Migration Eclipse 3 -> 4

Demo Camp Eclipse

Toulouse, 28 novembre 2013

OPC 13 DEM PRE EC4 01 A

E3 to E4 migration.

Agenda

- Différences entre Eclipse 3 et 4
- Concepts d'Eclipse 4
- Pourquoi et quoi migrer ?
- Quand migrer ?
- Qui peut faire une migration ?
- Comment s'organiser ?

OPCoach

- **Olivier Prouvost**, expert XML, Java, Eclipse
- SARL créée en juin 2009 *http://www.opcoach.com*
- Membre de la fondation Eclipse (Solution Member)
- OPCoach propose sur Eclipse :
  - **Formation sur mesure** : RCP, E4, Modeling, ...
  - **Expertise** : réponse appel d'offre, architecture, audit, expertise à l'heure sur skype
- Twitter : @OPCoach_Eclipse, mailing liste sur la page d'accueil du site.
- Références


AIRFRANCE
cea
intel
DGA
Alcatel·Lucent
EADS
JANSSEN PHARMACEUTICA
WINCOR NIXDORF
freescale semiconductor
THALES
ALL4TEC
Technologies
ALYOTECH
AKKA TECHNOLOGIES
ASTRIUM AN EADS COMPANY
Atos
AUSY
bsi meet your clients
CASSIDIAN AN EADS COMPANY
Dynasty
ESTEREL Technologies
ifp Energies nouvelles
cnes
Ifremer
LAAS-CNRS
Laboratoire Dr COLLARD www.labocollard.be
obeo Model Driven Company
EMC²
APSIDE
magellium
MINISTÈRE DE LA DÉFENSE
OPTILOGISTIC
SOGETI
assystem
SILICOM
Umanis
TECH'advantage
Atos Origin WORLDWIDE IT PARTNER
zenika ARCHITECTURE INFORMATIQUE
Polish Air Navigation

Image 1

Les raisons du changement

Les avantages d'Eclipse 3.X / RCP :

- architecture basée sur OSGi
- outillage avancé (JDT, PDE, EMF, ...)
- définition de point extensions
- open source and EPL license

Les raisons du changement

Les inconvénients d'Eclipse 3.X :

- une souche logicielle de 8 ans d'age et plus
- beaucoup de points d'extensions et d'extensions dispersés
- une forte liaison au framework (héritage, dépendances...)
- API peu uniformes et beaucoup de singletons
- Les mécanismes d'injection ne sont pas implémentés
- Pas de séparation du contenant et de l'apparence (pas de renderer)
- Difficulté de 'styler' l'application
- Pas d'utilisation des services OSGi

Concepts d'Eclipse 4

- Application model
- Injection
- Couche de compatibilité

Le modèle d'application E4

- C'est un modèle global qui rassemble les points d'extensions habituels :
  - vue, perspective, menus (visuels)
  - commande, handlers, key bindings (non visuels)
- Il décrit simplement la structure de l'IHM sans détailler son contenu
- Sa structure est définie par un méta modèle décrit en Ecore
- Il peut s'éditer avec un éditeur dédié
- Il peut être modifié et l'ihm est remise à jour
- Ce modèle est indépendant de l'affichage
  - un 'renderer' spécifique permet de l'afficher (swt et javafx)
- Les classes référencées dans le modèle d'application sont alors des POJOS annotés

Application model

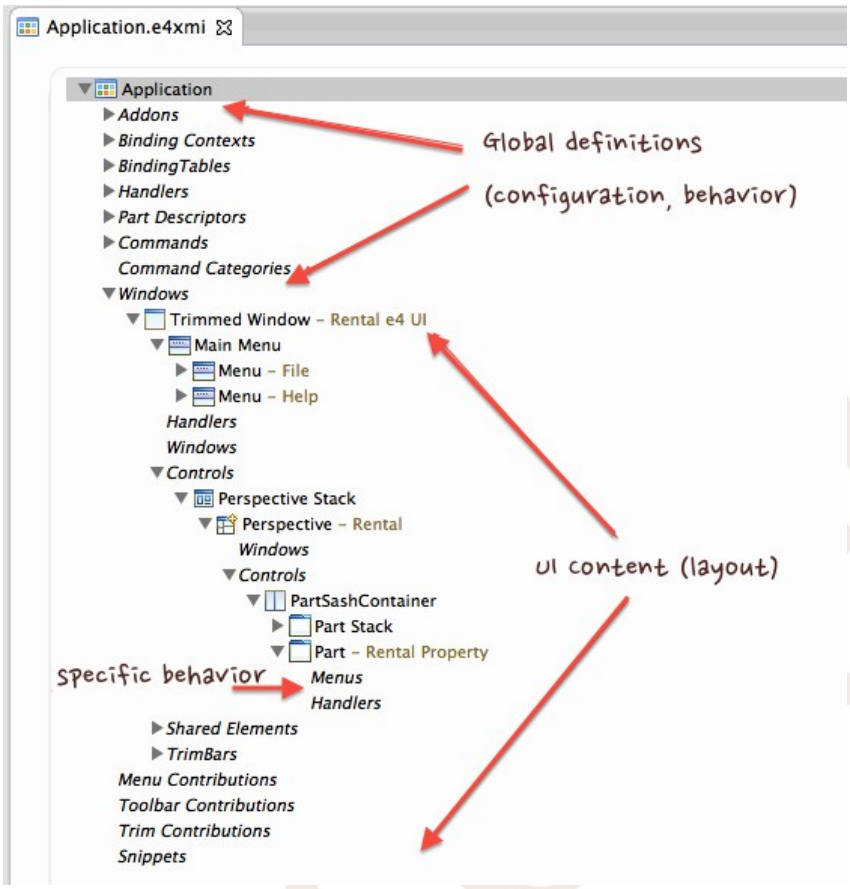

Image 2 Application Model

Introduction / Principe

- L'injection consiste à déléguer à un framework l'initialisation des valeurs des champs ou des paramètres
- L'injection fonctionne en utilisant un contexte qui contient les instances
- On utilise l'annotation **@Inject** (javax.inject) pour injecter les valeurs
- Elle peut s'appliquer sur un constructeur, une méthode ou un champ
- L'injecteur introspecte les classes pour gérer les artefacts qui ont une annotation **@Inject**

SampleInjectedClass.java

```java
package com.opcoach.training.e4.codesamples;

import javax.inject.Inject;

import org.eclipse.e4.core.di.annotations.Optional;

public class SampleInjectedClass
{
    // An injected field
    @Inject
    private MyService service;

    @Inject
    public SampleInjectedClass(Object1 object, @Optional Object2 o2)
    {
        // An injected constructor with 2nd parameter optional
    }

    @Inject
    public void anInjectedMethod(Object2 o2)
    {
        // An injected method
    }

}

```

Image 3 Sample injected class

Règle de base de l'injecteur E4

Si un objet est modifié dans le contexte, il sera automatiquement réinjecté :

- dans le champ concerné
- dans la méthode le recevant en paramètre **et la méthode sera réinvoquée**

Ceci permet de simplifier le code et de supprimer de nombreux listeners !

```java
    @Inject
    public void setSelection(@Optional @Named(IServiceConstants.ACTIVE_SELECTION) Object o,
                             Adapter adapter)
    {
        Rental r = adapter.adapt(o, Rental.class);
        setRental(r);

    }

```

Image 4 @Inject @Named @Optional

Contextes d'une application

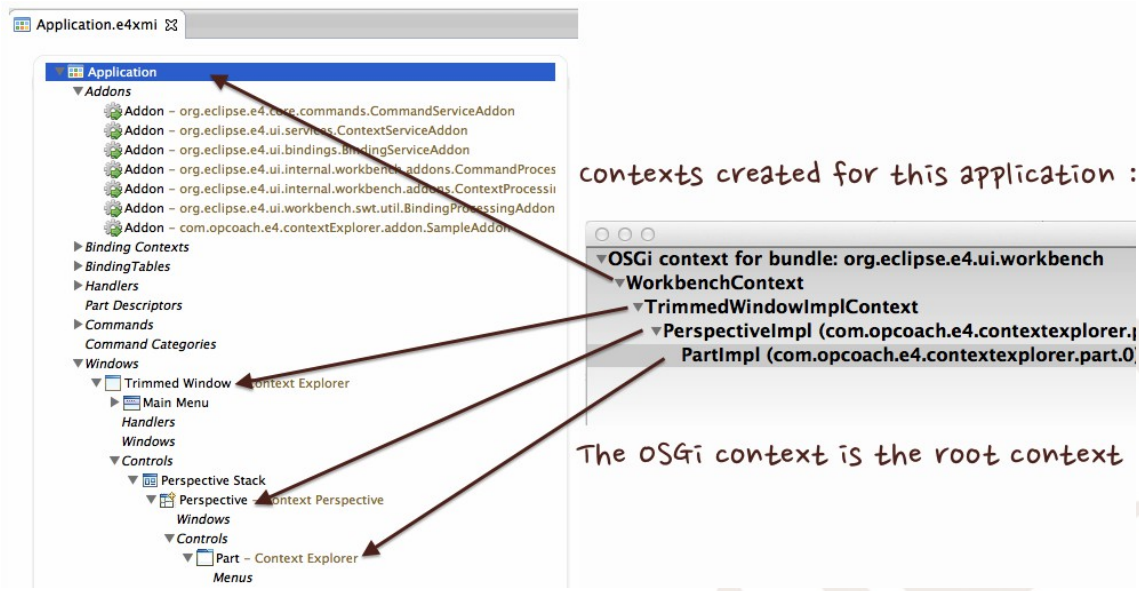

Image 5 Contexts in application

Context explorer

- Le context explorer permet de consulter le contenu des contextes
- Il est disponible sur github : **https://github.com/opcoach/contextExplorer**
- Il sera très prochainement intégré dans les E4 tools (Cf bug #442543)

Couche de compatibilité

La couche de compatibilité utilise le moteur E4 pour exécuter une application E3
Elle fournit un modèle d'application par défaut : le LegacyIDE.e4.xmi
Elle transforme les concepts E3 en concepts E4 :

- une extension de vue devient un MPart dans le modèle
- une extension de perspective devient une MPerspective
- etc...

Eclipse Juno 4.2, Kepler 4.3 sont lancés avec la couche de compatibilité

Lancer une application E3 sur E4

L'application ne doit avoir aucun code utilisant des 'internal'
Il faut simplement créer une nouvelle launch configuration
Et s'assurer que les plugins suivants soient présents :

- **org.eclipse.equinox.ds**
- **org.eclipse.equinox.event**
- **org.eclipse.equinox.util**
- **org.eclipse.e4.ui.workbench.addons.swt**
- **org.eclipse.e4.tools.emf.liveeditor** (éventuellement -> Alt Shift F9)

C'est une première étape pour commencer un portage.

Les raison d'une migration

- Depuis Juin 2012, Eclipse 4 est l'architecture officielle pour les nouveaux développements !
- Le modèle d'application est dynamique et UI agnostique (SWT, Java FX)
- L'injection est super cool et elle réduit le volume de code
- Le système d'événements d'E4 (**IEventBroker**) est très concis et adapté à l'injection
- Vous pourrez utiliser les CSS d'E4
- Votre application va continuer à vivre et évoluer plusieurs années.
- Les nouveautés dans Eclipse seront développées sur cette architecture.

Les conditions d'une migration

Votre application doit être **bien architecturée et propre** !

- Il ne doit y avoir aucune utilisation de package 'internal'
- les UI et core plugins doivent être clairement séparés
- Les features doivent être clairement définies
- Les couplages avec E3 doivent être identifiés
- Les packages dans ui plugins doivent être bien nommés et identifiés : views, handlers...

Quels types de migrations peut on effectuer ?

Migration totale

Objectifs :

- tous les plugins sont portés sur E4
- le modèle d'application est géré par votre application
- la couche de compatibilité n'est plus utilisée

Conditions :

- votre application est bien architecturée : séparation core/ui
- les couplages avec E3 sont bien identifiés (extensions utilisées, code ...)
- il n'y a pas de vues standard d'Eclipse E3 : console, properties...

Exemples d'applications candidates :

- application de simulation
- application frontal de BDD
- ...

Migration partielle

Objectifs :

- seulement certaines features de votre application sont migrées
- chaque feature propose un fragment de modèle pour contribuer au modèle legacy
- la couche de compatibilité reste présente

Conditions :

- l'application est architecturée en features bien isolées qui peuvent se migrer facilement
- vous envisagez de faire évoluer l'application

Exemples d'applications candidates :

- Applications bien architecturées continuant à évoluer

Pas de migration

Ne pas migrer est aussi une solution !

Il peut y avoir plusieurs raisons pour ne pas migrer :

- votre application est ancienne et n'évoluera plus trop
- l'application n'est pas bien architecturée (il vaut mieux tout refaire !)
- l'application utilise trop de code E3 non encore migrable (console, ...)
- vous n'avez pas de budget et la plate forme E3 vous suffit.

Que faut il migrer ?

Que faut il migrer ?

Ce qu'il faut migrer :

- Tous les plugins UI
- Les plugins sans UI si on veut utiliser l'injection
- certains fragments de test
- les features (qui référencent les nouveaux plugins UI)
- Le fichiers pom parent (pour la target platform)

Ce qu'il ne faut pas migrer :

- les plugins sans UI qui n'utilisent pas l'injection
- les fragments de plugin contenant du code natif ou de l'i18n
- certains fragments de tests
- les fichiers pom de chaque artifact

Qu'est ce qu'E4 offre et n'offre pas par rapport à E3 ?

Eclipse 3 n'est pas entièrement porté mais JFace fonctionne sans portage...

Ce qu'on peut utiliser dans E4

- tout le code SWT et JFace de base : treeviewer, table viewer, composites...
- les wizards et les pages de propriété, mais sans le point d'extension
- les pages de préference (cf hack : github opcoach training e4)
- la partie non IHM de l'update manager

Ce qu'on ne peut pas utiliser facilement dans E4 :

- les vues standard : console, log, error ! disponibles dans Luna
- la partie IHM de l'update manager
- l'aide en ligne
- le common navigator framework

Quand commencer la migration ?

Quand migrer ?

Si on décide de migrer, on peut commencer dès maintenant :

- En préparant son application pour la migration
  - rearchitecture si nécessaire
  - suppression des 'internal'

- identifier les parties à migrer
- En se formant à cette nouvelle architecture
  - manipuler l'application model et les fragments
  - bien maitriser l'injection
  - en commençant à migrer les features les plus simples

Si on décide de migrer, comment procéder ?

Exemple de plan d'action

- Faire un état des lieux pour voir l'architecture : features, identifier les couplages e3...
- Utiliser Kepler (voire Luna Mx) et les dernières versions d'outils E4
- Lancer l'application E3 avec la couche de compatibilité
- Pour chaque feature à migrer
  - recopier les plugins à migrer et faire la migration par partie : vue, handler, wizard...
  - créer un fragment de modèle ou un processor qui complète le legacy
- Une fois toutes les features migrées, créer le plugin eap avec l'application model
  - intégrer les fragments de modèle dans le modèle global si nécessaire
- Utiliser git pour gérer votre code et pour créer vos branches !

Qui doit se charger de la migration ?

Les acteurs de la migration

Un équipe qui :

- connaît les concepts E4
- a une vue globale de l'application à migrer
- a des compétences d'architecture logicielle
- est curieuse et motivée ! :)

A. Trucs et astuces

Quelques astuces pour votre migration

Vous cherchez une des nombreuses dépendances à utiliser pour votre migration :

- Cf : *http://www.opcoach.com/2013/04/les-dependances-pour-utiliser-eclipse-4/*

Votre application semble mal se comporter alors qu'elle marchait bien avant :

- effacer complétement le runtime workspace

Le **@PostConstruct** n'est pas appelé :

- **Vérifier si java.annotation est utilisé en required-bundle au lieu de import-package**
- Cf :
  - FAQ Eclipse 4 (§4.1) : *http://wiki.eclipse.org/Eclipse4/RCP/FAQ*[1]
  - Bug 348123 : *https://bugs.eclipse.org/bugs/show_bug.cgi?id=348123*

1 - http://wiki.eclipse.org/Eclipse4/RCP/FAQ#Why_isn.27t_my_.40Inject-able.2F.40PostConstruct_methods_being_injected.3F

D'autres liens pour vous aider

Interview de Paul Webster sur les évolutions E4 :

- *http://www.infoq.com/interviews/paul-webster-eclipse-platform-UI*

Les raisons de migrer selon Wim Jongman

- *http://industrial-tsi-wim.blogspot.fr/2012/10/why-eclipse-e4-egg-laying-woolmilkpig.html*

Quelques tutorials de migration :

- from lars : *http://www.vogella.com/articles/Eclipse4MigrationGuide/article.html*
- from eclipse source : *http://developer.eclipsesource.com/tutorials/#eclipse4*

Le forum Eclipse 4 :

- *http://www.eclipse.org/forums/index.php/f/12*

Mon email :

- olivier.prouvost@opcoach.com

Pour récupérer cette présentation

Inscrivez vous sur la mailing list -> *http://www.opcoach.com*

Questions ?