Migration to E4

Eclipse Con France 2016

8th June 2016
# Table des matières

## I - Migration to E4

A. Presentation................................................................. 6
B. Migration to E4.............................................................. 6
C. Migration tooling............................................................ 9
D. Model Fragments and Processors....................................... 14
E. Context concerns............................................................ 17
F. Extension Migration........................................................ 19
G. Resources........................................................................ 25
This talk will explain:

- some general issues about migration
- the key points of your migration strategy
- tooling that could be used
- the model fragments and processors
- how to migrate some standard extension
A. Presentation

OPCoach

Image 1

➢ Training: RCP, E4, Modeling, Build, given in French, English and ... Spanish
➢ Consulting
➢ Recruitment service to link companies and job applicants
➢ Web site: http://www.opcoach.com/
➢ Twitter: @OPCoach_Eclipse, @OPCoach_Job

B. Migration to E4

The technical reasons for using E4 as a platform?

➢ The official Eclipse RCP Runtime is 4.X since June 2012
➢ Application model is dynamic and platform agnostic (SWT, Java FX...) thanks to POJOs
➢ Injection is pretty cool, reduces the amount of code and simplifies dramatically testing
➢ Eclipse 4 event notification system (IEventBroker) is very concise and easy to use with injection
➢ The UI customization is easier thanks to:
  ➢ CSS
  ➢ renderers that can be overiden
➢ E4 spies will help to develop your application
➢ Your application will still live several years:
  ➢ it will provide an opportunity to refactor and decouple your components
  ➢ the application will be better every year with the new runtime version

1 - http://www.opcoach.com/en
The global prerequisites

Be sure of your team's knowledge:

➢ do they know Eclipse 3 and Eclipse 4?
➢ do they know the application??
➢ do they know how to migrate?
➢ are they really involved in the migration process?

The global prerequisites

Create a migration strategy

➢ Identify the features you want to migrate and the reasons why
  ➢ Evaluate the UI E3 dependencies
  ➢ Evaluate the lifetime of your component
  ➢ Prefer the components that have no tests to test them now!
➢ Be aware that you will may not be able to migrate the entire application!

Big picture of 3.X application with 4.X runtime
Big picture of what we should do

Step by Step migration principle

The technical prerequisites
To prepare your E3 plugin/application migration you have to:

➢ ensure the application can be launched using the compatibility layer
  ➢ org.eclipse.equinox.ds
  ➢ org.eclipse.equinox.event
  ➢ org.eclipse.equinox.util
  ➢ org.eclipse.e4.ui.workbench.addons.swt
➢ clearly separate core and ui plug-ins
➢ have packages for each entities to migrate: views, handlers, etc...
➢ remove the org.eclipse.ui internal package uses and imports

Migration steps / Core
To migrate a core plugin you must:

➢ do nothing!
➢ because there are no dependencies to org.eclipse.ui

Migration steps / UI
To migrate an UI plug-in, you must:

➢ move the ui E3 extensions to a model fragment (or to the application model)
➢ migrate the relevant code
➢ remove all E3 extensions
➢ remove the `org.eclipse.ui` dependency when it is not used anymore
➢ add the `jface` dependency and others instead
➢ for the RCP main plugin (containing application), create the source application model. Then, once all the plug-ins have been migrated, it is possible to remove the compatibility layer.

**Practical advices**

➢ Create a xxx.e4.xxx package to put the migrated class, in current migrated plug-in
➢ for instance : xxx.e4.handlers or xxx.e4.parts
➢ Copy the E3 class and its dependencies in this package and keep the names
➢ Set the E3 classes as ‘deprecated’
➢ Annotate with a //E34 comment the current migrated areas when they are not finished
➢ Remove the old E3 packages when the migration is finished

These tips help maintain existing plugins and the build process

**Displaying the //E34 tasks**

It is possible to display the //E34 comments in the task view:

➢ open the ‘Tasks’ view
➢ add a E34 tag in the preference page of Java->Compiler->Task

E34 tasks

**C. Migration tooling**

**E4 Spies**

➢ The E4 spies are useful to develop an E4 application
➢ They help in browsing the application model, injection contexts, events, css....
➢ It is possible to write its own spy for any specific data
➢ Eclipse Mars does not include the E4 spies
➢ They will be soon delivered by default
To install them, upload the update site from:

- http://download.eclipse.org/e4/downloads

Download the zipped update site and install it:

- Menu Help -> Install New Software..
- 'Add..', 'Archive..'

Then select 'All Spies':

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse 4 - All Spies</td>
<td></td>
</tr>
<tr>
<td>Eclipse 4 - All Spies (Incubation)</td>
<td>0.1.0.v20150508-0800</td>
</tr>
<tr>
<td>Eclipse 4 - Bundle Spy</td>
<td></td>
</tr>
<tr>
<td>Eclipse 4 - Context Spy</td>
<td></td>
</tr>
<tr>
<td>Eclipse 4 - CSS Spy</td>
<td></td>
</tr>
<tr>
<td>Eclipse 4 - Event Spy</td>
<td></td>
</tr>
<tr>
<td>Eclipse 4 - Model Spy</td>
<td></td>
</tr>
<tr>
<td>Eclipse 4 - Preference Spy</td>
<td></td>
</tr>
</tbody>
</table>

Image 2 E4 tooling

**Using the spies**

There are 3 different ways to open the spy window:

- use one of the shortcut (Alt Shift F4 to Alt Shift F10 for instance) depending on the installed spies
- look for "spy" in the quick access field
- use the Window->Spies menu:

---

2 - http://download.eclipse.org/e4/downloads
It will open a specific E4 Spies Window with a toolbar to display each spy.
For instance the **Model Spy**:

![E4 Spies Window](image)

**e4 Spies Window**

*A tooling to help to evaluate the migration cost*

- OPCoach developed a specific statistic view dedicated to migration
- This plugin is available on github: [http://opcoach.github.io/E34MigrationTooling/](http://opcoach.github.io/E34MigrationTooling/)
- It is delivered under EPL license and it is free
➢ Select the projects in the workspace and get some statistics about used UI extension points:

Migration Stat View

**An evaluation form to check your migration**

➢ OPCoach provides a form to help you to evaluate the work


---

**The plugin templates with model fragments**

With Neon, it is now possible to create pure E4 plugins using model fragments.

➢ a plugin with a pure SWT E4 view
D. Model Fragments and Processors

Introduction

You can contribute to an application model by using two mechanisms:

- **a model fragment**: with the ID or xpath of model objects
- **a processor**: with a piece of code modifying the injected application

Model fragment

- The model fragment adds content to an existing application model
- To create a model fragment,
  - use the model fragment wizard (Ctrl N + 'fragment')
  - extract a piece of model into a fragment (contextual menu on application model editor)

Application fragment

It is possible to add any contribution to any object

- just select the ID of the object
- then select the feature to be populated
then add a content

**Application fragment**

If you contribute on the top level application, you can use:

- the ID of the application
- the ID of the legacy E4 application: `org.eclipse.e4.legacy.ide.application`
- the `xpath:` to get any application whatever its ID (see bug #437958)
  - This is the best practice for the top level contributions
  - Example for the spy fragment:

![Image 3](attachment:image.png)

**Model fragment**

Don't forget to declare the fragment in an extension (`org.eclipse.e4.workbench.model`)

![Image 4 Model Fragment](attachment:image.png)
**Processor declaration**

- The processor is used when the object's ID is not known (application for instance)
- The application is received using injection so as to be modified
- It must be declared in the `org.eclipse.e4.workbench.model` extension using a processor parameter:

```java
public class SampleE4Processor {
    @Execute
    public void process(MApplication application, IModelService modelService) {
        // Just create a command and add it in the application
        MCommand command = modelService.createElement(MCommand.class);
        command.setId("id.of.my.command");
        command.setCommandName("Launch My Command");
        String contributorURI = "platform://plugin/" + FrameworkUtil.getBundle(getClass()).getSymbolicName();
        command.setContributorURI(contributorURI);
        command.setDescription("A simple command added in application");
        application.getCommands().add(command);
    }
}
```

**Extension for a processor**

**Processor code**

- The processor code is a POJO with a `@Execute` annotation
- The method receives the application and needed services as fields or parameters
- Use the `modelService.createElement` method to create instances
E. Context concerns

Introduction
This presentation is not a course on injection. It has already been presented in different talks and articles. Refer to this articles to be aware of this powerful mechanism.

➢ Talk in Boston:

➢ Tutorial about injection:
http://eclipsesource.com/blogs/tutorials/eclipse-4-e4-tutorial-part-4-dependency-injection-basics/

➢ Eclipse 4 context usage:
http://www.vogella.com/tutorials/Eclipse4ContextUsage/article.html

➢ Eclipse Wiki

E4 Injection

Principles of E4 injection:
➢ The injector gathers hierarchically all common objects
➢ Listeners and initializations are simplified:
  ➢ Methods annotated with @Inject are called automatically if a parameter changes in the context
  ➢ Fields annotated with @Inject are automatically initialized if the value changes in the context
➢ Allows to have a framework independent of an external library (UI Agnostic)
➢ Simplify unit tests
➢ Example for the selection management:

Usage of injection for the selection

Just receive the selection object in the expected type and you will be notified!

```java
/** This method will be invoked only if current selection is a Rental instance */
@Inject @Optional
public void receiveSelection(@Named(IServiceConstants.ACTIVE_SELECTION) Rental r) {
    setRental(r);
}
```

Get the selection

The context spy to explore your contexts

4 - http://eclipsesource.com/blogs/tutorials/eclipse-4-e4-tutorial-part-4-dependency-injection-basics/
Open the spy window with the shortcut Alt Shift F10:

Nodes containing the filter are displayed in blue color

**E3 and E4 context sharing**

During a migration different use cases are possible:
- you define a value in E3 code and you want to publish it for E4 code
- you define a value in E4 code and you need to reuse it in E3 code

**Sharing data from you E3 code**

This use case happens when:
- you have still E3 plugins that are not migrated or will not migrate
- these plugins defines instances that should be stored in the context

It is possible to get the different E4 contexts:
- the OSGi context (this is the root context)
- the Application context
- the window context
Use this code to fill the context from E3 code:

```java
public void getContextMenuFromE3Code()
{
    // Get the OSGi global E4 context 
    Bundle e4Bundle = Platform.getBundle("org.eclipse.e4.ui.workbench");
    if (e4Bundle != null)
    {
        BundleContext e4BundleContext = e4Bundle.getBundleContext();
        IEclipseContext osgiCtx = EclipseContextFactory.getServiceContext(e4BundleContext);
        osgiCtx.set("myKeyInOsgi", "value");
    }

    // Get the application E4 context 
    IEclipseContext appliCtx = workbench.getService(IEclipseContext.class);
    appliCtx.set("myKeyInApp", "value");

    // Get the main window E4 context 
    IEclipseContext windowCtx = workbench.getActiveWorkbenchWindow().getService(IEclipseContext.class);
    windowCtx.set("myKeyInWindow", "value");
}
```

**Getting data from your E3 code in E4 code**

Just inject it as any other E4 value.

**Putting E4 data in context and reuse it in E3 code**

In this use case, E4 code fills the context like usual.
The E3 code can:
- extract the value from the context (get it with previous code)
- be notified automatically only if the current code instance has been created using injection.

**Manage the injected selection in a E3/E4 compliant code**

In mixed mode, the selection can have different types:
- from the E3 code it is still an ISelection
- from the E4 code it is directly the selected type

Be aware to receive the both types in the E4 code.
A full example is provided in the : 'E4 plugin template with a view'.

**F. Extension Migration**

**Content**

- This part will give some advices to migrate the main `org.eclipse.ui` extensions
- To find how to migrate an element, you can launch your application using the model spy and check what the compatibility layer has generated in the model.
**View migration**

An `org.eclipse.ui.views` extension is actually a `PartDescriptor` in the application model.

To migrate a view:

- Copy your ViewPart code in the `xxx.e4.parts` package.
- Transform the code into a POJO:
  - remove inheritance to ViewPart
  - add `@PostConstruct` before the `createPartControl` method
  - add `@Focus` before the `setFocus` method
  - update the code to manage the selection using injection
  - remove the extension and the E3 code
- Bind this pojo in a model fragment:

To make the view appear in the 'Window -> Show view' menu:

- add this tags in the supplementary tab
**Perspective Migration**

- Perspectives must be defined in the application model (using a fragment or a processor)
- There is no ‘PerspectiveDescriptor’
- There is no PerspectiveStack defined by compatibility layer
  - You should add a PerspectiveStack in the children of the Window with ID: ‘IDEWindow’
- You can consider that perspective stack has always the same ID: `org.eclipse.ui.ide.perspectivestack`
- The perspective description can be got after running the compatibility layer
- To migrate the Perspective
  - get the perspective description and put it in a model fragment bound to perspective stack
  - remove the perspective extension
  - delete the perspective factory code.

**Command Migration**

An `org.eclipse.ui.command` extension can be defined in the ‘commands’ feature of the application model

- keep the same ID
- add the command in the fragment:

**Handler Migration**

To migrate an `org.eclipse.ui.handlers` extension:

- Copy the E3 handler code in the `xxx.e4.handlers` package
Transform the code into a POJO:
- remove inheritance to AbstractHandler
- add `@Execute` before the `execute` method
- add `@CanExecute` annotated method if needed
- receive needed values as parameters (will be injected)
- Bind this pojo in a model fragment (`xpath:/` and `handlers`)

**MenuContribution Migration**

An `org.eclipse.ui.menus` extension must be redefined in the model fragment
- use `'xpath:/'` and `'menuContributions'` feature
- The link is done using the parent ID

**Menu Contribution**

**MenuContribution / Parameters**

The following parent ID can be used:
- ID of an existing view (it must have been registered using the `EMenuService`)
- ID of an existing menu
- `org.eclipse.ui.main.menu` : used for the main menu
- `popup` : used to be located in any part
- `org.eclipse.ui.main.toolbar` : used to be located in the main toolbar.

For the position:
- an ID of any existing object (command, menu, etc...)
➢ **after=**additions : the default location

It is possible to open the model Spy so as to check the values used by the IDE

**Wizard migration**

- **org.eclipse.ui.(???)Wizards**
- Wizards are not defined in the application model
- There is also no extension point outside of org.eclipse.ui
- Therefore, the main dialog to choose a wizard is not available in a pure E4 application
- Nevertheless it is possible to open a specific wizard in a pure E4 code
- Wizards are only JFace code and can be adapted to deal with injected selection
- They must not implement **INewWizard**, **IImportWizard** or **IExportWizard** anymore
- A command must be created to open the wizard, using the **WizardDialog** of JFace

**Sample wizard**

```java
package com.opcoach.training.e4.codesamples;
import javax.inject.Inject;
import org.eclipse.e4.core.contexts.ContextInjectionFactory;
import org.eclipse.e4.core.contexts.EclipseContext;
import org.eclipse.ui.jface.wizard.Wizard;

public class SampleWizard extends Wizard {
  private SampleWizardPage firstPage = null;
  private EclipseContext context;

  @Inject
  public SampleWizard(EclipseContext ctx)
  {
    setTitle("New Wizard");
    context = ctx;
  }

  @Override
  public void addPages()
  {
    firstPage = ContextInjectionFactory.make(SampleWizardPage.class, context);
    addPage(firstPage);
  }

  @Override
  public boolean performFinish()
  {
    // Do your stuff here by asking the pages...
    return true;
  }
}
```

*Sample wizard*
Sample wizard page

```java
package com.opcoach.training.e4.codesamples;
import java.io.File;

public class SampleWizardPage extends WizardPage
{
    private Object selection;

    @Inject
    public SampleWizardPage(@Named(IServiceConstants.ACTIVE_SELECTION) Object currentSelection)
    {
        super("wizardPage");
        setTitle("Wizard Page title");
        setDescription("Wizard Page description");
        selection = currentSelection;
    }

    @Override
    public void createControl(Composite parent)
    {
        Composite container = new Composite(parent, SWT.MOUNTED);
        filename = new Label(container, SWT.BORDER);
        if (selection instanceof File)
            filename.setText(((File)selection).getName());
        container.setText(((File)selection).getName());
    }

    @Override
    public boolean isPageComplete()
    {
        return filename.getText().length() > 0;
    }
}
```

Inject current selection for content init

Sample wizard page

Opening wizard

```java
public class OpenSampleWizard {
    @Override
    public void execute(I EclipseContext ctx, Shell s)
    {
        Wizard m = ContextInjectionFactory.make(MyWizard.class, ctx);
        WizardDialog wd = new WizardDialog(s, m);
        wd.open();
    }
}
```

Open wizard

Preference pages Migration

- Like wizards, preference pages are not defined in the application model
- It is possible to use the plugin: https://github.com/opcoach/e4Preferences

7 - https://github.com/opcoach/e4Preferences
➢ You need to:
   ➢ ensure that your preference pages are extending `FieldEditorPreferencePage`
   ➢ change the extension `org.eclipse.ui.preferencePages` to `com.opcoach.e4.preferences.e4PreferencePages`
   ➢ add the handler and the command in your model

For the default values, you can keep the `org.eclipse.core.runtime.preferences` extensions.

**Other migrations**

➢ There are still plenty tips for your migration
➢ Try to put it in your model fragment
➢ If you can not describe your contribution in a model fragment, use a model processor

**G. Resources**

**Articles about migration**

➢ Eclipse magazin about migration (german):
  ➢ [https://jaxenter.de/ausgaben/eclipse-magazin-6-15](https://jaxenter.de/ausgaben/eclipse-magazin-6-15)  8
➢ Recipes for your Eclipse 4 migration (english)
  ➢ [9]
➢ OPCODE's article in eclipse magazin (german)
➢ Comment migrer vers eclipse 4 (french)

**Ask your questions**

Feel free to ask your questions
➢ in E4 forum
➢ by email:
  ➢ [olivier@opcoach.com](mailto:olivier@opcoach.com)
➢ Now, after this talk or during the conference!
➢ Visit my booth!

---

8 - [https://jaxenter.de/ausgaben/eclipse-magazin-6-15](https://jaxenter.de/ausgaben/eclipse-magazin-6-15)
Evaluate the session

Thank you to evaluate this talk