

# Introduction

<b>INTRODUCTION .....</b>	<b>1</b>
TABLE DES MATIERES .....	2
QUELQUES GENERALITES .....	4
<i>les objectifs initiaux</i> .....	4
<i>un framework général d'intégration d'outils généraux</i> .....	4
<i>une architecture, des principes de conception et des librairies</i> .....	4
LES CONCEPTS (1) .....	5
<i>les abstractions de l'UI en image (1)</i> .....	5
LES CONCEPTS (2) .....	6
<i>le modèle d'UI (1)</i> .....	6
LES CONCEPTS (3) .....	7
<i>le modèle d'UI (2)</i> .....	7
LES CONCEPTS (4) .....	8
<i>le modèle de ressources</i> .....	8
<i>les abstractions du modèle</i> .....	8
L'ARCHITECTURE (1).....	9
<i>une vue générale</i> .....	9
<i>le point d'extension</i> .....	9
L'ARCHITECTURE (2).....	10
<i>quelques points d'extension</i> .....	10
L'ARCHITECTURE (3).....	11
<i>le contrôle de l'occupation mémoire et des performances : la stratégie de chargement (paresseux)</i> .....	11
LE DEVELOPPEMENT D'UN PLUGIN (1).....	12
<i>La démarche</i> .....	12
<i>l'exemple incontournable depuis K&amp;R !!</i> .....	12
LE DEVELOPPEMENT D'UN PLUGIN (2).....	13
<i>la marche à suivre</i> .....	13
LE DEVELOPPEMENT D'UN PLUGIN (3) .....	14
<i>la marche à suivre (suite)</i> .....	14
LE DEVELOPPEMENT D'UN PLUGIN (4) .....	15
<i>la marche à suivre (suite)</i> .....	15
LE DEVELOPPEMENT D'UN PLUGIN (5).....	16
<i>le descripteur plugin.xml généré</i> .....	16
LE DEVELOPPEMENT D'UN PLUGIN (6).....	17
<i>le lancement de l'application</i> .....	17

## développement de plugins Eclipse

LE DEVELOPPEMENT D'UN PLUGIN (7) .....	18
<i>l'affichage du message à partir du bouton ou du menu</i> .....	18
LE DEVELOPPEMENT D'UN PLUGIN (8) .....	19
<i>les dépendances</i> .....	19
<i>les versions des plug-ins</i> .....	19
LE DEVELOPPEMENT D'UN PLUGIN (9) .....	20
<i>les extensions</i> .....	20
L'ARBORESCENCE (1) .....	21
<i>l'organisation générale des informations dans Eclipse 3.x</i> .....	21
L'ARBORESCENCE (2) .....	22
<i>le répertoire plugins</i> .....	22
L'ARBORESCENCE (3) .....	23
<i>le répertoire features</i> .....	23
L'ARCHITECTURE (4) .....	24
<i>les abstractions de contrôle</i> .....	24
<i>quelques observateurs intégrés</i> .....	24
L'ARCHITECTURE (4) .....	25
<i>le contrôle du footprint et des performances : les conditions d'activation</i> .....	25
<i>le contrôle de : l'ordonnancement la réactivité de l'UI</i> .....	25
LES INTERFACES IPLUGINDESCRIPTOR ET IPLUGIN .....	26
<i>le rôle</i> .....	26
<i>le chargement</i> .....	26
<i>l'accès aux ressources statiques</i> .....	26
DES ELEMENTS DE BIBLIOGRAPHIE .....	27

## Quelques généralités

### les objectifs initiaux

- offrir un environnement d'intégration d'outils
  - pas nécessairement limité aux IDE
  - pas nécessairement limité à des ressources java

offrir une plateforme extensible de développement

Evolue vers la construction de clients « riches »

Eclipse : environnement écrit en java mais pouvant intégrer des outils travaillant sur des ressources non java (C++, C, ...)

### un framework général d'intégration d'outils généraux

- un modèle d'UI général : **Fenêtre (Window), Perspective, Page, ...**
  - modèle d'organisation et d'interaction : **Editeur (Editor), Vue (View), Menu global, ...**
  - modèle de gestion des **Preferences, ...**
  - modèle de documentation : aide en ligne, **infopop**
- un modèle de ressources général : **Ressource, Espace de travail (Workspace), Projet (Project), .....**
- un modèle de travail coopératif et de développement (modèle de debug, de lancement d'applications)
- un modèle d'installation et de mise à jour : **Plugin, Feature**

### une architecture, des principes de conception et des librairies

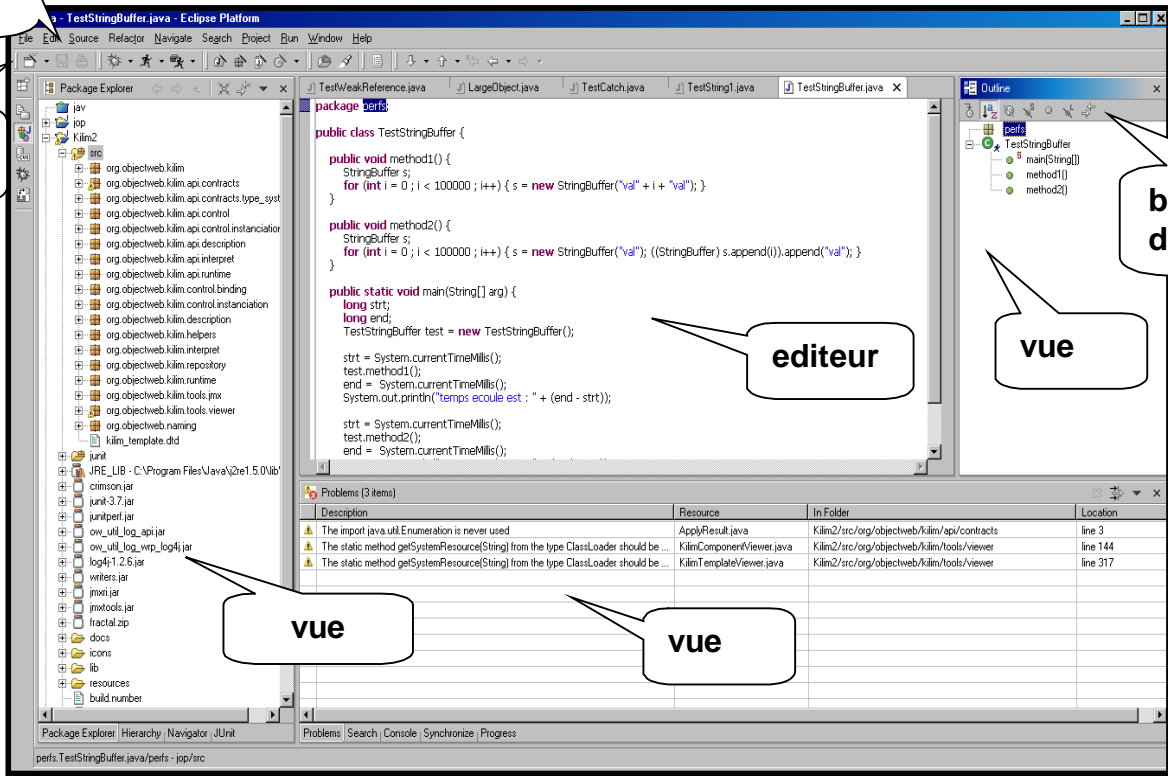
- une architecture très modulaire et incrémentale à base de plug-ins
- un objectif technique : minimiser le l'occupation mémoire et optimiser les performances
  - mécanisme de chargement souple et incrémental : **Plugin, Fragment**
- l'UI repose sur les librairies graphiques : **SWT, JFace**

aujourd'hui  
plugin = bundle osgi

## les abstractions de l'UI en image (1)

barre de menu  
de fenêtre

barre d'outils  
de fenêtre



## Les concepts (2)

### le modèle d'UI (1)

- **Atelier (Workbench)** : l'abstraction de l'environnement (sans manifestation physique)

- correspond à un ensemble de **Fenêtres**

à tout instant une seule fenêtre active (**ActiveWindow**)

- **Fenêtre** : la manifestation physique du Workbench

- indépendance mutuelle des Fenêtres
- correspond à un ensemble de **Pages**

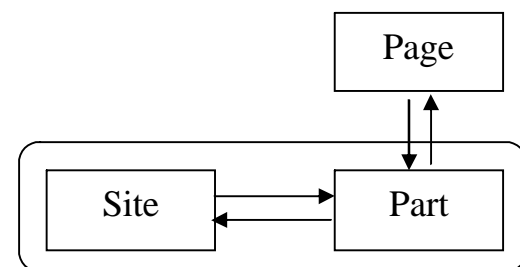
contient une barre de titre (en haut)  
 contient une barre de menus (sous la barre de titre)  
 une barre d'outils (sous la barre de menus)  
 une barre de perspective (en haut à droite)  
 une ligne de statut (en bas)  
 un indicateur de progression (en bas)

- **Page** : un ensemble de **Parts** (et **Sites**)

distinction entre Part et Site → faciliter l'évolution des outils

- **Part** : une partie visible de page

- correspond à une instance d'éditeur, de vue : **EditorPart, ViewPart**
- **Part** : capture les fonctions sans relation avec l'extérieur
- Part est toujours associée à un Site : **EditorSite, ViewSite**
- **Site** capture les fonctions en relation avec l'extérieur



### le modèle d'UI (2)

- **Editeur** : outil d'édition de ressources
  - PAS de barre locale de menus (les menus spécifiques apparaissent dans la barre de menu de la fenêtre)
  - PAS de barre locale d'outils menus (les outils spécifiques apparaissent dans la barre d'outil de la fenêtre)
  - plusieurs instances d'un éditeur d'un type donné possibles dans une perspective
  - affichage dans l'editor area uniquement (stacked ou tiled)
- **Vue** : visualisation des ressources
  - éventuellement une barre locale d'outils
  - éventuellement une barre locale de menus
  - une seule instance de view d'un type donné dans une perspective
  - affichage n'importe où sauf dans l'editor area

différence de comportement entre un editeur et une vue : un editeur a un état « isDirty » exprimant la modification de la ressource associée

- **Perspective** : arrangement initial d'une page
  - **Editor Area**
  - **View**
  - **Placeholder, PlaceholderFolder**

## le modèle de ressources

- représentation des ressources d'un système de fichiers
- **ressource** Eclipse = handle sur une ressource d'un système de fichiers associée à :
  - un modèle de notification d'événements (création, destruction, modification, ...)
  - un modèle d'annotation général (**Property, Marker**)
  - une gestion d'historique

Synchronisation de la  
représentation non garantie !!

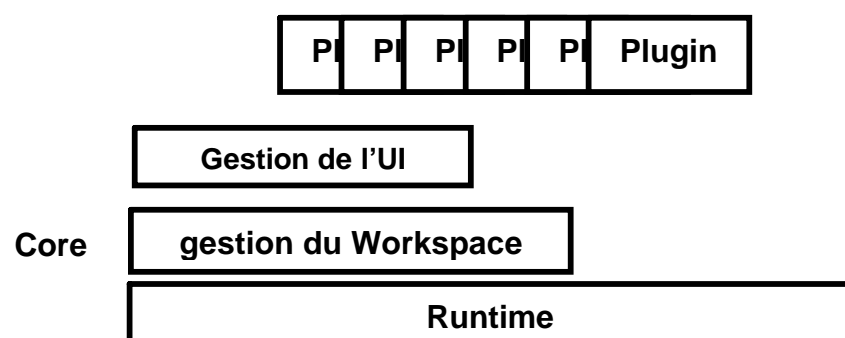
## les abstractions du modèle

- **Espace de travail (Workspace)** : un ensemble de **Projets**
- **Porjet (Project)** : une unité de gestion de ressources
  - un ensemble de Folders (répertoires) et de Files (fichiers)
- **Folder** : un ensemble de Folders (sous-répertoires) et de Files (fichiers)
- **File** : un conteneur d'information
  
- **Nature** d'un Project
  - Builder : un outil lancé à certains instants (souvent automatiquement sur événement ou périodiquement)
  - exemple : compilateurs (incrémentaux)
  - associé à un éditeur
  
- distinction entre **Primary Resource** et **Secondary Resource** (une ressource créée par un builder)

## L'architecture (1)

### une vue générale

- le noyau : runtime minimal assurant l'intégration des plugins (« composants Eclipse »)
- quelques plugins fondamentaux : mécanismes de chargement, gestion de l'UI, gestion des ressources, library plugins
- une architecture visant à minimiser l'occupation mémoire et ne charger (dynamiquement) que ce qui est nécessaire
  - le runtime = mécanismes de base pour le chargement dynamique de classes (basé sur osgi)
  - le runtime gère les dépendances entre plug-ins



A l'exception du runtime ....  
tout est plug-in (gestion de l'UI, gestion de  
ressources, ..., library plug-ins, ....)

### le point d'extension

- un point d'intégration d'un plugin
- spécifie les informations nécessaires
  - au chargement du plugin (avec prise en compte des dépendances)
  - à la mise en place de la communication avec les autres plugins
  - à la gestion graphique (affichage, interaction avec l'utilisateur)

## L'architecture (2)

### quelques points d'extension

- contribution à des points d'extension (fourniture des informations et du code nécessaire à l'intégration)

<b>org.eclipse.ui.actionSets</b>	<b>contribution à la barre de menu et/ou la toolbar de la fenêtre du Workbench</b>
<b>org.eclipse.ui.actionSetPartAssociation</b>	<b>Contribution à une action multi-perspective</b>
<b>org.eclipse.ui.editorActions</b>	<b>contribution à la barre de menu et/ou la toolbar de la fenêtre d'un editor</b>
<b>org.eclipse.ui.popupMenus</b>	<b>contribution à un popup menu d'un editor, view ou objet</b>
<b>org.eclipse.ui.viewActions</b>	<b>contribution à la toolbar et au menu pulldown d'une view</b>
<b>org.eclipse.ui.exportWizards</b>	<b>contribution au choix export du menu standard (par un nouveau wizard)</b>
<b>org.eclipse.ui.importWizards</b>	<b>contribution au choix import du menu standard (par un nouveau wizard)</b>
<b>org.eclipse.ui.newWizards</b>	<b>contribution au choix new du menu standard (par un nouveau wizard)</b>
<b>org.eclipse.ui.preferencePages</b>	<b>contribution aux Dialog de préférences</b>
<b>org.eclipse.ui.propertyPages</b>	<b>contribution au Dialog de propriétés</b>
<b>org.eclipse.ui.views</b>	<b>définition d'une view</b>
<b>org.eclipse.ui.editors</b>	<b>définition d'un éditeur</b>
<b>org.eclipse.ui.perspectives</b>	<b>définition d'une perspective</b>
<b>org.eclipse.ui.perspectiveExtensions</b>	<b>extension d'une perspective existante</b>
<b>org.eclipse.core.resources.builders</b>	<b>définition d'un builder</b>
<b>org.eclipse.ui.help</b>	<b>définition de l'aide en ligne</b>

## L'architecture (3)

### le contrôle de l'occupation mémoire et des performances : la stratégie de chargement (paresseux)

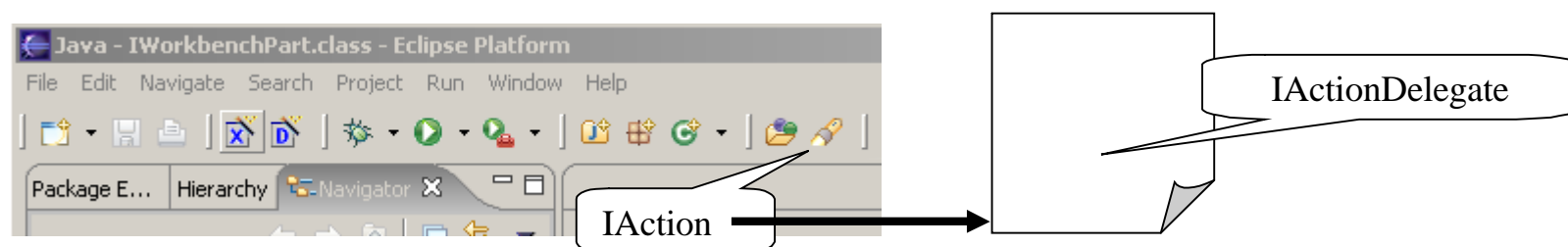
- repose sur des descripteurs XML de plugin (fichier plugin.xml)  
construction d'une représentation internalisée des plugins au démarrage (stockage dans le PluginRegistry)

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>

<plugin id="org.fho.tools.currencyConvertor" name="CurrencyConvertor Plug-in" version="1.0.0"
  provider-name="FHO" class="org.fho.tools.currencyConvertor.CurrencyConvertorPlugin">
  <runtime>
    <library name="currencyConvertor.jar">
      .....
    </library>

    <extension name="views convertisseur de devises" point="org.eclipse.ui.views">
      <category name="exemples fho" id="org.fho.tools.category1"/>
      <view class="org.fho.tools.currencyConvertor.views.CurrencyConvertorView0"
        .....
        id="org.fho.tools.currencyConvertor.view1" icon="icons/p-blue[1].gif"/>
    </extension>
  </runtime>
</plugin>
```

- distinction entre la représentation UI et l'action associée (cf ActionDelegate par la suite)



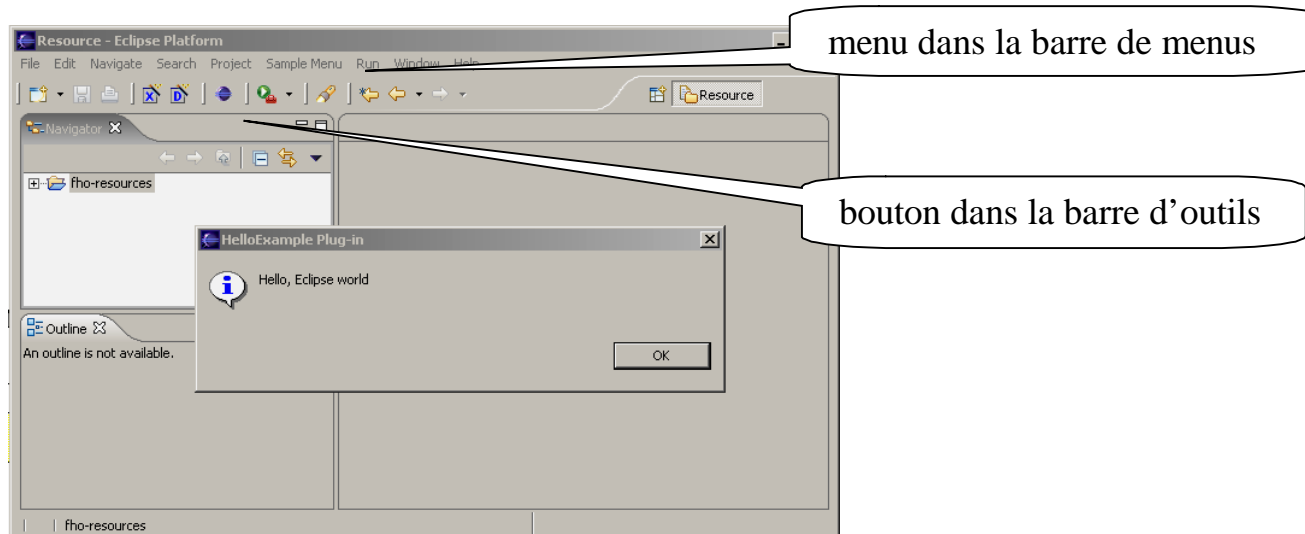
## Le développement d'un plugin (1)

### La démarche

- créer (réutiliser) un **Plug-in Project**
- définir les points d'extension requis et offerts (voir suite)
- définir le descripteur du plugin (plugin.xml)
- implémenter les fonctions des classes associées au plugin

### l'exemple incontournable depuis K&R !!

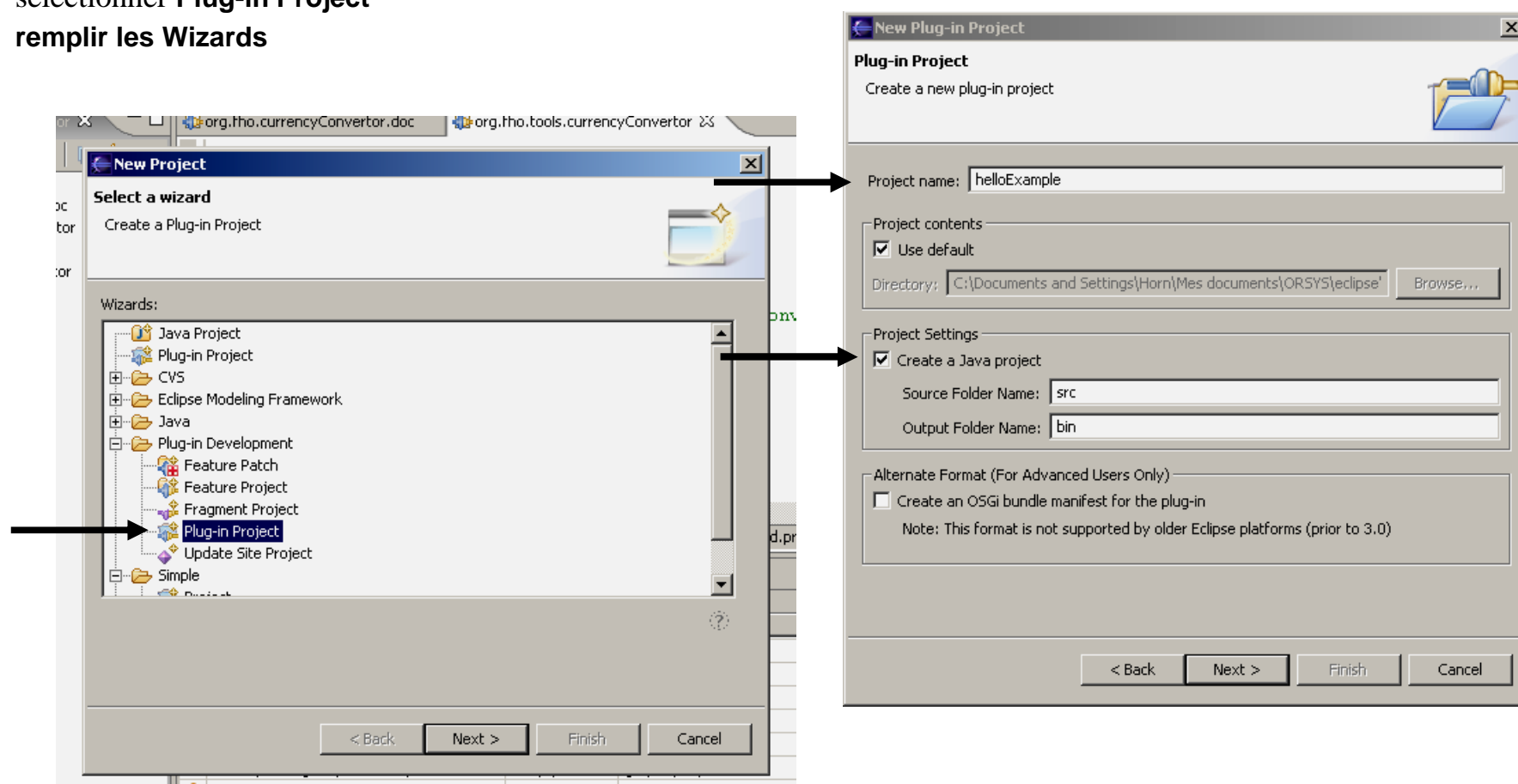
- affichage d'un `MessageDialog` via un menu (de la barre des menus) et un bouton (de la barre d'outils) de la fenêtre



## Le développement d'un plugin (2)

### la marche à suivre

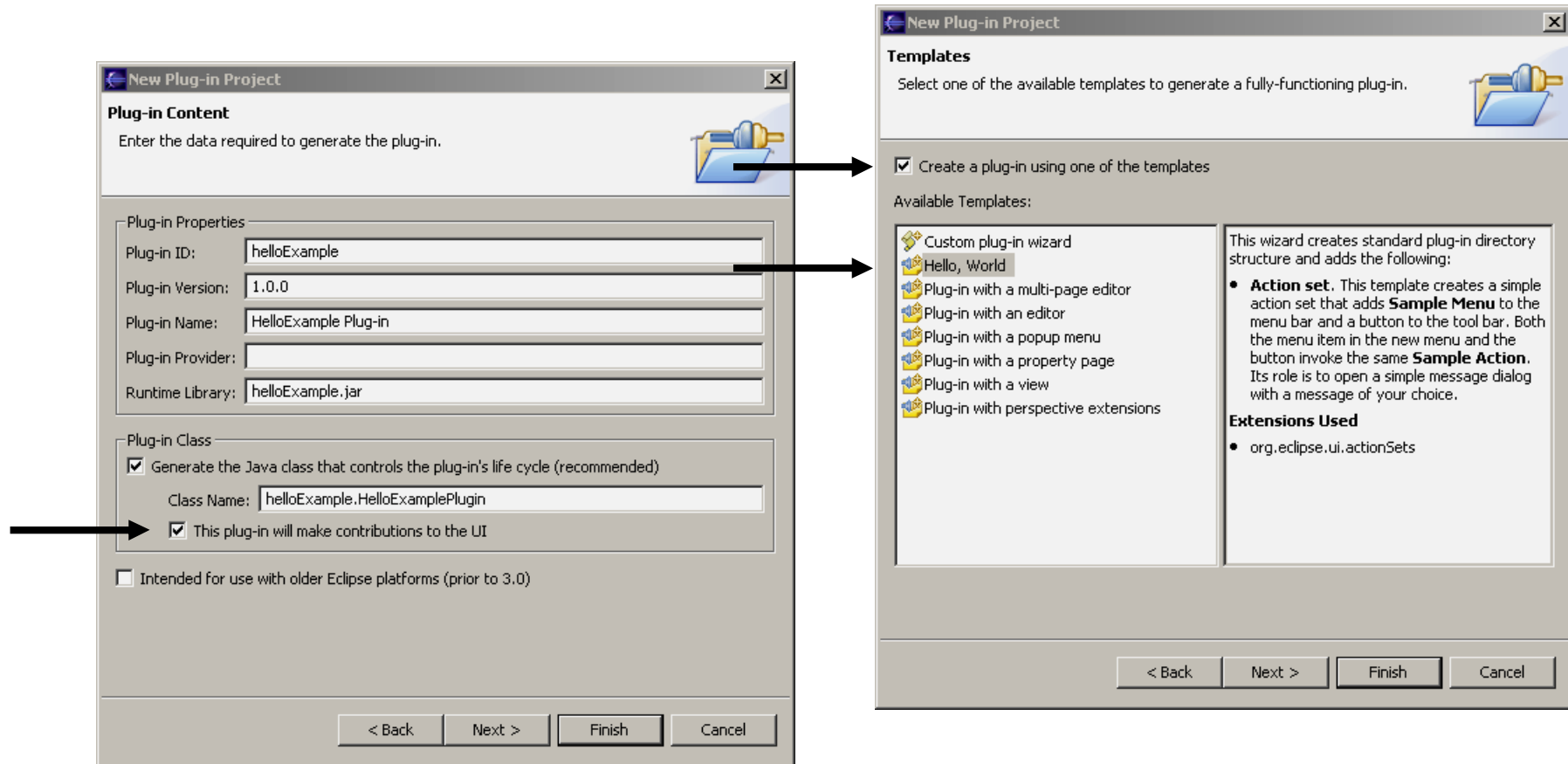
- créer un nouveau projet : **New > Project**
- sélectionner **Plug-in Project**
- remplir les Wizards



## Le développement d'un plugin (3)

### la marche à suivre (suite)

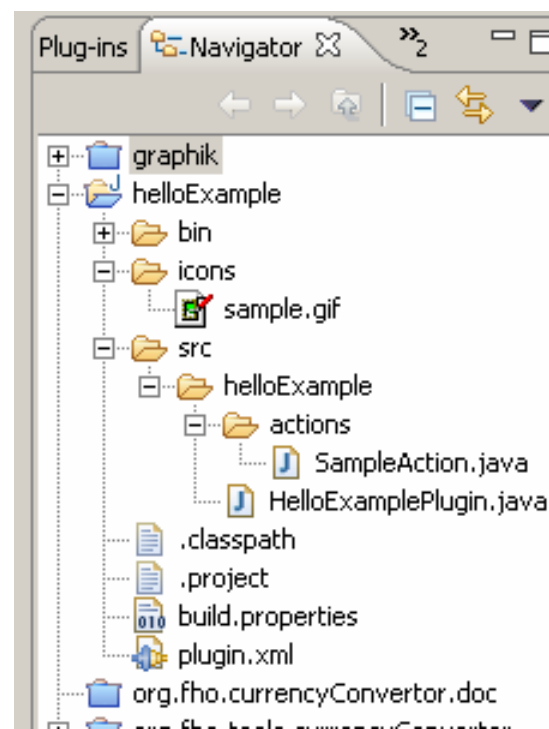
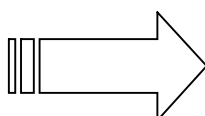
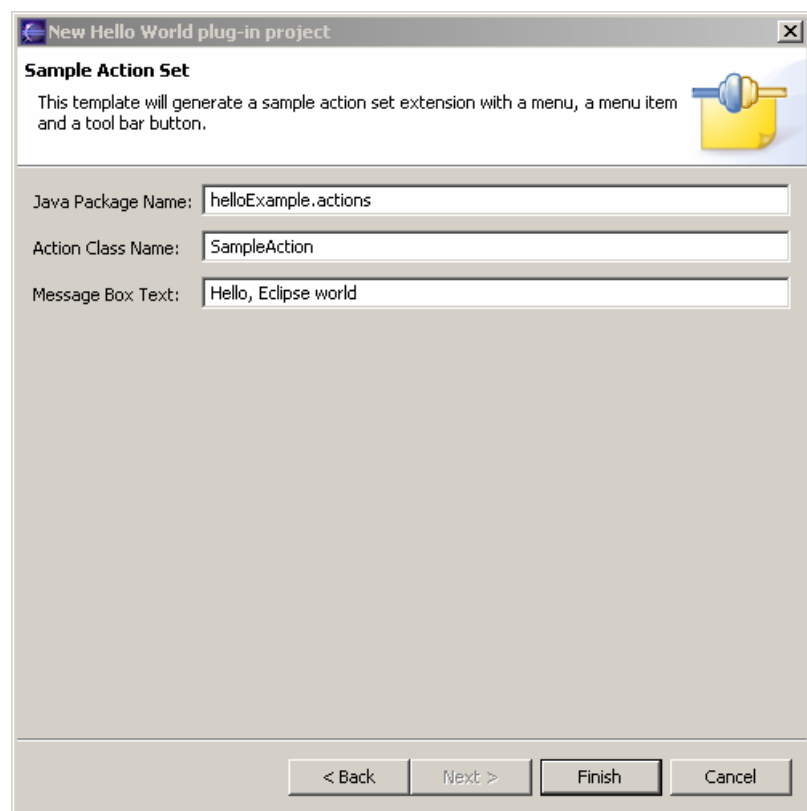
- les principales caractéristiques du plugin sont générées automatiquement
- choisir le template Hello World



## Le développement d'un plugin (4)

### la marche à suivre (suite)

- le Wizard crée un projet avec les classes java et les ressources



## Le développement d'un plugin (5)

### le descripteur plugin.xml généré

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin id="helloExample" name="HelloExample Plug-in" version="1.0.0"
  provider-name="" class="helloExample.HelloExamplePlugin">

  <runtime>
    <library name="helloExample.jar">
      <export name="*" />
    </library>
  </runtime>

  <requires>
    <import plugin="org.eclipse.ui" />
    <import plugin="org.eclipse.core.runtime" />
  </requires>

  <extension point="org.eclipse.ui.actionSets">
    <actionSet label="Sample Action Set" visible="true" id="helloExample.actionSet">
      <menu label="Sample &Menu" id="sampleMenu">
        <separator name="sampleGroup"></separator>
      </menu>
      <action id="helloExample.actions.SampleAction"
        label="&Sample Action" icon="icons/sample.gif"
        class="helloExample.actions.SampleAction"
        menubarPath="sampleMenu/sampleGroup" toolbarPath="sampleGroup"
        tooltip="Hello, Eclipse world">
      </action>
    </actionSet>
  </extension>
</plugin>
```

**l'ID unique du plugin**

**La classe java représentant le plugin**

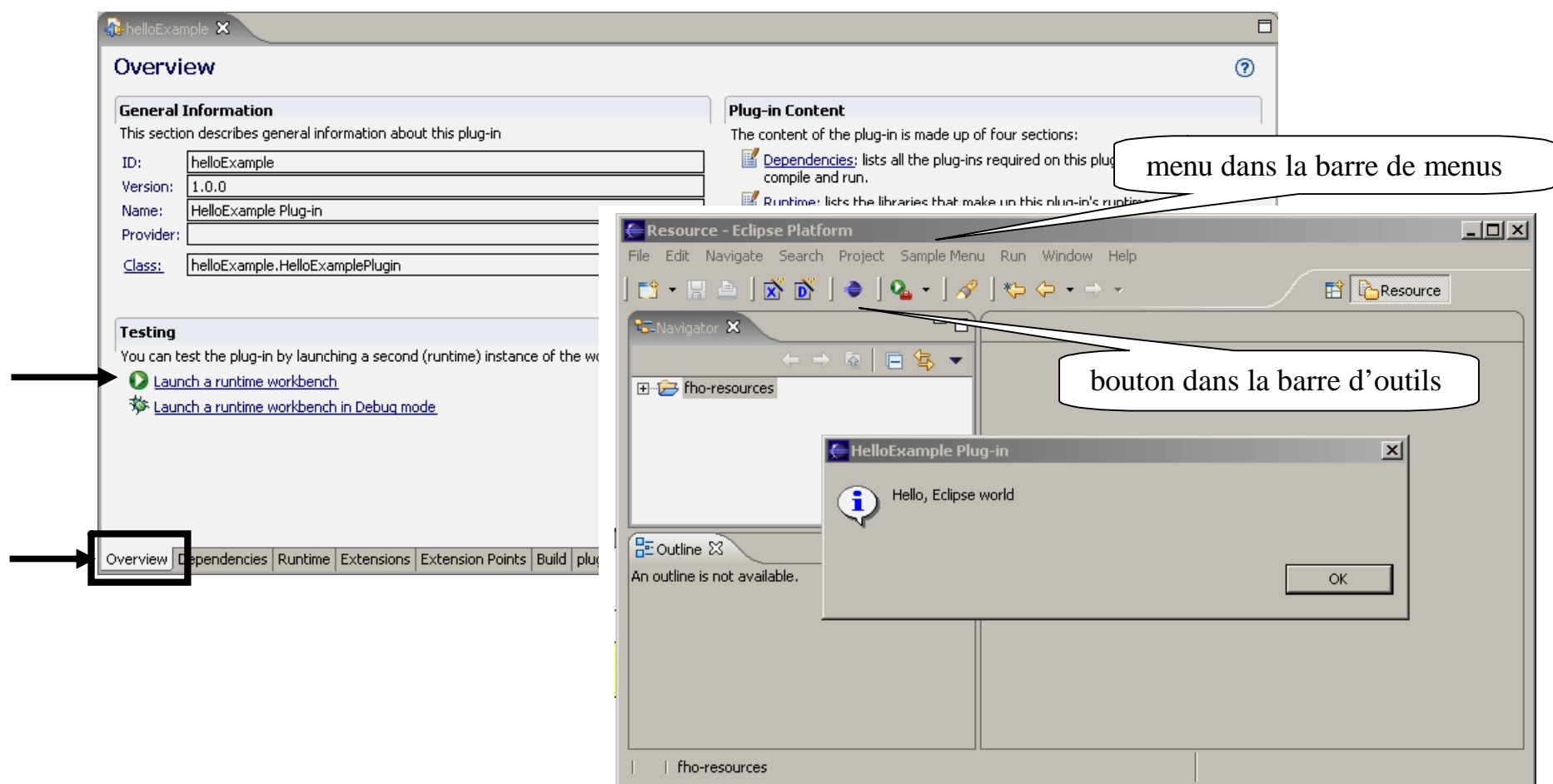
**les plugins requis : utilisés par le class loader pour construire son classpath !!!!**

**les caractéristiques UI**

**la classe ActionDelegate**

# Le développement d'un plugin (6)

## le lancement de l'application



## Le développement d'un plugin (7)

### l'affichage du message à partir du bouton ou du menu

un Delegate de la barre d'outil implémente :

```
public interface org.eclipse.ui.IWorkbenchWindowActionDelegate {  
    public void init(IWorkbenchWindow aWindow);  
    public void selectionChanged(IAction aAction, Iselection aSelection);  
    public void dispose();  
    public void run(IAction aAction);  
}
```

```
public class SampleAction implements IWorkbenchWindowActionDelegate {  
    private IWorkbenchWindow window;  
    public SampleAction() {}  
  
    public void init(IWorkbenchWindow window) { this.window = window; }  
    public void run(IAction action) {  
        MessageDialog.openInformation(window.getShell(), "HelloExample Plug-in", "Hello, Eclipse world");  
    }  
  
    public void selectionChanged(IAction action, ISelection selection) {}  
    public void dispose() {}  
}
```

## Le développement d'un plugin (8)

### les dépendances

- tous les plugins dont on utilise les classes et interfaces (NECESSAIRE au ClassLoader)

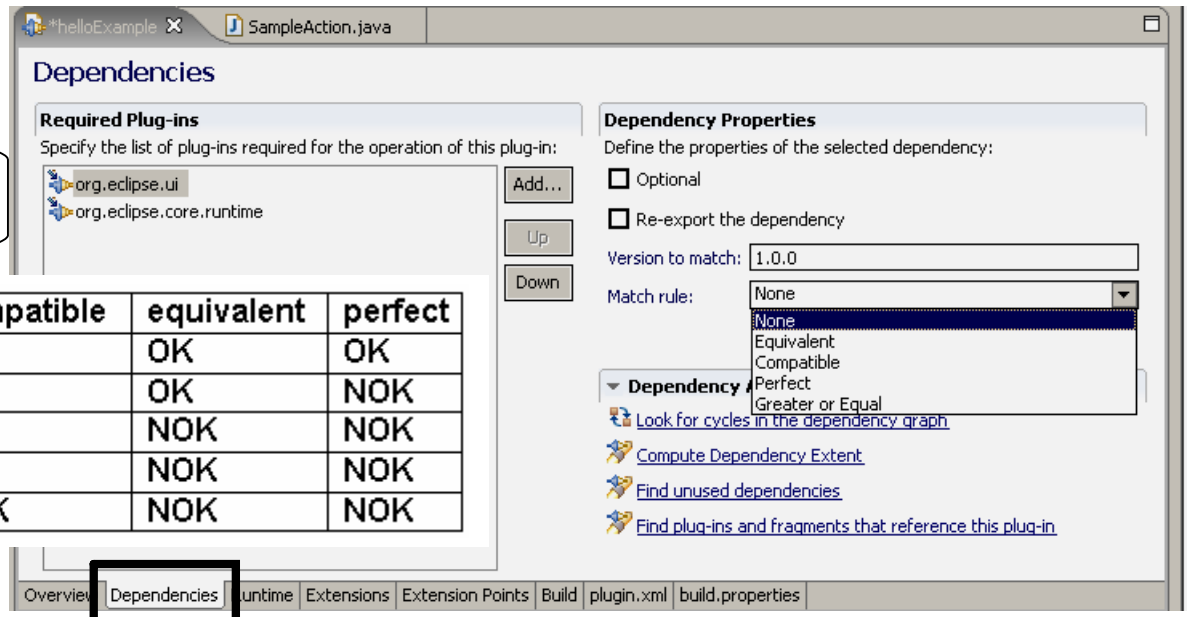
### les versions des plug-ins

- major.minor.service.qualifier (avec qualifier optionnel)
- le major définit les niveaux de compatibilité
- le minor définit des variations acceptables (si niveau de compatibilité est **compatible**)
- le service définit des variations acceptables (si niveau de compatibilité est **equivalent**)
- le qualifier est ignoré (sauf si niveau de compatibilité est **perfect**)

pas d'hypothèse de compatibilité ascendante

compatibilité des différentes versions, si la version requise est 1.0.0

version installée	greaterOrEqual	compatible	equivalent	perfect
<b>1.0.0</b>	OK	OK	OK	OK
1.0.1	OK	OK	OK	NOK
1.1.0	OK	OK	NOK	NOK
1.2.1	OK	OK	NOK	NOK
2.0.0	OK	NOK	NOK	NOK

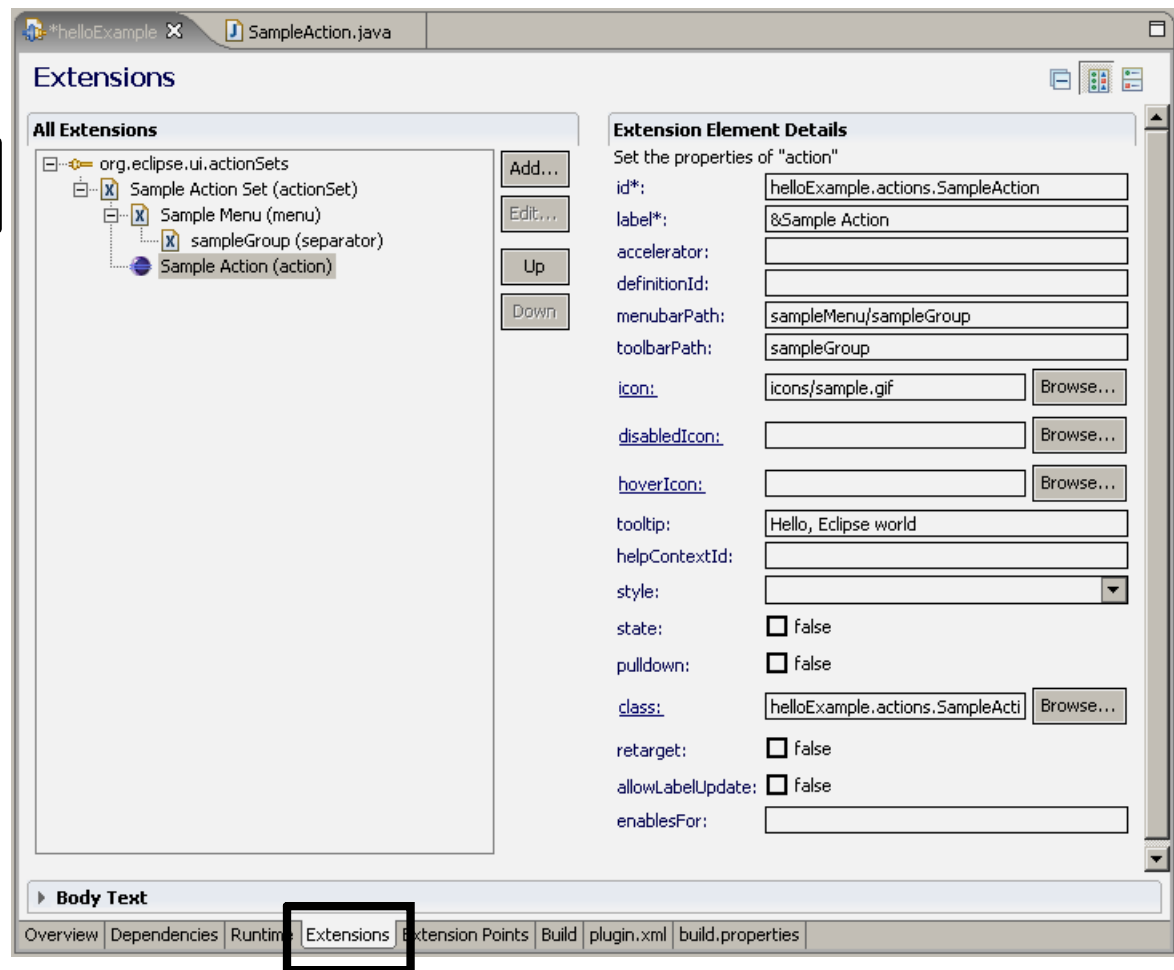


## les extensions

- indique les services UTILISES par le plugin

**NE PAS CONFONDRE  
Extension et Extension-points**

- éditeur associé à chaque extension



# L'arborescence (1)

## L'organisation générale des informations dans Eclipse 3.x

9 objet(s) (Espace disque disponible : 32)

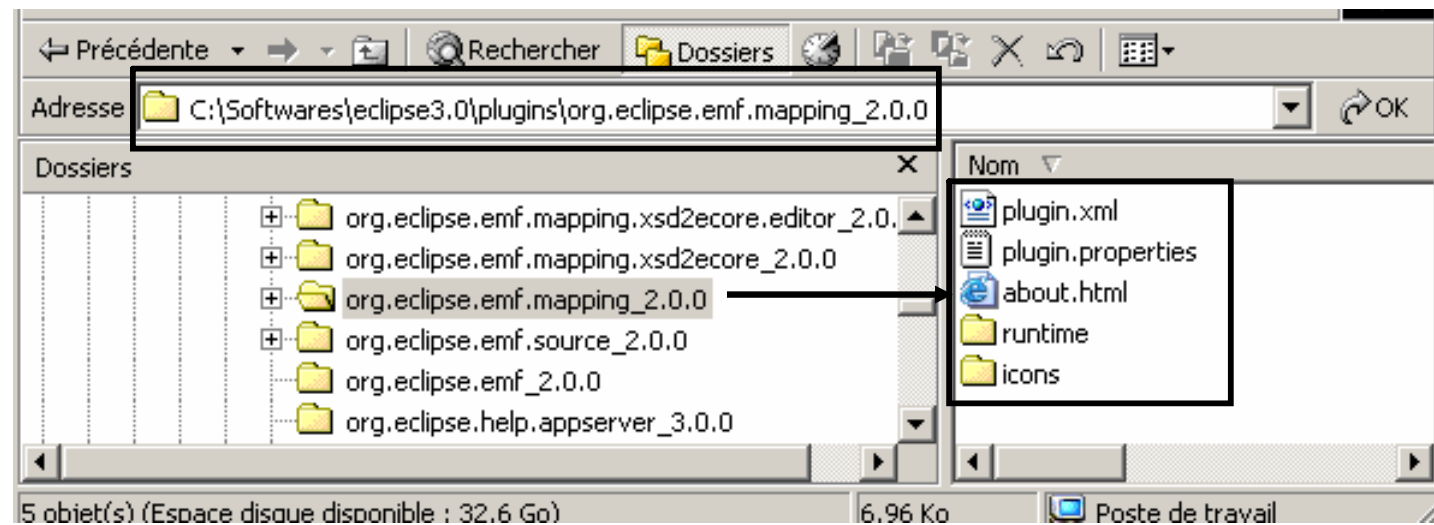
- cpl-v10 : common public license
- eclipse : lanceur du workbench
- startup : .jar nécessaire au démarrage
- .....

## L'arborescence (2)

### le répertoire plugins

- chaque répertoire plugin a un nom unique se terminant par `_numéro_version`
- chaque répertoire contient :
  - OBLIGATOIREMENT un fichier `plugin.xml`
  - EVENTUELLEMENT un fichier `plugin.properties` (internationalisation)
  - EVENTUELLEMENT un fichier `about.html`
  - EVENTUELLEMENT des répertoires pour les ressources propres au plugin
    - répertoire `runtime` (contenant le code)
    - répertoire `icons` (contenant les icônes)
    - .....

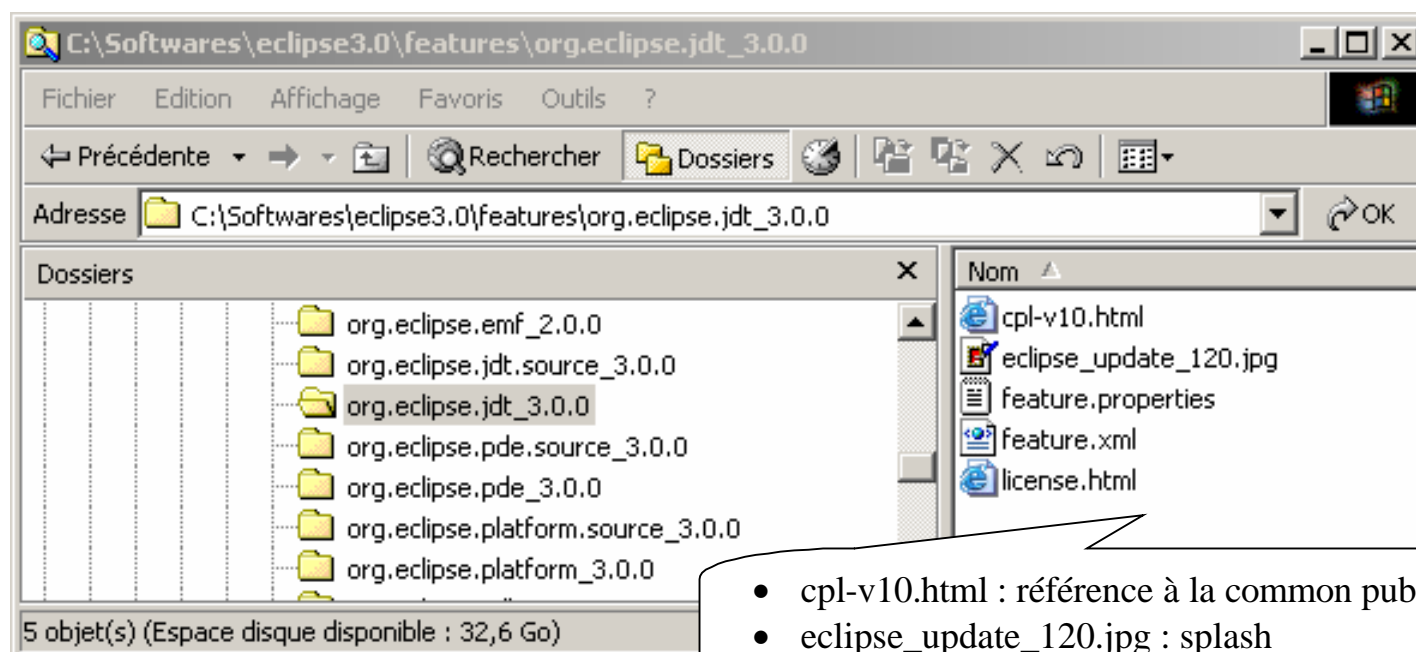
`<major>.<minor>.<release>_<qualifier>`



## L'arborescence (3)

### le répertoire features

- chaque répertoire feature a un nom unique se terminant par `_numéro_version`
- chaque répertoire contient :
  - OBLIGATOIREMENT un fichier `feature.xml`
  - EVENTUELLEMENT un fichier `feature.properties` (internationalisation)
  - .....

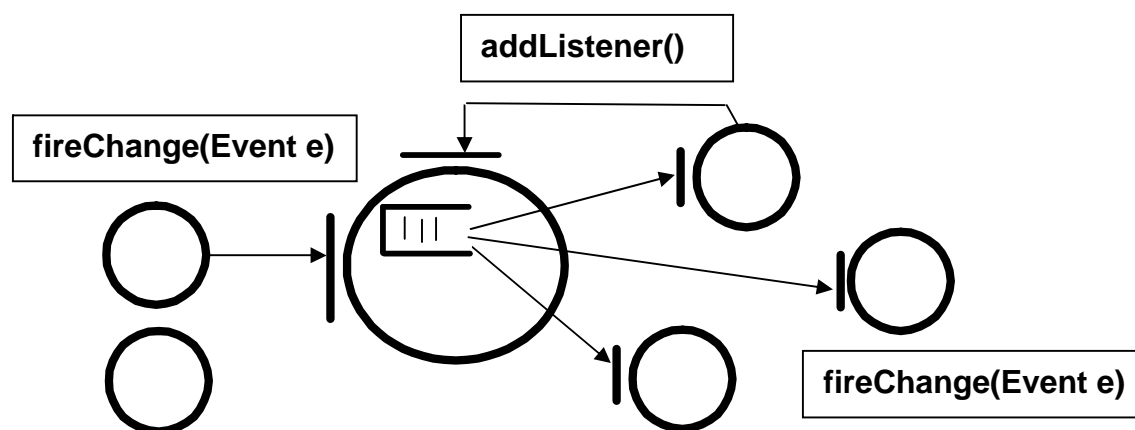


- `cpl-v10.html` : référence à la common public license
- `eclipse_update_120.jpg` : splash
- `features.properties` : internationalisation

## L'architecture (4)

### les abstractions de contrôle

- Design Pattern d'Observer : **Observer, Listener, Event**



### quelques observateurs intégrés

- observateur des modifications sur les ressources (intégré au niveau Workspace)
- observateur de l'état des WorkbenchPart (intégré au niveau de la WorkbenchPage)
- observateur de Selection (intégré au niveau de la WorkbenchWindow)

ResourceChangeService et ResourceChangeListener

PartService et PartListener

SelectionService et SelectionListener


### le contrôle du footprint et des performances : les conditions d'activation

- le runtime gère des conditions d'activation-inactivation :
  - prise en compte du type des objets sélectionnés
  - prise en compte du statut et de l'état des ressources sélectionnées
  - prise ne compte des propriétés de la JVM
  - .....

### le contrôle de : l'ordonnancement et la réactivité de l'UI

- le runtime propose un contrôle des activités asynchrones (Job)

```
class TrivialJob extends Job {  
    public TrivialJob() { super("un job trivial") ; }  
    public IStatus run(IProgressMonitor mon) { ..... return IStatus.OK_STATUS; }  
}  
  
.....  
Job job = new TrivialJob();  
job.schedule();  
.....
```



- le JobManager : contrôle l'exécution des jobs en assurant l'interface avec les Threads Java sous-jacents  
**IStatus schedule(), void cancel(), void sleep(), void join(), .....**

## Les interfaces IPluginDescriptor et IPlugin

### le rôle

- **IPluginDescriptor** est une interface de manipulation de la représentation java du plugin.xml
- **Plugin : Singleton** offrant un point d'accès aux :
  - ressources statiques associées au Plug-in
  - aux préférences spécifiques et informations d'état
  - initialisation/finalisation

### le chargement

- initialisation au chargement
- finalisation au déchargement
- la méthode **startup()** est invoqué au chargement des classes (pas du Plug-in)
- chargement au démarrage obtenu par :
  - `<extension point= "org.eclipse.ui.startup">`
  - implémentation de l'interface `org.eclipse.ui.IStartup`

```

public void startup() throws CoreException {
    super.startup();
    .....
}

public void shutdown() throws CoreException {
    super.shutdown();
    .....
}
    
```

toujours invoquer la méthode parente

### l'accès aux ressources statiques

- accès aux ressources déclarées dans le plugin.xml
  - `URL find(IPath resPath)`
  - `InputStream openStream(IPath resPath)`

### Utilisation d'Eclipse

- K. Djaafar : Développement J2EE avec Eclipse et WSAD, Eyrolles 2004
- S. Holzner : Eclipse, O'Reilly 2004
- S. Shavor, J. d'Anjou, S. Fairbrother, D. Ken, J. Kellerman, P. McCarthy : The Java™ Developer's Guide to Eclipse, Addison Wesley, 2003

### Construction de plug-ins

- J. Arthorne, C. Laffra : Official Eclipse 3.0 FAQs, in the eclipse series, Addison Westley 2004
- E. Clayberg, D. Rubel : Eclipse, Building Commercial Quality Plug-ins, in the eclipse series, Addison Westley 2004
- E. Gamma, K. Beck : Contributing to Eclipse, Principles, Patterns and Plug-Ins, in the eclipse series, Addison Westley 2004
- S. Shavor, J. d'Anjou, S. Fairbrother, D. Ken, J. Kellerman, P. McCarthy : The Java™ Developer's Guide to Eclipse, Addison Wesley, 2003
- R. Warner, R. Harris : The Definitive Guide to SWT and JFace, Apress, 2004

### EMF

- F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, T. J. Grose : Eclipse Modeling Framework, , in the eclipse series, Addison Westley 2004