2. Le modèle du Plug-in Eclipse

Au sein d'une instance d'Eclipse qui tourne, nous trouvons le plug-in incorporé dans les classes comme "plug-in runtime class " ou bien" plug-in class". Ces classes fournissent le support pour configurer et gérer l'instance de plugin. Une classe plug-in en Eclipse doit étendre org.eclipse.core.runtime.Plugin, qui est une classe abstraite qui regroupe des méthodes génériques pour faciliter la gestion des plugins

Un plug-in est descrit dans un fichier XML : plugin.xml, qui se trouve dans le dossier du plugin. Le fichier manifest sert à ce que l'instance d'Eclipse (eclipse runtime) puisse l'activer.

2.1. Le déploiement d'un Plug-in et son Activation

Relations entre les plug-in s (spécifiés dans fichier manifest) :
1. Dépendance : un plug-in a besoin d'installation d'un autre plug-in prérequis pour fonctionner et pour le supporter ou relation du dépendance. [Element requires dans fichier XML]
2. Extension : role host plug-in ou extender plug-in. L'extender plug-in extend les fonctionnalités d'autre appelé host plug-in. [Element extension dans fichier XML]

Ces plug-ins sont utilisé quand les autres de base ont besoin d'etre suportés ou extensibles

2.3. Extension d'un Plug-in

Un plug-in peut être complété par des nouveaux éléménts, ce processus s'appelle une extension. Les éléments ajoutés peuvent être des éléments d'UI , comme par exemple pour que l'utilisateur puisse accéder au plug-in "help" on doit mettre en place le menu help , ou d'autres éléments de traitement.

Chaque plug-in peut etre étendu par des éléments de traitement. ces derniers modifient le comportement du plug-in host selon les services customisées définies dans le extender plug-in.

Dans les cas simples, un seul acte d'extension génére un seul "callback object" à partir duquel les plug-ins host et extender communiquent entre eux. Les classes callback sont identifiées par leur nom dans la partie de la déclaration de chaque extension dans le plug-in extender.

Un plug-in peut se permettre d'être augmenté par plusieurs types différents d'extensions. Pour cela il faut qu'il déclare des différents "slots" (fentes, portes ?) qui vont servir à brancher les extensions. Il s'agit des extension-points, (points d'extensions). Sur un extension-point peuvent venir se brancher plusieures extensions.
Donc les mots clés sur lesquels on va s'appuyer pour décrire le rôle de chaque objet dans une extension sont : Host plug-in, extender plug-in et callback object.

2.3.1 Les Acteurs d'une Extension

1) Host Plug-in

C'est le plug-in qui offre des points d'extensions, qui contrôle et coordonne plusieurs extensions. Dans son fichier manifest le point d'extensions est declaré : <extension-point id='actionsSets' …..> plug in étendu dans ce cas est actionsSets.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
1  id="org.eclipse.ui"
   name="Eclipse UI"
   version="2.1.0"
   provider-name="Eclipse.org"
   class="org.eclipse.ui.internal.UIPlugin">

2  <extension-point id="actionSets" name="Action Sets"
     schema="schema/actionSets.exsd"/>
   <!-- Other specifications omitted. -->
</plugin>
```

Listing 2.3. Declaring an Extension-Point.

2) Extender Plug-in

Il défini l'extension, assure d'être compatible et disponible pour un certain host plug-in (1er rôle) et également pouvoir ajouter des nouveaux éléments et fonctionnalités dans son environnement.
Déclaration dans le fichier manifest : extension . Cet élément a plusieurs attributs comme: point pour faire la liaison avec le point d'extension du host plug-in ;
action : regroupe tous les sous-attributs nécessaires pour décrire et réaliser les fonctionnalités qui vont être appelées par le host plug in.

```
<plugin
        id="org.eclipse.help.ui"
        name="Help System UI"
        version="2.1.0"
        provider-name="Eclipse.org"
        class="org.eclipse.help.ui.internal.WorkbenchHelpPlugin">
    <!-- ... -->
    <!-- Action Sets -->
    <extension
        1 point="org.eclipse.ui.actionSets">
        <actionSet
                label="Help"
                visible="true"
                id="org.eclipse.help.internal.ui.HelpActionSet">
        2 <action
                label="&Help Contents"
                icon="icons/view.gif"
                helpContextId="org.eclipse.help.ui.helpContentsMenu"
                tooltip="Open Help Contents"
                class="org.eclipse.help.ui.internal.HelpContentsAction"
                menubarPath="help/helpEnd"
                id="org.eclipse.help.internal.ui.HelpAction">
            </action>
            <!-- ... other actionSet elements -->
        3 <action
                label="&Help..."
                icon="icons/search_menu.gif"
                helpContextId="org.eclipse.help.ui.helpSearchMenu"
            4 class="org.eclipse.help.ui.internal.OpenHelpSearchPageAction"
                menubarPath="org.eclipse.search.menu/dialogGroup"
                id="org.eclipse.help.ui.OpenHelpSearchPage">
            </action>
        </actionSet>
    </extension>
<!-- ... -->
</plugin>
```

Listing 2.4 Declaring an Extension

3) Le rôle du Callback

L'objet qui joue le rôle de callback est un objet Java (a plain old Java object ??) qui est appelé par le host plug-in quand certains événements spécifiés dans le contrat de l'extension-point correspondant sont reconnu par le host plug-in.
L'interface pour les objets callback est fourni par le plug-in host, et est documentée dans la documentation de l'extension-point qui va être étendu.
L'implémentation des callback objects est défini dans une classe spécifique à l'extension en question et est fournie par le fournisseur du plug-in extender (the provider of the extender plug-in). Or, comme l'implémentation du callback object dans le extender références l'interface callback , qui elle se trouve dans le package du plug-in host, un plug-in extender dépends aussi du plug-in host.

3. Extension Processing

Comment est programmée et réalisée une extension , comment on écrit le code du host plug-in pour chaque point d'extension ?

Quand le host plug-in est activé, les extensions déclarées dans les autres plug-ins doivent être procédées.
La configuration d'un point d'extension est quasi-arbitraire,elle est connue par le host plug-in et le schéma de définition d'un point d'extension est produit par le design du host plug-in.
Un host plug-in afin de procéder une extension, il faut avoir la liste de ces extensions par le Eclipse runtime et pour chaque extension , une version parse des membres d'extension.
Il existe un API registre qui permet la programmation des représentations parsés des spécifications de plug-in.

La classe ProcessExtensions fournit une méthode de traitement d'une extension **process** (1), qui boucle d'abord sur toutes les extensions d'une point d'extension (2), ensuite pour chaque extension , boucle sur tout les membres de cette extension (4).

```
package com.bolour.sample.eclipse.demo;

import org.eclipse.core.runtime.IConfigurationElement;
import org.eclipse.core.runtime.IExtension;

public interface IProcessMember {
    public Object process(IExtension extension,
      IConfigurationElement member);
}
```

```
package com.bolour.sample.eclipse.demo;

import org.eclipse.core.runtime.IConfigurationElement;
import org.eclipse.core.runtime.IExtension;
import org.eclipse.core.runtime.IExtensionPoint;
import org.eclipse.core.runtime.IPluginRegistry;
import org.eclipse.core.runtime.Platform;

public class ProcessExtensions {
1 public static void process(String xpid, IProcessMember
processor) {
        IPluginRegistry registry =
Platform.getPluginRegistry();
        IExtensionPoint extensionPoint =
            registry.getExtensionPoint(xpid);
        IExtension[] extensions =
extensionPoint.getExtensions();
```

```
        // For each extension ...
    2   for (int i = 0; i < extensions.length; i++) {
            IExtension extension = extensions[i];
    3       IConfigurationElement[] elements =
              extension.getConfigurationElements();
            // For each member of the extension ...
    4       for (int j = 0; j < elements.length; j++) {
                IConfigurationElement element = elements[j];
    5           processor.process(extension, element);
            }
        }
    }
}
```

Le lien :

http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html#3.