e4 workshop

Tom Schindl <tom.schindl@bestsolution.at>

OSGi

- Standardized by OSGi Alliance
- Multiple Implementations
  - Equinox from Eclipse.org (used by us)
  - Felix from Apache Foundation
  - ...
- Meta-Information stored in MANIFEST.MF

OSGi Bundle

- Layer above Classes and Packages
  - defines API and none API packages by **explicitly** exporting packages
- Bundles define dependencies between each other
  - Require-Bundle
  - Import-Package

Bundle-Lifecycle

- Bundle has a lifecycle
  - INSTALLED
  - RESOLVED
  - ACTIVE

MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: %pluginName
Bundle-SymbolicName: ch.sbb.address.model;singleton:=true
Bundle-Version: 1.0.0.qualifier
Bundle-ClassPath: .
Bundle-Vendor: %providerName
Bundle-Localization: plugin
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Export-Package: ch.sbb.address.model.addressbook,
 ch.sbb.address.model.addressbook.impl,
 ch.sbb.address.model.addressbook.util
Require-Bundle: org.eclipse.core.runtime,
 org.eclipse.emf.ecore;visibility:=reexport
Bundle-ActivationPolicy: lazy
```

Exported API

Dependency

Services

- Always 3 stackholders involved
  - Definition (always one)
  - Implementation (multiple possible)
  - Consumer (mulitple very likely)
- Consumer should NEVER know about the implementor (decoupleing)

Declarative Services

- Implementor Bundle
  - Must have „BundleActivation-Policy: lazy“
  - Service registration done by XML
  - Importance defined in XML with „service.ranking“ property
- equinox.ds must be part of the launched bundles

Sample XML

Implementation

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/
v1.1.0" name="ch.sbb.osgi.service.console.consolegreet">
   <implementation class="....ConsoleGreetingService"/>
   <service>
      <provide interface="....GreetingService"/>
   </service>
   <reference bind="setTranslate"
      cardinality="1..1"
      interface="....Translate" name="Translate"
      policy="static" unbind="unsetTranslate"/>
   <property name="service.ranking" type="Integer"
     value="1001"/>
</scr:component>
```

Service Interface

Dependency Injection

- General coding pattern in webspace since Spring
- Allows to better decouple and test things
- General rule: „Do not reach out to get stuff“

DI „Bean“

```
public class ApplicationDI {
  @Inject
  @Named("fieldPerson")
  @Optional
  private Person fieldPerson;

  @Inject
  public ApplicationDI(Person person, IEclipseContext context) {

  }

  @Inject
  public void setGreetingService(GreetingService greetingService) {

  }

  @PostConstruct
  void initDone(Person person) {
  }

  @PreDestroy
  void destroyingObject() {
    System.err.println("Destroying");
  }
}
```

Field Injection

Constructor Injection

Method Injection

Post Construct

Before Destruction

Dependency Injection

- Follows JSR 299 and JSR 330
  - @Inject, @PostConstruct
- Custom annotations
  - @Optional: null is allowed

Dependency Injection

- IEclipseContext
  - Hierachical Map to store values for injection
  - Final Map is the OSGi-Service-Registry => all OSGi-Services can be injected

Dependency Injection

- Instance creation with ContextInjectionFactory#make

```
IEclipseContext diContext =
EclipseContextFactory.getServiceContext(Activator.getContext());

Person p = new Person("Tom","Schindl");
diContext.set(Person.class, p);
diContext.set("2ndperson", new Person("Hans", "Mustermann"));

ApplicationDI di =
ContextInjectionFactory.make(ApplicationDI.class, diContext);
```

Dependency Injection

- Method calling in ContextInjectionFactory#invoke

```
IEclipseContext ctx =
EclipseContextFactory.getServiceContext(Activator.getContext());
ctx.set("exchangedValue", Double.valueOf(2.0));

ExchangeApp app = ContextInjectionFactory.make(ExchangeApp.class, ctx);
ContextInjectionFactory.invoke(app, Execute.class, ctx);
```

Dependency Injection

- Events via Injection
  - @EventTopic
  - @UIEventTopic - synchronized to event thread
- Event delivery through IEventBroker

Dependency Injection

- Receiving

```
public class ExchangeInfoLogger {
    @Inject
    @Optional
    void currenyEventCallback(@EventTopic("ch/sbb/currency/*") Double value) {
        System.out.println("DI-Event-Handler: " + value);
    }
}
```

- Sending

```
public class ExchangeApp {
    @Inject
    IEventBroker broker;
    @Execute
    public void exchange(@Named("exchangedValue") Double value) {
        broker.send("ch/sbb/currency/exchange", value);
    }
}
```

Dependency Injection

- Preferences
  - @Preference

```
public class ExchangeInfoLogger {
    @Inject
    void myPreference(@Preference(nodePath="mybundle",value="myPref") String value) {
        System.out.println("The preference: " + value);
    }
}
```

Dependency Injection

- ContextFunction
  - Create values on request / factory like
  - Registration through DS

Dependency Injection

- DS-Registration

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
   name="ch.sbb.address.core.addressbookfunction">
   <implementation class="ch.sbb.address.core.LoadAddressBook"/>
   <service>
      <provide interface="org.eclipse.e4.core.contexts.IContextFunction"/>
   </service>
   <property name="service.context.key" type="String"
       value="ch.sbb.address.model.addressbook.AddressBook"/>
</scr:component>
```

Service-Interface

Key in Context

Dependency Injection

- Custom annotations - Definition

```
@Qualifier
@Documented
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
public @interface Translation {

}
```

Dependency Injection

- Custom annotations - Registration through DS

Service-Class

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
   name="org.eclipse.e4.tools.services.translationsupplier">
   <implementation
     class="org.eclipse.e4.tools.services.impl.TranslationObjectSupplier"/>
   <service>
      <provide interface="org.eclipse.e4.core.di.suppliers.ExtendedObjectSupplier"/>
   </service>
   <property name="dependency.injection.annotation" type="String"
     value="org.eclipse.e4.tools.services.Translation"/>
</scr:component>
```

Annotation-Class

Application Model

- **All** informations in are store in the model
- Model defined through EMF
- Default serialization in XML (XMI)

Application Model

- Main Types
  - UI-Structure
    - Window, PerspectiveStack, Perspective, PartSashContainer, PartStack
    - ToolBar, ToolItem, Menu, MenuItem
  - Action
    - Addon, Handler, Command, KeyBinding

Application Model

- Connection between Java and Model
  - URI: bundleclass://$bundlename/$classname

```
<?xml version="1.0" encoding="UTF-8"?>
<application:Application elementId="org.eclipse.e4.ide.application">

  <handlers contributionURI="bundleclass://mybundle/MyHandler" />

</application:Application>
```

Application Model

- Parts
  - Leafes where control handed over too you

```
public class ExchangeReceiver {
  @PostConstruct
  public void init(Composite parent) {
  }

  @Inject
  @Optional
  void valueExchanged(@UIEventTopic("ch/sbb/currency/e4/app") final Double value) {
  }

  @PreDestroy
  void destroy() {
  }
}
```

Application Model

- Handler
  - Do not use @Inject
  - Annotate method with @Execute

```
public class OpenReceiverFromDescriptor {
  @Execute
  public void open(MApplication application,
    EModelService modelService,
    EPartService partService) {
  }
}
```

Application Model

- Contribution to application model
  - through fragment
  - through processor

Application Model

- Fragment contribution
  - use extension point org.eclipse.e4.workbench.model
  - fragment-element pointing to e4xmi-fragment

```
<?xml version="1.0" encoding="ASCII"?>
<fragment:ModelFragments>
  <fragments xsi:type="fragment:StringModelFragment"
     featurename="trimBars"
     parentElementId="mainWindow">
  </fragments>
</fragment:ModelFragments>
```

Attribute-name

Element-Id

Application Model

- Processor contribution
  - use extension point org.eclipse.e4.workbench.model
  - processor-element pointing to java-class

```
public class ModelProcessor {

  @Execute
  public void processModel(MApplication application, EModelService modelService) {
  }

}
```

Application Model

- Translations in model
  - prefix value with %
  - put key behind % in bundle.properties

e4 bridge

- Write part with e4 DI programming model
- subclass DIViewPart and pass part

```
public class View extends DIViewPart {
   public static final String ID = "ch.sbb.address.app38.view";

   public View() {
      super(PersonList.class);
   }

}
```