Eclipse 4

JUG Brussels

June 15 2015

Table des matières

I - Eclipse 4 *5*

A. Application Model *10*

B. E4 injection and annotations *12*

C. Event Broker *17*

D. CSS Styling *20*

E. Compatibility Layer *22*

F. Summary *23*

Eclipse 4

Introduction / OPCoach

Image 1

- Company founded in June 2009 *http://www.opcoach.com* [1]
- Member of the Eclipse Foundation (Solution Member)
- Eclipse committer on e4.tools and platform.ui
- 3 Eclipse-based activities:
  - Consulting
  - Training
  - Employment

1 - http://www.opcoach.com

Image 2

Web site

www.opcoach.com :

- mailing list
- next courses
- employment part to upload your application
- training testimonials

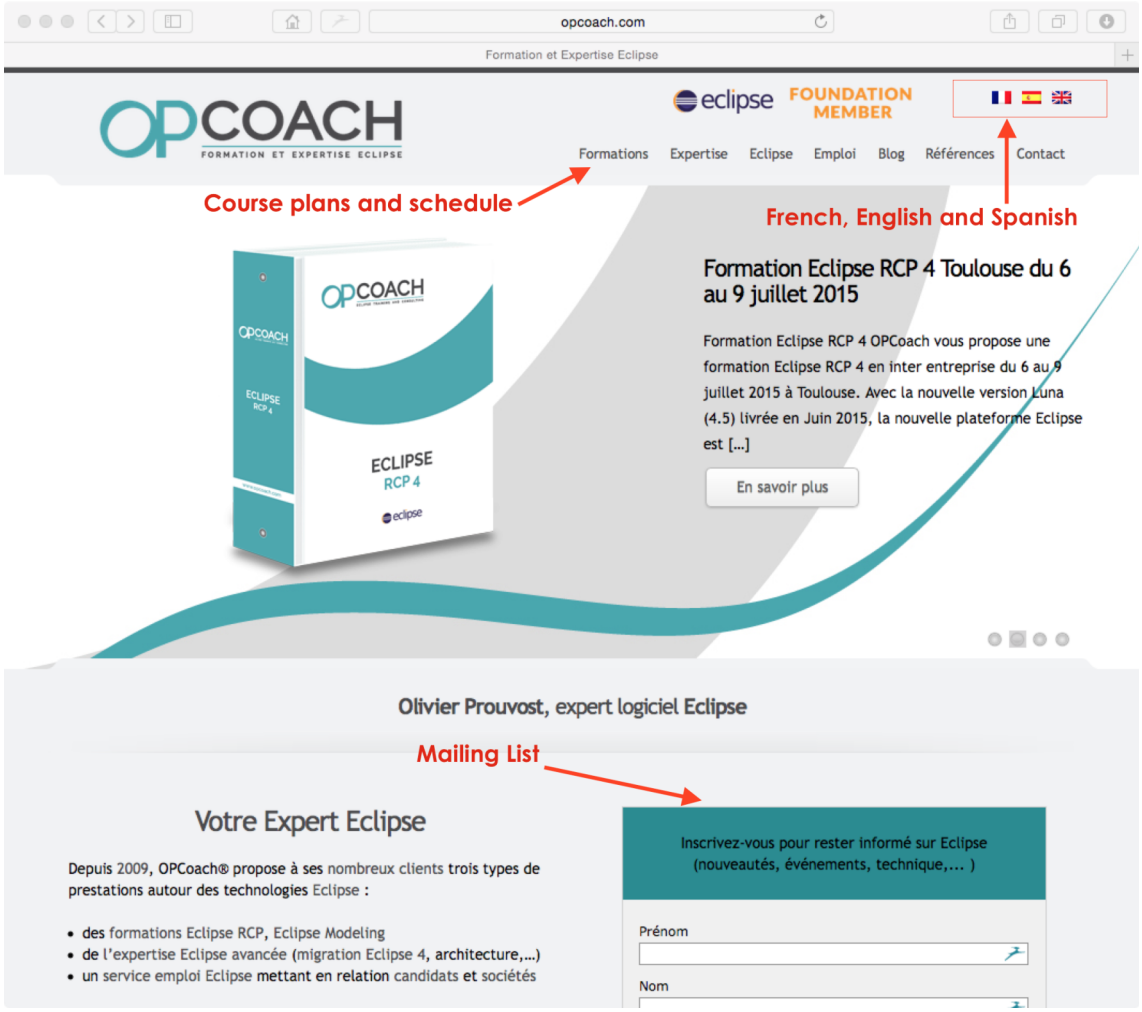


Image 3 site web

Twitter

@OPCoach_Eclipse

Agenda

This talk will focus on :

- 3.X history, benefits and disadvantages
- E4 architecture: the application model
- Injection and annotations
- Event broker
- CSS Styling
- E4 spies
- Compatibility layer
- Questions : keep 3.X ? use 4.X ?

Eclipse deliveries

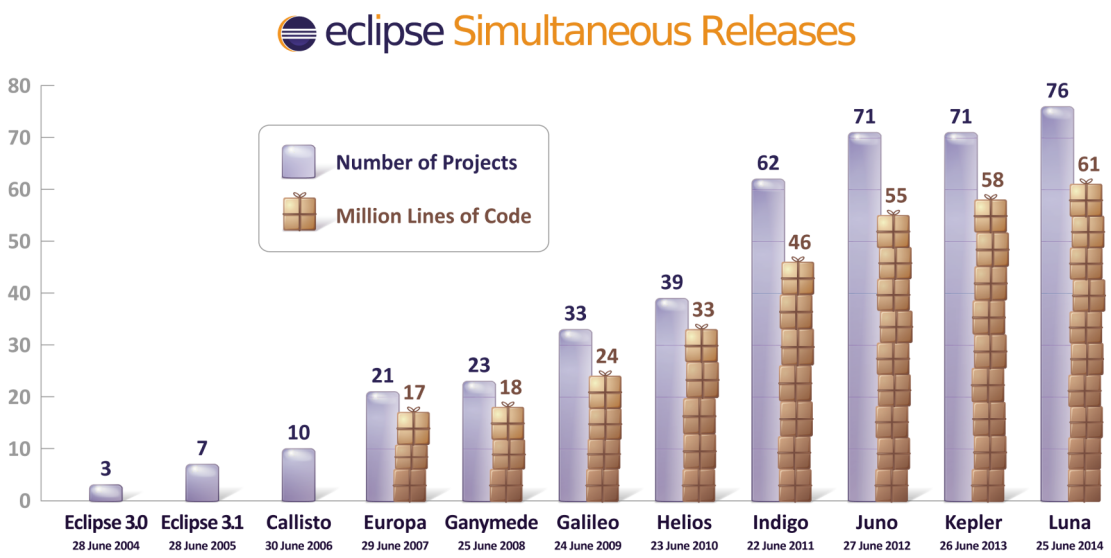


Image 4 Luna Release

Reasons for a new version

The benefits of Eclipse 3.X:

- Rich Client Platform (RCP)
- OSGi based architecture
- extension point concept
- portable on Windows, Mac, Linux, Web (with RAP)
- advanced Tooling (JDT, PDE, EMF, ...)
- adopted by many providers
- EPL : open source license

But ...

Eclipse 3.X drawbacks

- 'old' software (11 years)
- many extension points and extensions dispersed
- the developed application depends largely on the framework (inheritance, dependencies ...)
- not always consistent API
- too many singletons and not enough OSGi services
- injection mechanisms are not implemented
- no separation of content and appearance (no renderer)
- no CSS : difficult to parametrize the UI

E4 vs. E3

Strengths of E4

- Definition of an application model : centralizes all components of the application
- Unified UI agnostic components : view, editor, perspective, ...
- Injection support
- CSS Styling
- Compatibility layer

What is kept from E3:

- some extension points (mainly the org.eclipse.core.*)
- SWT / JFace and the business code
- architecture artifacts : bundle, plugins, fragments, features, repositories...
- build scripts with maven tycho

e4 Architecture

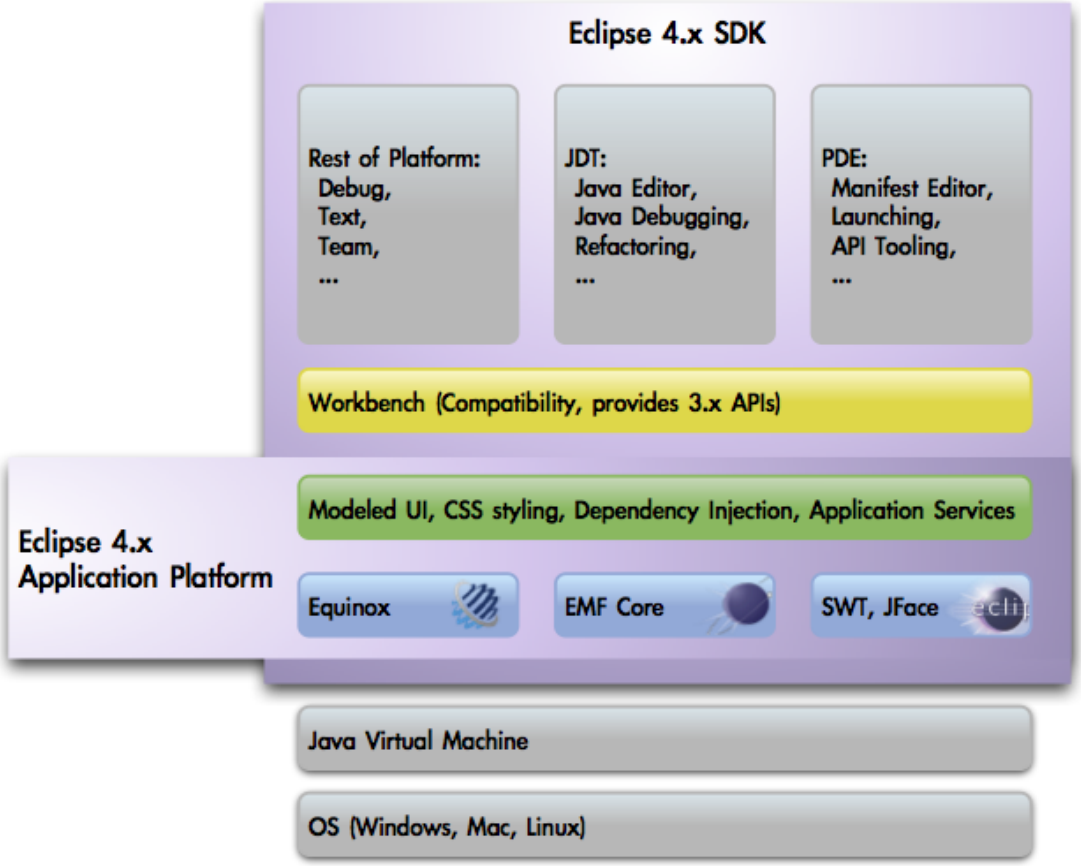


Image 5 e4 Architecture

What will change in the code:

- accessing the platform (using the injection)
- removing many inheritance relations (ViewPart, AbstractHandler, etc ...)
- moving some extensions to the model (views, handlers..)
- using services : ESelectionService, EMenuService
- replacing some useless listeners with EventBroker
- using the CSS management

A. Application Model

The application model

This is a global model that gathers the usual 3.X extension points :

- view, perspective, editor
- commands, handlers, menu

The application model

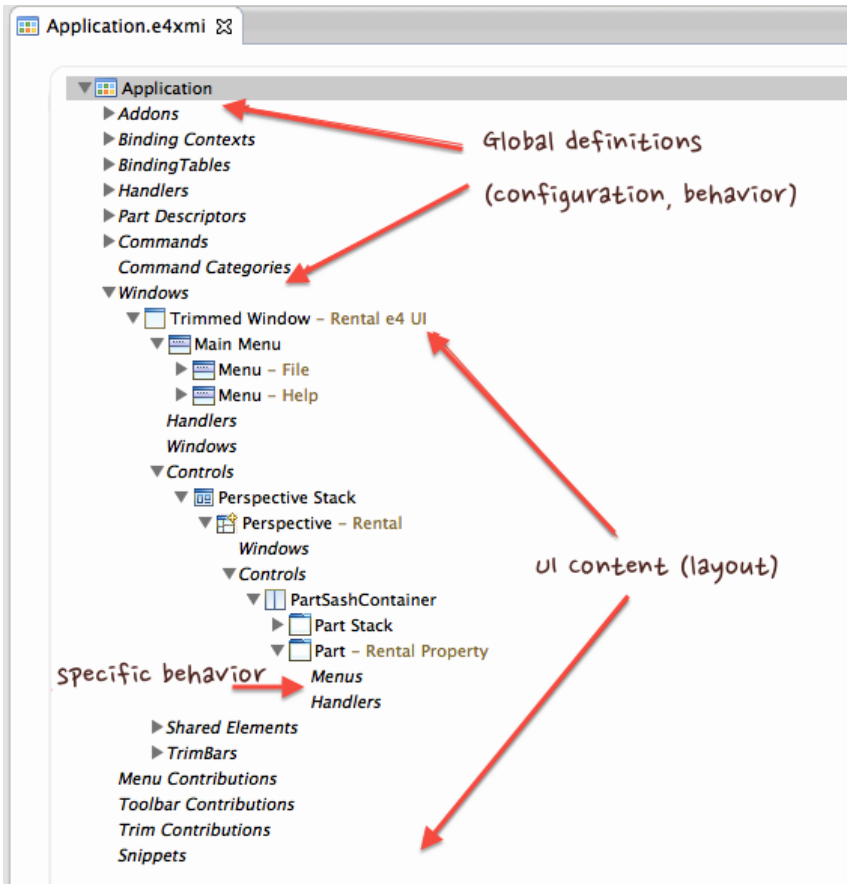


Image 6 Application Model

The contents of the UI or the behaviors of commands are defined by the code

Using the E4 model

In the case of a 3.X application running on 4.X runtime, the model is filled by the compatibility layer.

- this is the 'legacy' model (internal)

In the case of an 4.X application the model is associated to a product (using extension point):

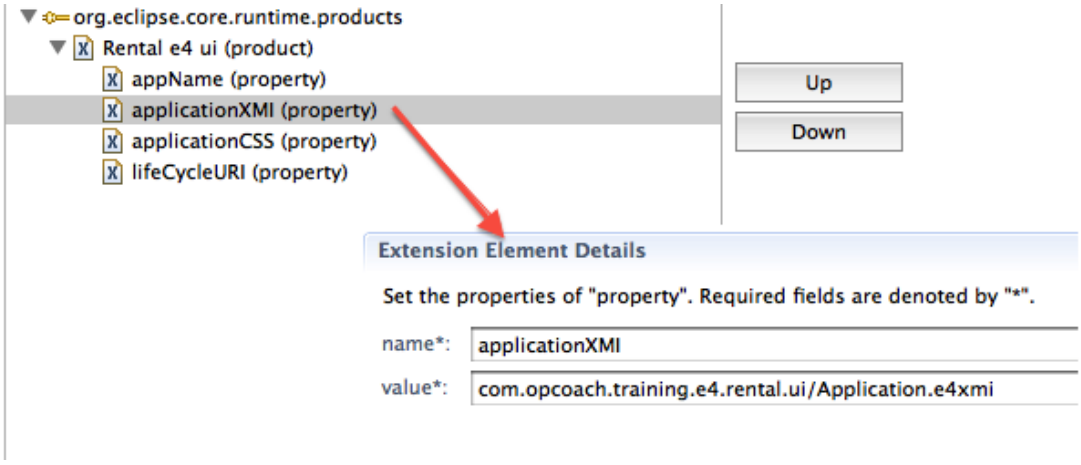


Image 7 product extension point

Model used at runtime

In all cases:

- The model is interpreted dynamically -> any changes will be released on the UI
- You can view the current model with the Alt Shift F9 shortcut (model spy).

Demo :

- Open the model spy to display the current model
  - check views, perspectives
  - find the menus, commands...
  - change their labels
- To do this, the spies must be installed before :
  - official update site : [2]
  - use this mirror (if not yet updated on eclipse (bug #470128)) : [3]
    - *http://dl.bintray.com/vogellacompany/e4tools-luna/* [4]

B. E4 injection and annotations

Introduction / Principle

The injection mechanism delegates to a framework the initialization of class fields or method parameters.

Using the **@Inject** annotation applied to a constructor, method or field.

Usage:

The UI renderer will instantiate the Parts by injecting the expected values.

2 - http://download.eclipse.org/e4/downloads/
3 - http://dl.bintray.com/vogellacompany/e4tools-luna/
4 - http://dl.bintray.com/vogellacompany/e4tools-luna/

```
15
16 public class SampleView
17 {
18     // This field is initialized by injector
19     @Inject
20     private ESelectionService selectionService;
21
22     // Constructor called with 2 injected parameters, second is optional
23     @Inject
24     public SampleView(Composite parent, @Optional IStylingEngine styleEngine)
25     {
26         // Code to create the view...
27     }
28
29 }
30
```

Image 8 Inject sample

The framework introspects classes to find the methods, fields and constructors annotated with **@Inject**.

The injection context

- The context stores the values associated with keys or classes
- The context is hierarchical
- Branches in context are activated by the framework (UI focus, part activation, ...)
- Branch activation gives an access to the available values in this branch
- If a value is not found, the context will ask its parent (**getParent**())
- The context is an implementation of **IEclipseContext**

Contexts for an application

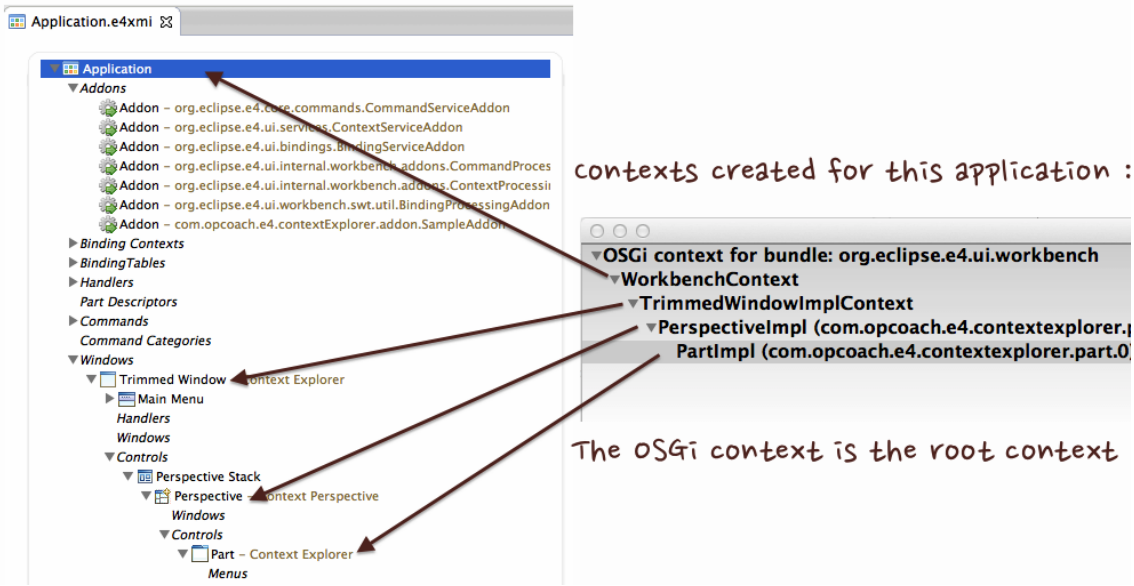


Image 9 Contexts in application

Example of use (in current branch)

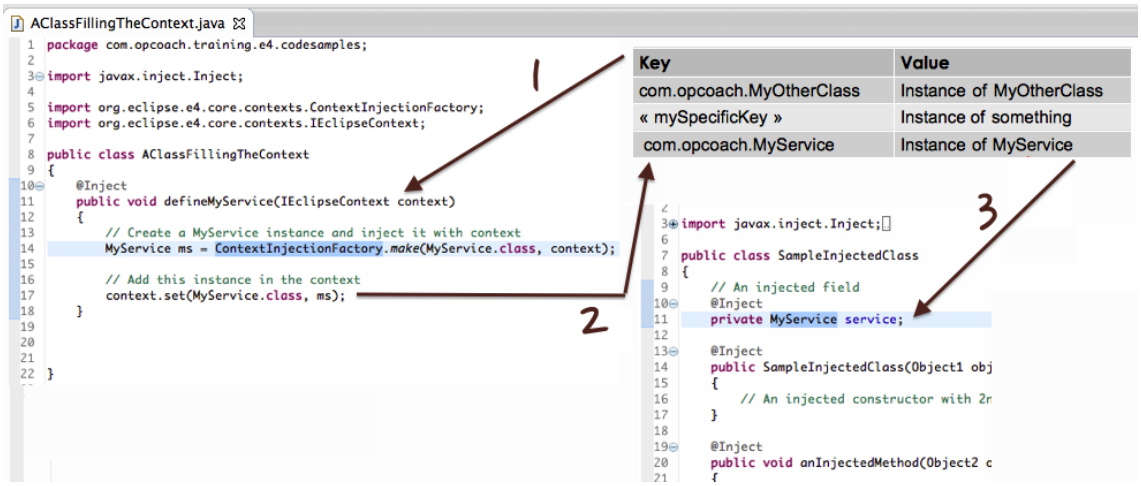


Image 10 Injection Sample

Additional annotations for @Inject

3 additional annotations are used to manage object creation:

- **@PostConstruct** : is used to annotate a method that must be called at the end of initialization
- **@Optional** : is used to indicate that a field or a parameter could be null
- **@Named** : is used to get an object by its name

Call order

The injection is processed in the following order:

- **@Inject** Constructor call
- **@Inject** Fields initialization
- **@Inject** Methods call
- **@PostConstruct** Method call

CallOrder.java

```
1  package com.opcoach.training.e4.codesamples.inject;
2
3  import javax.annotation.PostConstruct;
4  import javax.inject.Inject;
5  import javax.inject.Named;
6
7  import org.eclipse.e4.core.di.annotations.Optional;
8  import org.eclipse.e4.ui.workbench.modeling.ESelectionService;
9
10 public class CallOrder
11 {
12     public static final String MYPARAM = "myparam";
13
14     // --> Call #2 field is initialized after call to constructor
15     @Inject
16     private ESelectionService selectionService;
17
18     @Inject
19     // --> Call #1   Constructor is called first
20     public CallOrder()  { }
21
22     @Inject
23     // --> Call #3 (after field init)
24     public void aMethod()     { }
25
26     @Inject
27     // --> Call #4
28     public void anotherMethod()  { }
29
30     @Inject
31     // --> Call #5 and after when a value change
32     public void methodWithInjectedParam(@Optional @Named(MYPARAM) Object o)   { }
33
34     @PostConstruct
35     // --> Call #6 at the end
36     public void endOfInit() { }
37
38 }
```

2 1 3 4 5 6

fields are not yet initialized when constructor is running

Image 11 Call order

Fondamental : Basic rules of E4 injection

If a value previously set in the injector changes after an injection:

- it will automatically be re-injected into the relevant **@Inject** fields
- **@Inject** methods that have received this value as a parameter are automatically re-called.

If a value can not be injected the framework throws an exception (except if the **@Optional** is used)

Sample with @Named

```
116     @Inject
117     public void setSelection(@Optional @Named(IServiceConstants.ACTIVE_SELECTION) Object o,
118                              Adapter adapter)
119     {
120         Rental r = adapter.adapt(o, Rental.class);
121         setRental(r);
122
123     }
124
```

Image 12 @Inject @Named @Optional

Sample with @Preference

The **@Preference** retrieves the value of a preference.

It can be used for a class field or for a parameter method

```
141     private final static String PLUGIN_ID = RentalUIActivator.PLUGIN_ID;
142
143     @Inject
144     public void refreshTree(@Preference(nodePath=PLUGIN_ID, value=CUSTOMER_KEY) String custCol,
145             @Preference(nodePath=PLUGIN_ID, value=RENTAL_KEY) String rk,
146             @Preference(nodePath=PLUGIN_ID, value=RENTAL_OBJECT_KEY) String rok)
147     {
148         if (agencyViewer != null)
149         {
150             labelProvider.initPalette();
151             agencyViewer.refresh();
152         }
153     }
154
155
```

Image 13 @Preference

Final example code of 'Part'

```
26  public class ASampleView
27  {
28
29      @Inject
30      private ESelectionService selectionService;
31
32      @Inject
33      private EMenuService menuService; // This field is initialized by injector
34
35      @Inject
36      public void anotherMethod(@Optional EMenuService service)
37      {
38          // This method will be called
39      }
40
41      @PostConstruct
42      public void createContent(Composite parent, @Optional IStylingEngine styleEngine)
43      {
44          // Code to create the view
45      }
46
47      @PreDestroy
48      public void dispose()
49      {
50          // Call this method before deleting object (to remove listener for  instance)
51      }
52
53      @Focus
54      public void onFocus()
55      {
56          // This method is called when part takes focus
57      }
58  }
59
```

Image 14 A viewPart with annotations

Injection: advantages / disadvantages

Benefits :

- significant reduction in the coupling (the inheritance with the framework is no longer

necessary)
- the injector gathers all shared objects (hierarchical contexts)
- the dependencies are only on very high level interfaces

Disadvantages :
- when debugging an injected method, the caller code is introspection code
- requires knowledge of the objects that can be injected
- must master the lifecycle of the framework:
  - see: *http://www.vogella.com/articles/Eclipse4RCP/article.html* [5] : Chapter 16 Behavior
- there is less code control when editing:
  - classes are referenced by name in non-java editors (application model editor)
  - errors can be found only at runtime (no message 'never initialized object')

Injection Contexts, what can you inject ?

- it depends on where you are.
- Eclipse e4 creates the contexts when they are needed
- Check the available values using the context spy (Alt Shift F10)

Demo :
- Open the context spy to check the available values

C. Event Broker

Introduction

- Usually, to be notified of an event, a listener must be defined
- And for each case a specific method must be defined
- Example: if you want to listen to what is going on with rental, we would have
  - RentalListener with rentalCreated (RentalEvent), rentalModifyed (RentalEvent) ...
  - or you must use the EMF adapters.
- In E4 there is a more simple pattern: the **IEventBroker**

IEventBroker

The eventBroker is used to send an event identified with the relevant object

You can thereafter:
- receive event automatically when it appears (by injection)
- subscribe and be notified with an **EventHandler** (java.beans)

5 - http://www.vogella.com/articles/Eclipse4RCP/article.html

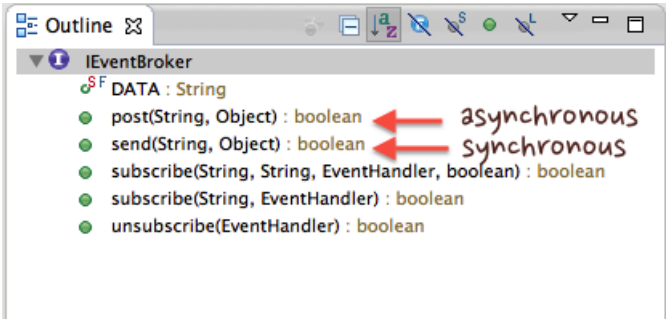


Image 15 Event Broker API

Sending an event

- Sending an event is simply a call to send or post
- The IEventBroker is received by injection

```
@Inject
IEventBroker broker;

@Execute
void execute()
{
    // Create a new rental using factory
    Rental r = RentalFactory.eInstance.createRental();

    // broadcast the rental creation in broker
    broker.send("rental/new", r);
}
```

publish an event using a specific ID

Image 16 Event Broker send

Receipt and processing an event

- We can do it in an injected method that receives a **@UIEventTopic** parameter
- The method being injected is called for each sent event
- String patterns can be used to receive events

```
public class RentalReact
{
    @Inject
    @Optional
    void reactOnRentalEvent(@UIEventTopic("rental/*") Rental newRental)
    {
        // Use the rental to update it in a view for instance
    }

}
```

receive here any event with ID starting with rental

Image 17 EventBroker receive

UIEvents

The UIEvents class declares the framework events :

- UILifeCyle (BRINGTOTOP, ACTIVATE...)

- application model updates (UIElement.VISIBLE...)

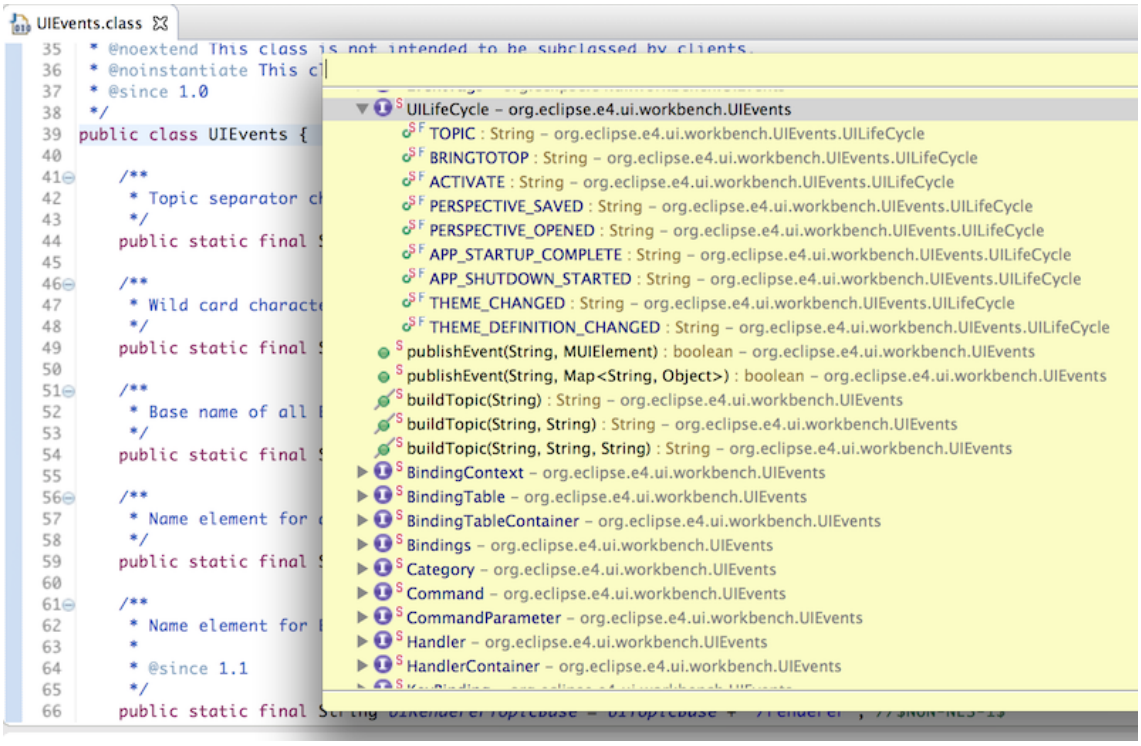
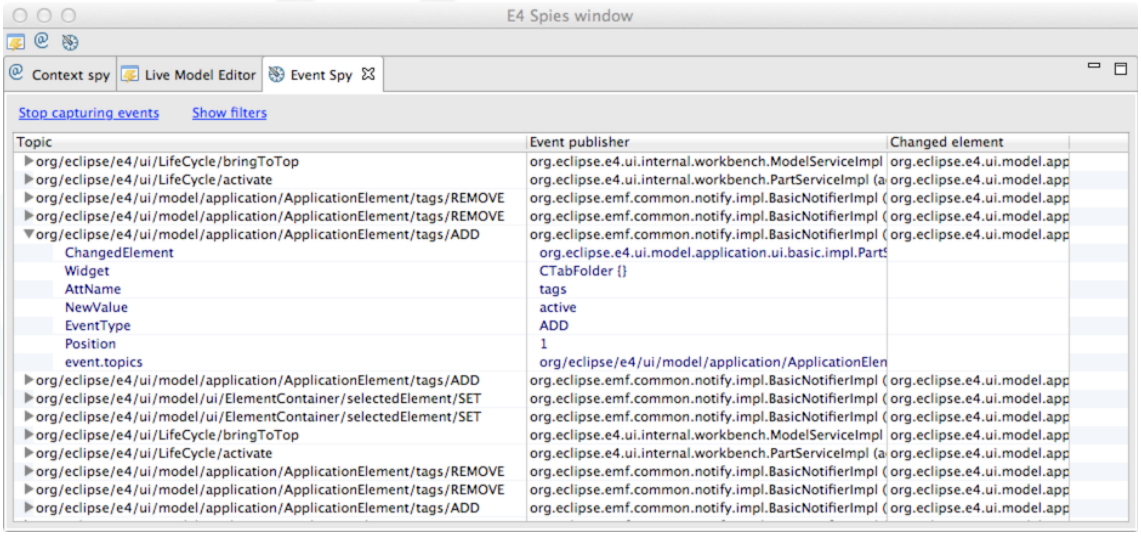


Image 18 UIEvents

Event Spy

- There is a dedicated spy to display events
- You can open it using the shortcut : **Alt Shift F8**
- Or use the event spy button in E4 spy window
- The **org.eclipse.e4.tools.event.spy** plugin must be launched



Event Spy

Demo : Event Spy

Open the event spy using the short cut : Alt Shift F8
Track the event when the windows are moved

D. CSS Styling

Introduction

The E4 runtime has its own CSS rendering engine
It can be used to style the application.

Content of the css

Writing a CSS can be done using the following CSS classes :

- name of the SWT widget : Label, Tree, ...
- name of a model element : MTrimmedWindow, ...
- name of an internal class created by code (using a **IStylingEngine**) :

default.css

```
1
2
3 .MTrimmedWindow.topLevel {
4     margin-top: 15px;
5     margin-bottom: 2px;
6     margin-left: 20px;
7     margin-right: 20px;
8     background-color: #FFF #AAA 100%
9 }
10
11 .MTrimBar {
12     background-color:  #CFCFCF #A8A8A8 100%;
13 }
14
15 .agencyViewer {
16     background-color: #FFFFFF #0000FF 100%
17 }
```

Use UI Model Java class

Use the CSS class defined in Java code

Image 19 Css

Result of CSS

Without CSS:

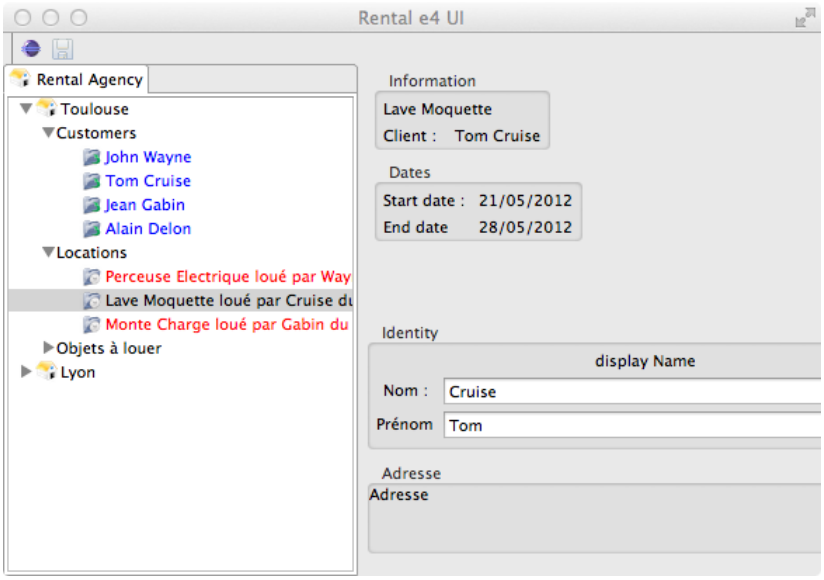


Image 20 without css

CSS:

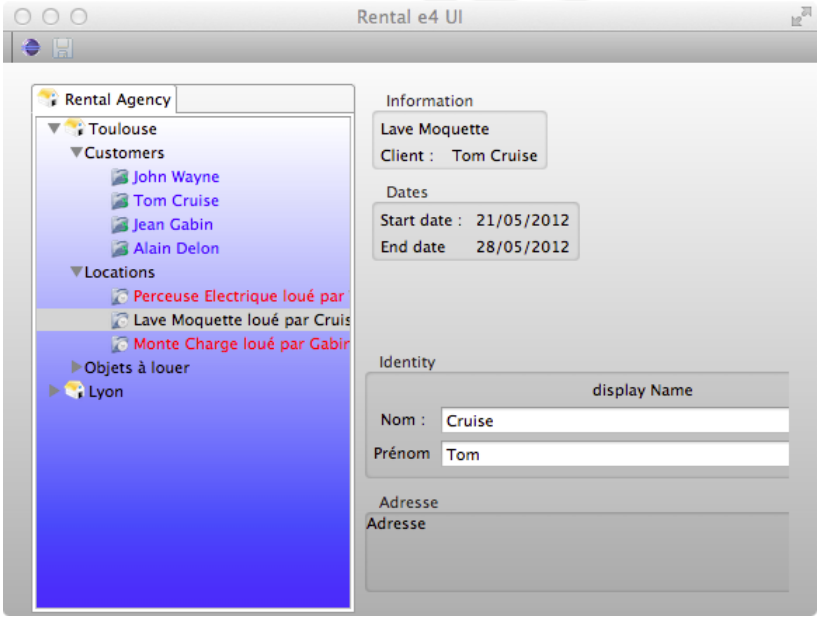


Image 21 with css

CSS Spy

- This spy can be used to find the code to write in the CSS
- It can point any widget in the UI and display the CSS sample

- **Demo** :
  - Open the CSS Spy using the Alt Shift F5 short cut
  - Use also the CSS Scratch pad to test the CSS values

E. Compatibility Layer

Introduction

E4 comes with a compatibility layer that allows:

- to launch an E3 application on the E4 platform
- to gradually migrate the application

Usage

Install Eclipse 4.5 (mars)

Install an Eclipse 3.X projects in the workspace

Create a specific launch configuration :

- select your application's main plugin
- add the following plugins (compatibility layer) :
  - **org.eclipse.equinox.ds**
  - **org.eclipse.equinox.event**
  - **org.eclipse.equinox.util**
  - **org.eclipse.platform**
- To view the application model line (Alt Shift F9) and spies, also add:
  - **org.eclipse.e4.tools.emf.liveeditor**
  - **org.eclipse.e4.tools.*.spy**
- add required plugins

From 3.X to 4.X (quick guide)

- Conditions to be able to start to migrate :
  - no internal APIs calls
  - no deprecated extension points
  - clean separation between ui and core
  - identified features to migrate feature after feature
- Create an application model : **application.e4xmi**
  - following the application model obtained with Alt Shift F9
  - or using the **LegacyIDE.e4xmi** (In org.eclipse.ui.workbench)
- Modify the code:
  - views and editors: removing inheritance on ViewPart, using injection
  - handlers: declaring commands and handlers in the model
  - advisors: moving the construction of menus in the model
  - selection: using ESelectionService and injection
  - change the access to Platform, PlatformUI, ...
  - extract the specific graphic in CSS
- Keep :
  - all the business code (EMF, non UI specific code) and test unit
  - all non UI extensions (adapters, expression definition ...)
  - architecture with plugins and fragments
  - internationalization

Evaluate the cost of migration

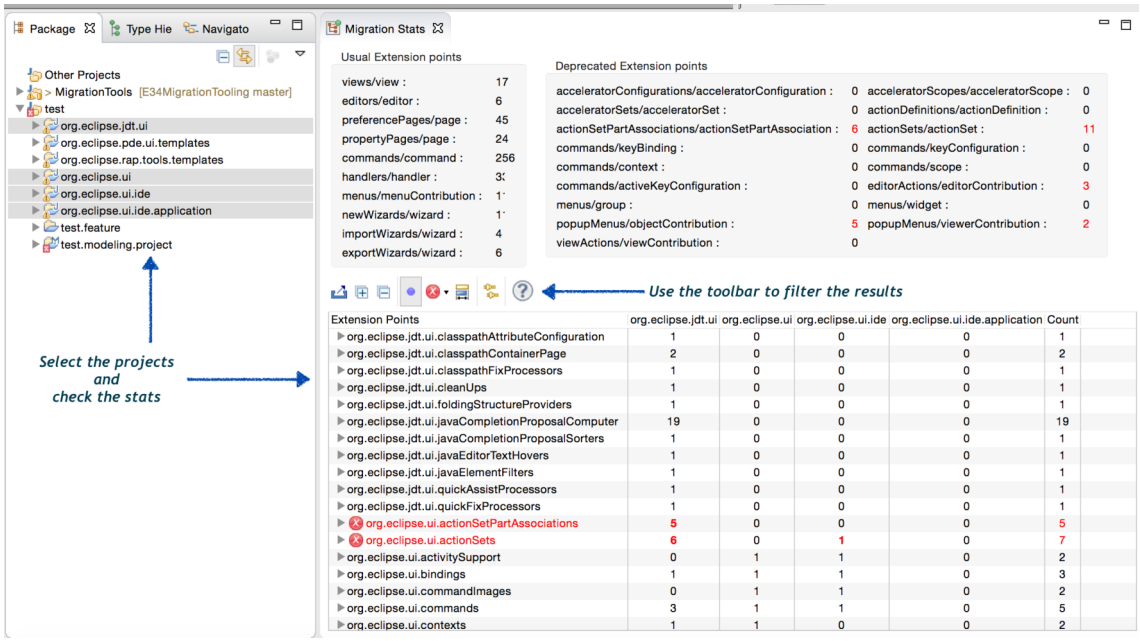
The cost of migration depends on several issues :

- number of ui extensions to migrate : views, editors, commands...
- number of plugins and features
- volume of specific 3.X code
- ...

OPCoach provides on github a specific plugin to help to evaluate the volume

- *http://opcoach.github.io/E34MigrationTooling/*[6]

It provides a specific view computing migration statistics relative to selected projects :



Migration tooling

F. Summary

What to do?

Your application is developed on 3.X

- it is almost finished -> keep the 3.X target platform or try the compatibility layer with 4.5
- it is under development, according to the progress, you can :
  - keep the 3.X target platform and try the compatibility layer
  - continue new developments using 4.X and 3.X code
  - migrate feature after feature the code up to a full 4.X application

The development of your application has not yet started:

- start with E4.X !
- if some issues are missing keep the compatibility layer for them, while it has not been provided (console, properties...).

6 - http://opcoach.github.io/E34MigrationTooling/

IN ALL CASES : Use Eclipse Mars (4.5) as an IDE

E4 status

- E4 framework is a great framework to make your applications
- The application model is really easy to use and the API is uniform
- Injection reduces the code and is very efficient
- The spies are helpful to understand what is happening
- Courses on these technologies are available in french, english and spanish.

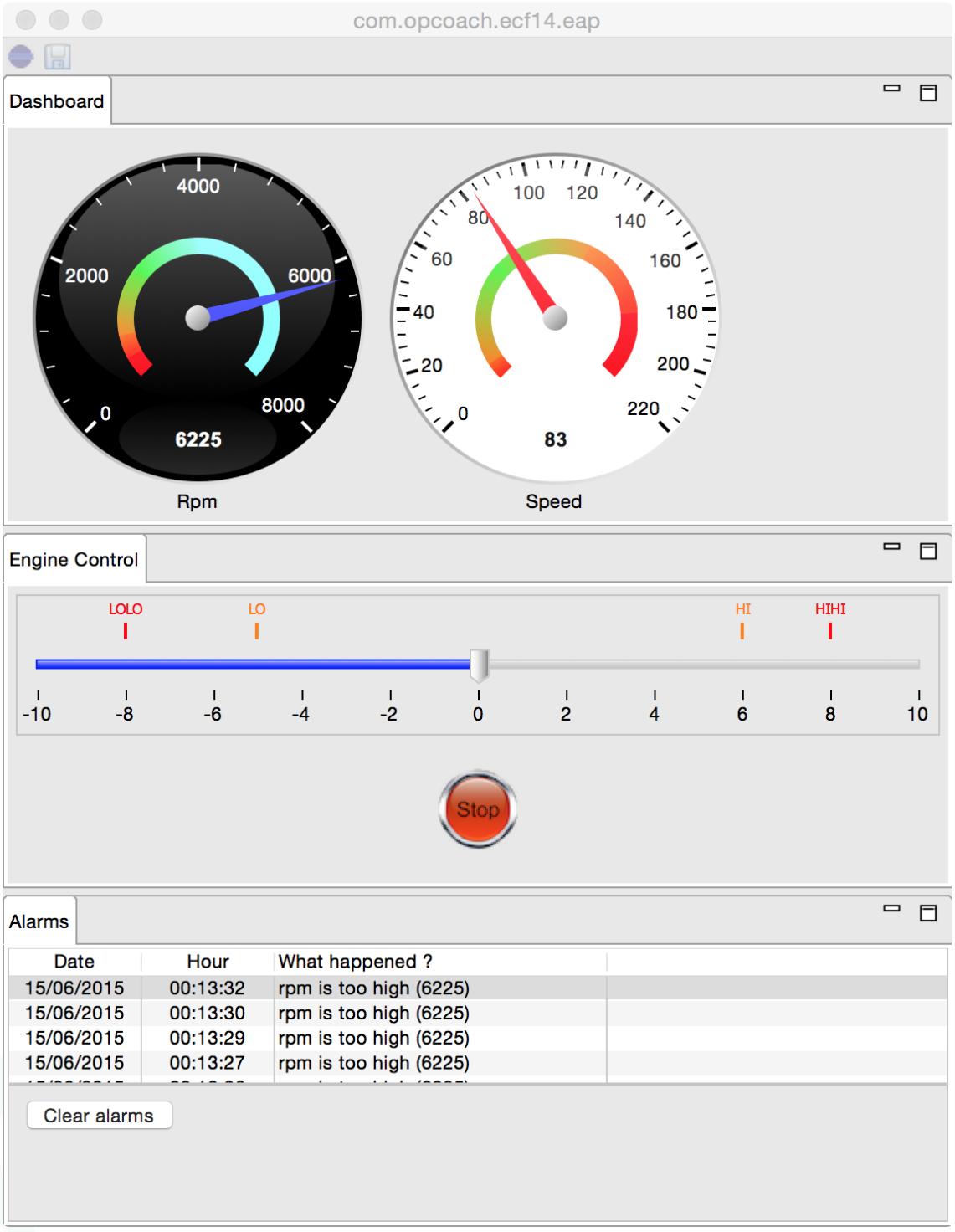
Eclipse courses

An E4 Application sample

During Eclipse Con France 2014, I provided an E4 application containing :

- an engine simulator (to provide sample data)
- an engine UI (to start/stop the engine and to accelerate/brake)
- a dashboard to display the speed and rpm value
- an alarm manager to trigger alarms depending on conditions

Basically the application is like this :



Main screen

Workshop Resources

- The goal of this workshop is to create the application from scratch
- All the code of this workshop is available on github :
- [7]
  - Import in workspaces only the 'ecf14_files' project
- The project is explained here :
  - *http://www.opcoach.com/eclipse4_workshop/*[8]
- And the pdf of the workshop is available here :
  - *https://www.eclipsecon.org/france2014/sites/default/files/slides/WorkshopE4_papier.pdf*[9]

Eclipse Con France 2015

- Located in Toulouse
- June 22, 23 -> Unconference
- June 24,25 -> Conference
- Register on : *www.eclipsecon.org*[10] (240 attendees in 2014)

7 - https://github.com/opcoach/Conferences/
8 - http://www.opcoach.com/eclipse4_workshop/
9 - https://www.eclipsecon.org/france2014/sites/default/files/slides/WorkshopE4_papier.pdf
10 - http://www.eclipsecon.org

- OPCoach is Bronze Sponsor with a booth :

If you can't go to Toulouse, I have some goodies for you.

Any Questions ?