Conception de Logiciels Extensibles

Manuel maintenance

Brian GOHIER, Yannis GREGO, Kevin MOKILI, Pierre NOYON, Aida TAPSOBA
28 avril 2013

Table des matières

1 Installation et lancement de la plate-forme

La plate-forme développée en tant que projet eclipse.
Pour installer la plate-forme il faut extraire tous les dossiers de l'archive dans le *workspace* d'eclipse. Le projet principal est "**ProjectCLE**" de l'archive. Vous pouvez l'importer via l'onglet *importer* d'éclipse (puis *from existing source*).
Dans ce projet nous avons un répertoire "**plugins**" qui fait référence à tous les répertoires de plugins que nous souhaitons associer à la plate-forme et un fichier de configuration propre à la plate-forme. Ce répertoire de plugins contient les fichiers binaires des plugins qui pourront être chargés par la plate-forme.
Il faut mettre à jour le "**pluginPath**" du fichier de configuration. Par défaut le chemin pris en compte sera celui qui est présent dans le dossier du projet principal si le chemin entré est incorrect.
Exemple : "**pluginPath = /home/rep1/rep2/plugins**"

Ensuite chaque plugin est répertorié dans ce fichier :
NomPlugin = Nom du répertoire du plugin contenant les fichiers binaires (paquages éventuels et "**config.ini**").
Exemple : "**MyPlugin = MyPluginPath**"
La plate-forme se lance ensuite en lançant l'exécutable du coeur (effectuer un *run* de la classe core).

1.1 Structure de la plate-forme

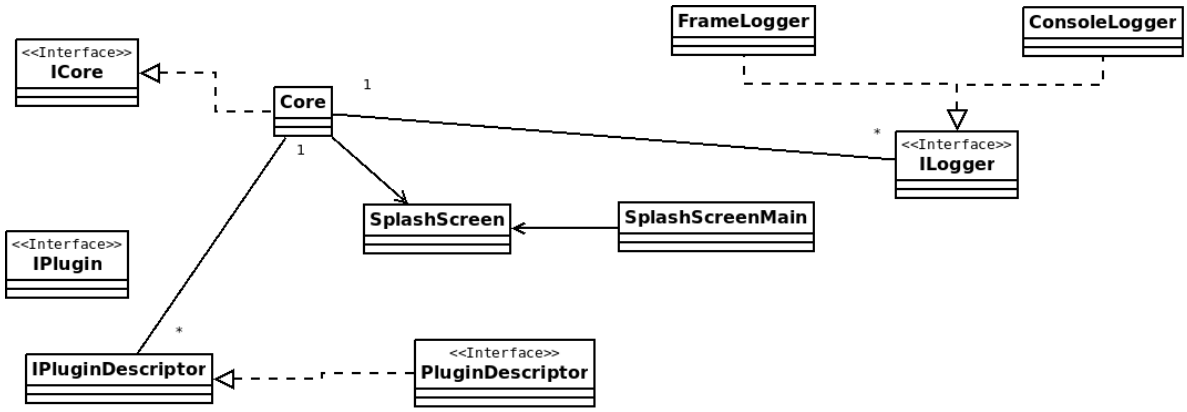

FIGURE 1 – Diagramme de classe de la plate-forme

2 Traitement des plugins

A partir du fichier de configuration du coeur nous avons donc le répertoire de chaque plugin. Depuis ce répertoire sont chargés les fichiers de configuration des plugins qui vont créer des instances de *IPluginDescriptor* qui seront en fait les descripteurs de plugin, c'est-à-dire un objet référençant le nom du plugin, le nom du dossier dans lequel sont placés ses fichiers binaires, le nom de sa classe principale et tous ses autres attributs qui permettront de configurer le plugin.
La classe "**Core**", possède un attribut "**plugins**", c'est une liste (*ArrayList*), de *IPluginDescriptor* qui liste donc tous les plugins disponibles.
Pour le fonctionnement de chaque méthode du coeur vous pouvez vous référer à la javadoc présente dans le dossier "**doc**" du projet principal.

3 Fonctionnement de la plate-forme

3.1 Constructeur "Core()"

La plate-forme possède un attribut "**fileName**" qui correspond au nom du fichier de configuration de la plate-forme. Ici nous l'avons définit "**config.ini**".

3.2 loadCore(), loadConfigs()

A l'instanciation du coeur on va lire dans ce fichier pour définir le chemin vers le dossier contenant tous les plugins ainsi que chaque plugin en leur associant le dossier qui leur correspond.

3.3 getPluginConfig(pluginName, pluginPath)

Ensuite pour chacun de ces plugins on va lire son fichier de configuration pour y associer ses paramètres dans son descripteur (*IPluginDescriptor*).

3.4 loadDefaultPlugins()

Puis nous allons charger tous les plugins par défaut de la plate-forme. Il s'agit de tous les plugins qui ont leur attribut "**default**" à **true**. Les autres ne seront chargés qu'à la demande de l'utilisateur ou aux besoin d'un plugin.

4 Possibilités de la plate-forme

4.1 Libraries externes

La plate-forme est en mesure de charger dans le "**classloader**" du plugin les librairies externes dont il peut avoir besoin. Il suffit de placer les libraries dans un dossier "**libs**" à placer à la racine du dossier contenant tous les plugins ("**plugins**" par défaut). Il faut ensuite pour ce plugin spécifier le nom de ses librairies séparés par des virgules (Vous pouvez regarder l'exemple du plugin "**PluginManager**" qui lui utilise 3 librairies externes).

4.2 Dépendance à un plugin

Un plugin qui utilise d'autre plugins dans son code (exemple du plugin "**PluginEditor**") va devoir déclarer quelles *Interfaces* ou **Classes** vont dépendre ses sous-plugins. Ainsi ces derniers ne pourront être chargés que si le plugin principal est chargé. Lors du chargement du plugin principal on regarde dans la liste des plugins du coeur lesquels ont pour attribut "**interfaces**" le même nom d'interface que celui que le plugin principal possède en attribut "**dependencies**". Ainsi le "**classloader**" du plugin principal comprendra le chemin vers ce plugin-ci puis ceux vers ses sous-plugins. Quand un sous-plugin sera chargé il utilisera le "**classloader**" du plugin principal.
Il est possible de mettre n'importe quel nom en tant qu'"**interfaces**" ou "**dependencies**" du moment qu'ils correspondent, mais il est préférable d'utiliser le chemin vers une interface existante du plugin principale car avec le gestionnaire de plugin "**PluginManager**" il est possible de créer directement un sous-plugin qui contiendra une librairie *.jar* qui contient l'interface passée dans l'attribut "**dependencies**" du plugin principal.

4.3 Gestion des plugins actifs

Certains plugins peuvent avoir accès aux fonctions déclarées dans l'interface du core (*ICore*, voir Javadoc). Dans ce cas ce plugin doit avoir un constructeur qui prend en paramètre cette interface, ainsi que l'attribut "**active**" défini à **true**. Lors du chargement de la plateforme on créera donc une instance à partir de ce constructeur, contrairement à un plugin simple dont on créer une instance à partir de la classe (utilisation du constructeur par défaut dans ce cas). Si jamais le fichier de configuration n'a pas son attribut "**active**" à **true**, lors du chargement de l'instance du plugin ("**getPluginInstance(String pluginName, Class< ?> classe, boolean active**") avec active à **false**) on va cherche à l'instancier via la création d'instance de la classe, ce qui va normalement renvoyer « null » et le coeur va alors rappeller cette fonction avec le paramètre "**active**" à **true**.