event Strategy)*
  - *Field injection (Field Strategy) (DS 1.3)*
  - *Constructor injection (DS 1.4)*
  - *Lookup Strategy*

OSGi Declarative Services
@Reference

- ***Method injection*** – using event methods for *bind/updated/unbind*

```
@Component(...)
public class StringModifierCommand {

    private StringModifier modifier;

    @Reference
    void bindStringModifier(StringModifier modifier) { this.modifier = modifier; }

    void updatedStringModifier(StringModifier modifier, Map<String, ?> properties) { //do something }

    void unbindStringModifier(StringModifier modifier) { this.modifier = null; }

    public void modify(String input) { System.out.println(modifier.modify(input)); }
}
```

OSGi Declarative Services
@Reference - Event Method Parameter

- `ServiceReference`
- <service type>
- <service type> + `Map<String, ?>`

With DS 1.3

- `ComponentServiceObjects`
- Different variations of the parameter list

OSGi Declarative Services
@Reference

- ***Field injection*** (*DS 1.3*) – using instance fields

```
@Component(...)
public class StringModifierCommand {

    @Reference
    private StringModifier modifier;

    public void modify(String input) {
        System.out.println(modifier.modify(input));
    }
}
```

OSGi Declarative Services
@Reference

- ***Constructor injection*** (*DS 1.4*)

```
@Component(...)
public class StringModifierCommand {
    private StringModifier modifier;

    @Activate
    public StringModifierCommand(@Reference StringModifier modifier) {
        this.modifier = modifier;
    }

    public void modify(String input) {
        System.out.println(modifier.modify(input));
    }
}
```

OSGi Declarative Services
@Reference

- ***Lookup Strategy*** *(DS 1.2)* – lookup everytime needed, do not store

```
@Component(...)
public class StringModifierCommand {

    private ComponentContext context;
    private ServiceReference<StringModifier> reference;

    @Activate
    void activate(ComponentContext context) { this.context = context; }

    @Reference
    void setStringModifier(ServiceReference<StringModifier> reference) { this.reference = reference; }

    public void modify(String input) {
        StringModifier modifier = (StringModifier) context.locateService("StringModifier", reference);
        System.out.println(modifier.modify(input));
    }
}
```

OSGi Declarative Services
@Reference

- ***Lookup Strategy*** *(DS 1.3)* – lookup everytime needed, do not store

```
@Component(...
    reference=@Reference(name="modifier", service=StringModifier.class)
)
public class StringModifierCommand {

    private ComponentContext context;

    @Activate
    void activate(ComponentContext context) { this.context = context; }

    public void modify(String input) {
        StringModifier modifier = (StringModifier) context.locateService("modifier");
        System.out.println(modifier.modify(input));
    }
}
```

OSGi Declarative Services
@Reference

- ***Lookup Strategy*** *(DS 1.4)* – lookup everytime needed, do not store

```
@Component(...
    reference=@Reference(name="modifier", service=StringModifier.class)
)
public class StringModifierCommand {

    @Activate
    ComponentContext context;

    public void modify(String input) {
        StringModifier modifier = (StringModifier) context.locateService("modifier");
        System.out.println(modifier.modify(input));
    }
}
```

OSGi Declarative Services
@Reference

Type Element	Default	Type Element	Default
cardinality	• 1:1 for event methods • 1:1 for non-collection fields, 0..n for collections	**bind** *(since 1.3)*	the name of the annotated method or empty
name	• bind event method name without bind prefix • name of the field	**field** *(since 1.3)*	the name of the annotated field or empty
policy	STATIC	**fieldOption** *(since 1.3)*	REPLACE
policyOption	RELUCTANT	**scope** *(since 1.3)*	BUNDLE
service	full qualified class name of the referenced service		
target	empty String	**parameter** *(since 1.4)*	0
unbind	unbind<name> unset<name> remove<name>	**collectionType** *(since 1.4)*	CollectionType.SERVICE
updated	updated<name> if such a method exists		

OSGi Declarative Services
DS 1.4 – Component Property Types Update

```
@Component(property= {"osgi.command.scope:String=fipro",
                      "osgi.command.function:String=modify"},
    service=StringModifierCommand.class
)
public class StringModifierCommand { }
```

```
@ComponentPropertyType
public @interface GogoCommand {
    String osgi_command_scope() default "";
    String osgi_command_function() default "";
}
```

```
@Component(service = StringModifierCommand.class)
@GogoCommand(osgi_command_scope = "fipro", osgi_command_function = "modify")
public class StringModifierCommand {}
```

OSGi Declarative Services
DS 1.4 – Additional Goodies

- Standard *Component Property Types* for standard service properties
  - `@ServiceDescription (service.description)`
  - `@ServiceRanking (service.ranking)`
  - `@ServiceVendor (service.vendor)`
  - `@ExportedService` (remote service properties of an exported service)
- Injection of a `Logger`

```
@Component
@ServiceRanking(10)
public class StringInverter implements StringModifier {

   @Reference(service=LoggerFactory.class)
   Logger logger;
   …
}
```

OSGi Declarative Services
Further information

- OSGi Compendium Release 7 Declarative Services Specification
  https://osgi.org/specification/osgi.cmpn/7.0.0/service.component.html

- OSGi Blog
  https://blog.osgi.org/2018/03/osgi-r7-highlights-declarative-services.html

- Blog posts
  Getting Started with OSGi Declarative Services
  http://blog.vogella.com/2016/06/21/getting-started-with-osgi-declarative-services/
  OSGi Component Testing
  http://blog.vogella.com/2016/07/04/osgi-component-testing/
  Configuring OSGi Declarative Services
  http://blog.vogella.com/2016/09/26/configuring-osgi-declarative-services/
  Control OSGi DS Component Instances
  http://blog.vogella.com/2017/02/13/control-osgi-ds-component-instances/
  Control OSGi DS Component Instances via Configuration Admin
  http://blog.vogella.com/2017/02/24/control-osgi-ds-component-instances-via-configuration-admin/


eclipsecon
Europe 2019
OSGi
Community Event
2019
LUDWIGSBURG, GERMANY | OCTOBER 21 - 24, 2019
EVALUATE THE SESSIONS
Sign in and vote using the conference app or eclipsecon.org
-1 0 +1