Rapport de stage - Coloane

Monir CHAOUKI

Janvier 2008

Le but de mon stage est, la concéption d'un point d'extension, et la réalisation d'une extension pour Coloane. La concéption de points d'extensions, permettra à Coloane, d'intégrer de nouvelles fonctionalités, via les contributions d'autres développeurs (en respectant certaines conditions), sans que ceux–ci, est à modifier le code de Coloane.

1 Notions impotrtantes

Nous allons présenter quelques notions, avant de passer à la présentation de Coloane, et aux mécanismes mis en jeu pour la conception et à la réalisation d'un point d'extension.

1.1 Plugin

Un plugin, est un programme qui s'ajoute à une application principale, pour lui offrire de nouvelles fonctionnalités. Il existe plusieurs applications qui proposent d'ajouter des plugins. Par exemples, le navigateur internet "Mozilla FireFox", propose plusieurs plugins allant du simple traducteur, aux lecteur de MP3 intégré au navigateur. Tous ces plugins sont issus des contributions de developpeurs.

1.2 Extensions

Une extension peut être vue comme un plugin limité. En faite, une extension, permet d'ajouter de nouveaux services, à une fonctionnalité déjà présente dans une application. Reprenons notre précédent exemple sur le navigateur internet "Mozilla FireFox". Comme il a été dit plus haut, "Mozilla FireFox" proposent des plugins qui permettent de traduire des pages, dans une langue spécifique. Il est possible, si les plugins le permettent, d'ajouter de nouvelle langue au traducteur, via les contributions d'autres developpeurs, si ceux–ci developpent un extensions qui s'ajoute aux plugins existant.

1.3 Points d'extensions

Nous avons vue, qu'une extension, est un service qui venai s'ajouter aux fonctionnalités, d'une application (ou d'un plugin). Pour qu'une extension puisse s'ajoute à une application (ou à un plugin), il faut que celle–ci déclare un point d'extension. La déclaration de points d'extensions, permet ainssi, à une application d'être extensible, et d'offrire de nouveaux services, venant d'autres contributions. Donc, un point d'extension, permet d'indiquer, aux extensions où elle doivent venire se greffer, pour qu'elle puisse enrichire une application (ou un plugin).

2 Coloane

2.1 Presentation de Coloane

Coloane est un plugin, pour Eclipse, qui permet de modéliser, les réseaux de Petri. L'une de ses philosophies est d'être multi–platforme. Le moteur physique, utilise un format non standard, le CAMI, pour travailler, sur ces réseaux de Petri.

2.2 Evolutions de Coloane

L'une des évolutions trés impotrantes, à mettre en place pour Coloane, c'est d'offrire la possiblité d'utilsier différents formats de représentations de réseau de Petri, permettant ainssi l'échange de différents formats de fichiers.

En effet, aujourd'hui, l'inconveniant majeur de Coloane commme nous l'avons écrit plus haut, c'est qu'il n'utilisent qu'un format, non standardiser le : CAMI. Or, il existe plusieurs formats pour la représentation des réseaux de Petri, dont un principale en cours de normalisation le : PNML.

2.3 Limites liés aux l'evolutions de Coloane

Aujourd'hui, il est biensûre possible de faire évoluer Coloane, mais cela implique d'entrer à l'intèrieur du code, et de le modifier directement pour ajouter les évolutions voulus.

Or, le fait d'entrer dans le code, de Coloane, qui est actuelement stable, est un facteur important d'apparition de bugs. De plus, si plusieurs développeurs souhaitent contribuer au projet Coloane pour le faire évoluer, le code risque vite d'être confus, sans régles etablies.

Il serait donc, intéresant de pouvoire faire évoluer Coloane, sans toucher au code. Le fait de faire évoluer Coloane correspondrait, à ajouter de nouvelles fonctionnalitées ou de nouveaux services. Or, ajouter des fonctionnalitées ou services, c'est ajouter un plugin ou une extension.

Par conséquant, si l'on veut faire evoluer Coloane, il faudrait songer à utiliser des plugins qui se mettent à coter de lui, ou des extensions qui viennent se greffer à lui.

3 Convértiseur de formats de représentations de Réseau de Petri

Nous avons vue précedement, que l'une des évolutions importantes pour Coloane, à mettre en place, est un convértiseur de format de représentations de Réseau de Petri, permettant ainssi à Coloane d'importer et exporter différents formats dont, le plus importants le : PNML.

3.1 Solutitions envisagées

Comme nous l'avons écris, deux solutions sont envisagales pour permettre à Coloane d'integré de nouvelle fonctionnalités et d'être évolutif (i.e. intégrer de nouveaux formats) : un plugin ou un point d'extension.

3.1.1 le plugin

Dans le cas d'un plugin, Coloane, n'aura aucune mettrise au niveau de l'intérface graphique qui permet d'intéragir avec l'utlisateur pour l'importations et l'exportations de nouveaux format . Toutes les contributions pourrons gérer l'intérface graphique,et implementer les fonctions d'importations et d'exportations de nouveaux format, comme bon leurs semblent.

3.1.2 le point d'extension

Dans le cas d'un point d'extension, Coloane gérera toujours l'intérface graphique, mais délèguera aux extensions qui viendront se greffer sur ce point, seulement, les fonctions d'importations et d'exportations de nouveaux formats et ceux–ci devrons respecter certaines régles.

3.2 Solutions choisies

Il est bien évident que nous allons choisir d'utiliser un point d'extension dans Coloane pour que l'on est un certaines maîtrise, des contributions qui permetteront de faire des importations et des exportations de nouveaux format.

En effet, chaque contributions devrons avoir la même intérace graphique pour interagire avec l'utilisateur, pour que celui–ci, ne soit pas perturber lorsqu'il veut exporter ou importer dans des formats diffférent. De plus, on imposera aux contributions de respecter certaines régles qui est en faites : une intérfaces à implementer.

Ainssi toutes nouvelles contributions offrants de nouveaux format n'aura qu' à offrire ses service en utilisant le point d'extension définie par Coloane.

4 Précisions sur les points d'extensions de Coloane

Nous allons utiliser des points d'extensions, comme nous l'avons expliqués précedement, pour permettre aux contributions d'offrire leur services d'importations et d'exportations. Utiliser des points d'extensions impliquent de préparer Coloane. Pour cela, il faut ajouter quelques lignes dans le fichier **plugin.xml** pour définire les points d'extensions, créer les fichiers **imports.exsd** et **imports.exsd** dans un repertoire **schema/** qui définissent la grammaire des points d'extensions, et modifier le fichier **MANIFEST.MF** pour lui spécifier les packages à exporter et qui sont sucéptibles d'être utilisés par les extensions.

Une remarque importante, Eclipse offre une intérface graphique trés agréable : PDE (Plug-ins Development Environment), qui permet la concéption et la definition de points d'extensions sont diffuclées.

4.1 plugin.xml

Dans le fichier **plugin.xml**, il faut juste définire les points d'extensions qui existent pour le plugin Coloane, ici nous avons avons décalé deux points d'extensions : *Imports* et *Exports*. En effet les contributions ne sont obligées d'implementer à la fois les fonctions d'importatios et d'exportations.

Voicie, les deux lignes ajoutées dans le fichier **plugin.xml** :

```
<extension-point id="exports" name="Exports" schema="schema/exports.exsd"/>
<extension-point id="imports" name="Imports" schema="schema/imports.exsd"/>
```

4.2 imports.exsd et exports.exsd

Dans les fichiers **imports.exsd** et **exports.exsd**, on définie la grammaire des points d'extensions. Plus précisement, le nom des attributs, et leur types. Cela permet d'indiquer comment utiliser ces points d'extensions. Dans notre cas nous avons trois attributs/

- id : C'est l'identifiant du points d'extensions.
- name : C'est le nom du nom du point d'extension
- class : C'est l'intérface que devrons implements les extensions

Noter, qu'il est trés facile, grâce au PDE (Plug-ins Development Environment), de créer ces fichiers, **imports.exsd** et **exports.exsd**. En effet, il n'y a qu'à remplire des champs, et le PDE, se charge de générer automatiquement, les fichiers attendus.

4.3 MANIFEST.MF

Dans le fichier **MANIFEST.MF**, on doit définire les packages, que l'on doit exporter pour que les extensions puissent y avoir accés, afin d'implémenter les fonctions d'importation et d'exportation.

Voicie, un extrait du fichier **MANIFEST.MF**, présentant, dans notre cas les packages exportés, qui definissent les exceptions que dervons lever les extensions, ainssi que les interfaces à implementer, et les methodes à utiliser pour créer un IModelImpl, etc...

```
Export-Package: fr.lip6.move.coloane.core.exceptions,
 fr.lip6.move.coloane.core.interfaces,
 fr.lip6.move.coloane.core.main,
 fr.lip6.move.coloane.core.motor.formalism,
 fr.lip6.move.coloane.core.ui.model
```

5 Modifications apportées à Coloane

Nous avons vus approximativement, les modifications apportées à Coloane, pour déclarer des points d'extensions. A présent, nous allons présentés, les modifications au niveaux de l'intérface utilisateur, et les classes ajoutées.

5.1 L'intérface utilisateur

5.1.1 Les items *"Imports From XXX..."* et *"Exports To XXX..."*

Comme, il sera dorénavant, possible d'ajouter différents formats pour la représentations de Réseau de Petri, il est inimaginable, qu'a chaque format XXX d'ajouter un des items *Imports From XXX...* et *Exports To XXX...*, dans le menu *File*, cela devindrait trop gros.

Nous avons, donc, optés, pour seulement deux items *Imports From...* et *Exports To...*, dans le menu *File*, qui permettent, d'ouvrire une boite de dialogue, et demandant à l'utilisateur de choisir un format, et le nom du fichier à importer, ou à exporter.

Pour cela, nous avons déclarer, dans **plugin.xml**, deux itmes, pour *Imports From...* et *Exports To....*

```
<actionSet
            id="fr.lip6.move.coloane.actionSet.file"
            label="%FILEACTIONS_ID"
            visible="true">
         <action
               class="fr.lip6.move.coloane.core.ui.actions.ExportTo"
               icon="resources/icons/export_wiz.gif"
               id="exportTo"
               label="%EXPORT_TO_ITEM"
               menubarPath="file/import.ext"
               style="push">
         </action>
         <action
               class="fr.lip6.move.coloane.core.ui.actions.ImportFrom"
               icon="resources/icons/import_wiz.gif"
               id="importFrom"
               label="%IMPORT_FROM_ITEM"
               menubarPath="file/import.ext"
               style="push">
         </action>
      </actionSet>
```

On peut voire, que nous avons définie deux classe :

```
fr.lip6.move.coloane.core.ui.actions.ExportTo
fr.lip6.move.coloane.core.ui.actions.ImportFrom
```

ces classes sérvent à definre les actions à effectuer, lorsqu'un utilisateur appuie sur l'un de ces items. Il faut noter que ces classes peuvent être considérer comme génerique car elles sont capables de créer n'importe quelles instances pour importer ou exporter un format, en utilisant, **ExportToExtension** ou **ImportFromExtension**, nous y reviendrons, plus tard.

5.1.2 Les boîtes de dialogues

La définitions des boîtes de dialogues **ImportFrom** et **ExportTo**, sont faites dans le package :

`fr.lip6.move.coloane.core.ui.dialogs`

Ces deux classes hérites de :

`org.eclipse.jface.dialogs.Dialog;`

On peut remarquer, que ces classes font appel à **ExportToExtension** ou **ImportFromExtension**, pour afficher la liste des formats disponibles, nous y reviendrons plus tard.

5.2 Les contraintes pour les extensions

Pour que Coloane, puissent exporter et importer de nouveaux format, il faut que les extensions qui implémentes, ces services, réspectent certaines régles. Ces régles sont en faites des intérfaces à implémenter. Ici, nous avons deux interfaces :

```
fr.lip6.move.coloane.core.interfaces.IExportTo
fr.lip6.move.coloane.core.interfaces.IImportFrom
```

ces intérfaces permettent à Coloane, de faire appel aux methodes (qui doivent étre commune à toutes les extensions), d'importations et d'exportations, quand l'utilisateur le demande, via les itmes correspondant.

```
public interface IExportTo {
public void export(IModelImpl modeleCourant,String filePath) throws ColoaneException;
}

public interface IImportFrom {
public IModelImpl importFrom(String filePath) throws ColoaneException;
}
```

ces methodes sont utiliser par les classes :

```
fr.lip6.move.coloane.core.ui.actions.ExportTo
fr.lip6.move.coloane.core.ui.actions.ImportFrom
```

5.3 Comment savoire quelles extensions sont présentes

L'une des difficultés, c'est de savoir quelles sont les extensions présentent, pour pouvoir afficher leur nom dans la liste déroulante des boîte de dialogues et ainssi pouvoire créé une instance pour exporter ou importer le formats voulu par l'utilisateur.

Pour cela nous avons, créé les classes suivnates :

`fr.lip6.move.coloane.core.extensions.ExportToExtension`
`fr.lip6.move.coloane.core.extensions.ImportFromExtension`

Ces classes possedent deux methodes :

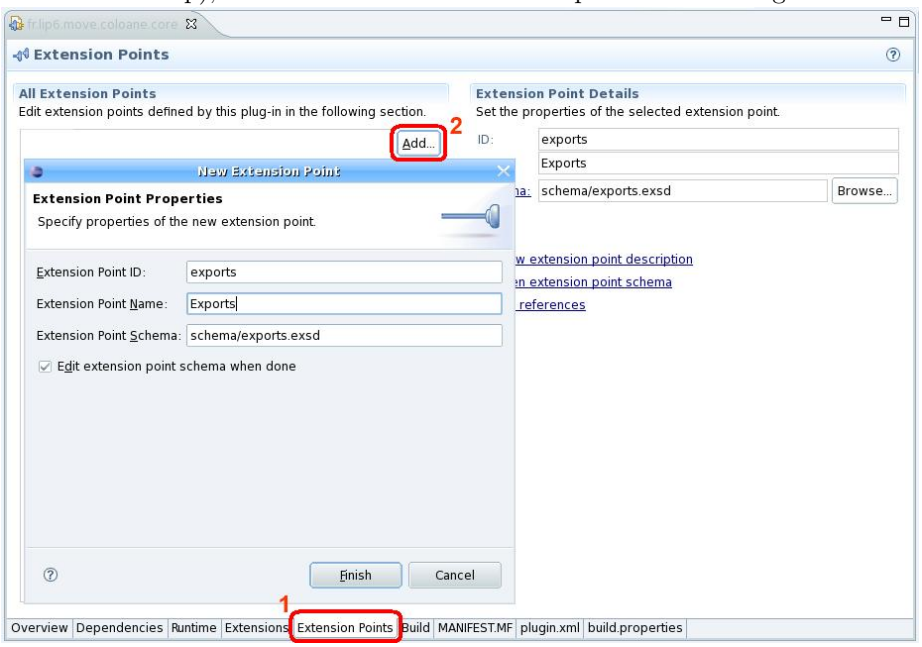
- `getAllNameExtensionConvert` : cette méthode est utiliser par les boîtes de dialogues **ImportFromDialog** et **ExportToDialog**, pour afficher la listes des noms des extensions.
- `createConvertInstance` : cette méthode est utiliser par les classes qui **ImportFrom** et **ExportTo**, pour créé une instance pour exporter ou importer un format.

6 Concéption de points d'extensions

Nous allons, vous présenter, comment créer un point d'extension, pour permttre à Coloane, d'intégrer via des contributions de nouveaux services d'exportations. Pour cela, nous allons utiliser le PDE (Plug–ins Developement Environment) présent dans Eclipse.

6.1 Déclaration du point d'extension

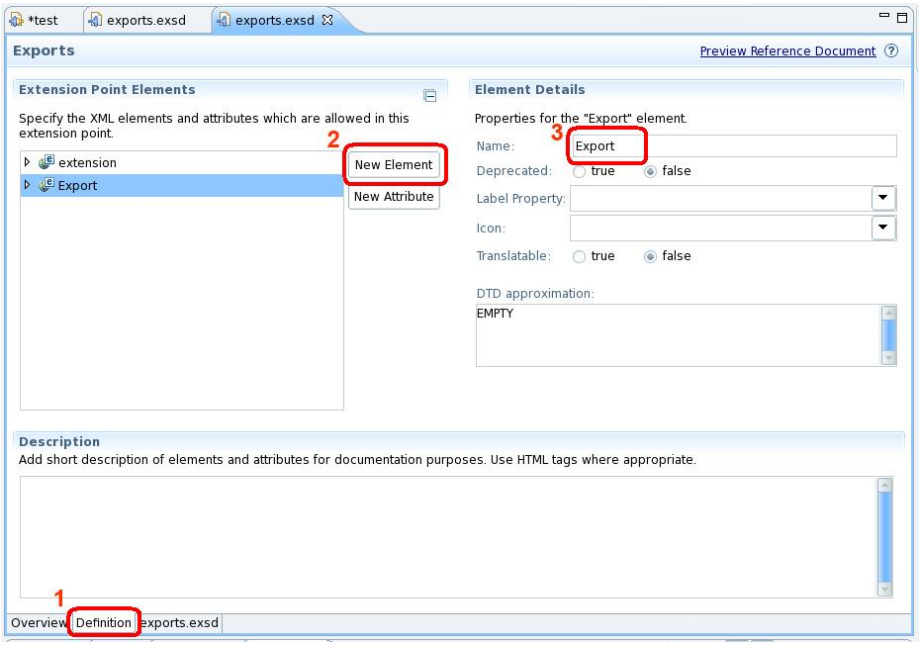
L'onglet 'Extension points' de l'éditeur de fichiers manifestes aide à la création de nouveaux points d'extension. Ouvrir cet onglet pour le plugin 'fr.lip6.move.coloane.core'. Le bouton 'Add...' ouvre un assistant permettant d'indiquer l'ID du nouveau point d'extension (l'ID réel sera la concaténation de l'ID du plugin et de la valeur de ce champ), son nom et le nom du fichier qui contiendra la grammaire :



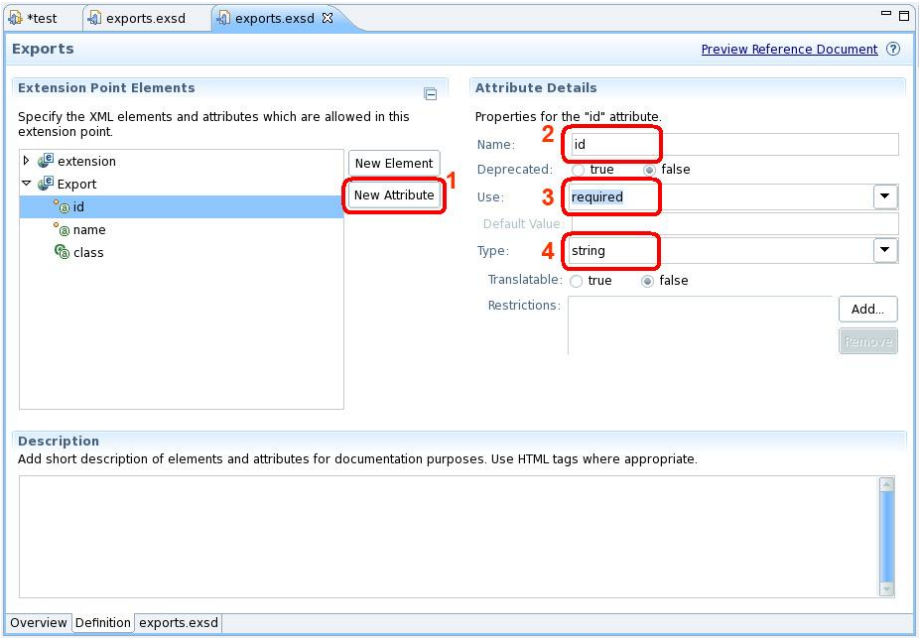
6.2 Définition de la grammaire

Notre point d'extension doit permettre à des plugins de contribuer de nouveaux types de formats d'exportations. Nous proposons donc que la grammaire soit composée d'un élément XML nommé 'Exports' comprenant trois attributs 'id', 'nom' et 'classe'.

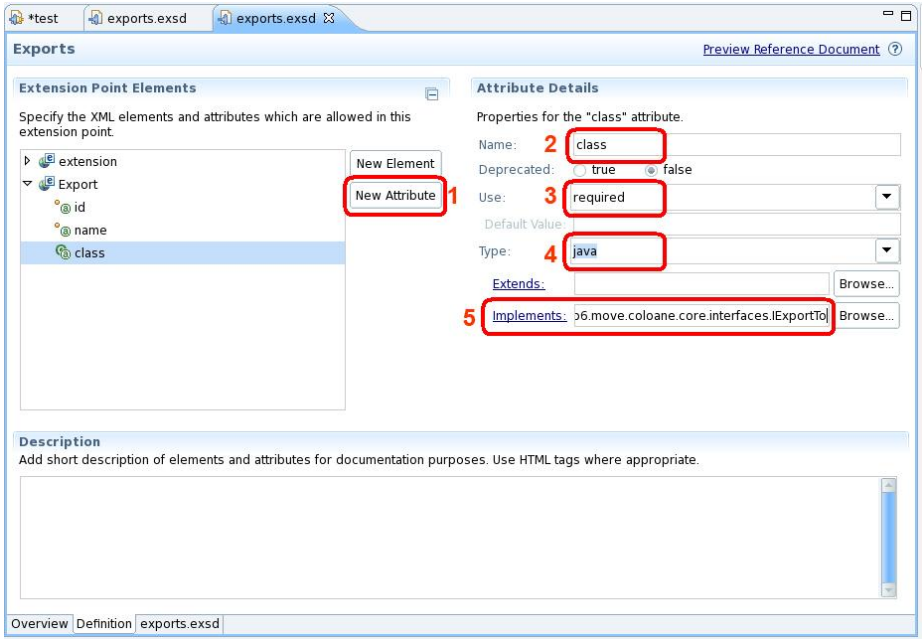
Cette grammaire doit être définie dans un fichier XMLSchema avec une extension .exsd. Le fichier a été créé automatiquement lors de l'étape précédente et l'éditeur de fichiers .exsd a été ouvert sur notre fichier. L'éditeur dispose de plusieurs onglets, se placer dans l'onglet 'Definition' et utiliser le bouton 'Add Element' pour créer un élément XML que nous nommerons 'Exports' :



Utiliser le bouton 'New Attribute' pour créer les attributs 'id' et 'nom'. Indiquer que ces attributs sont 'required' et de type 'string' :



Créer ensuite l'attribut 'classe'. Spécifier qu'il est 'required' et de type 'java'. Cet attribut sera utilisé par les plugins contributeurs pour indiquer le nom de la classe qui gère l'exportation d'un format. Pour que notre plugin (celui définissant le point d'extension) puisse manipuler les classes fournies par les plugins contributeurs, nous allons leur imposer une interface commune. Le nom de cette interface est à indiquer dans le champ 'Implements' :



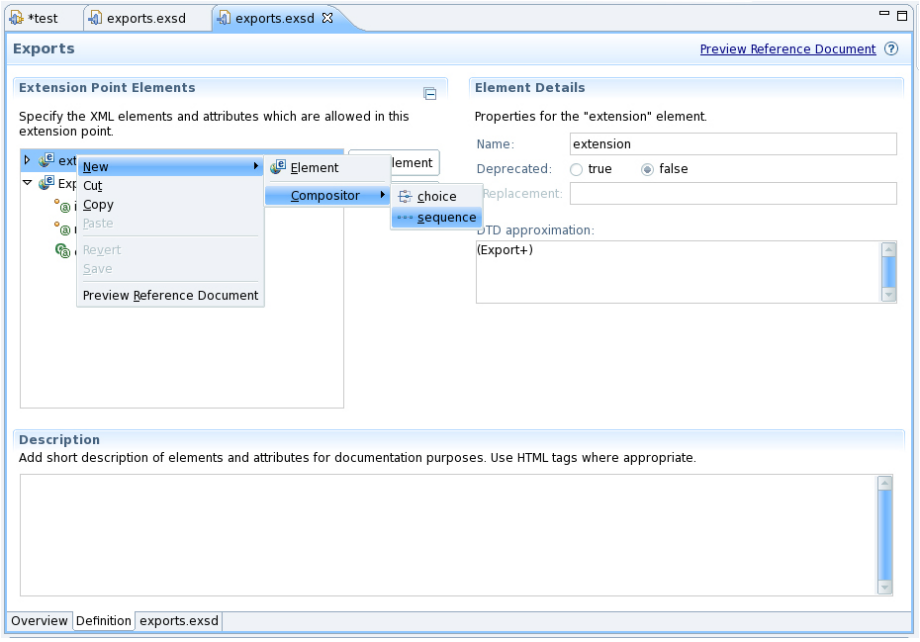
Ajouter l'interface dans le plugin 'fr.lip6.move.coloane.core.interfaces', son code est le suivant

```
package fr.lip6.move.coloane.core.interfaces;

import fr.lip6.move.coloane.core.exceptions.ColoaneException;
import fr.lip6.move.coloane.core.ui.model.IModelImpl;

public interface IExportTo {
        public void export(IModelImpl modeleCourant, String filePath)
                throws ColoaneException;
}
```

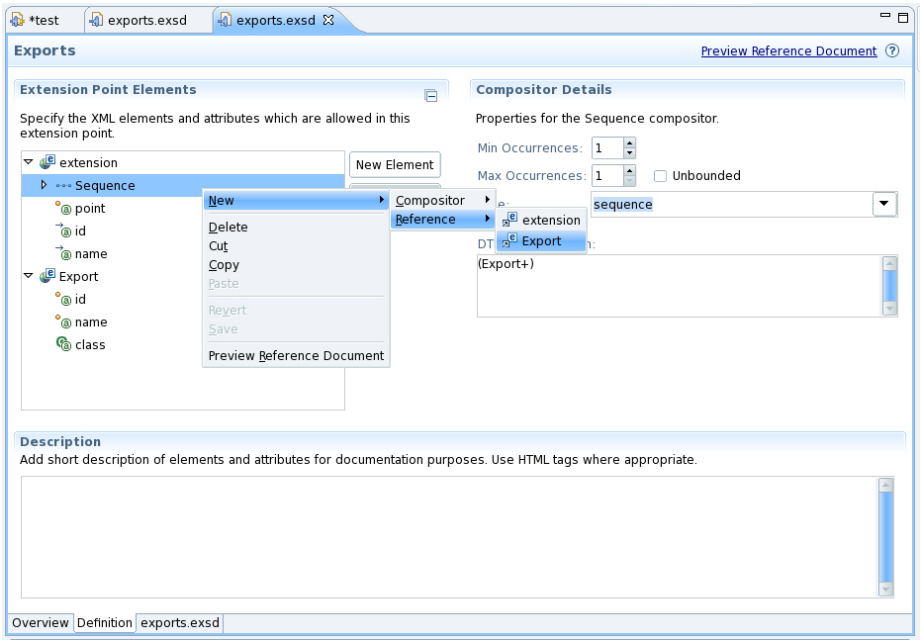
Cette interface devra être utilisable par les plugins dépendants : dans l'éditeur de fichier manifestes, sélectionner l'onglet 'Runtime' et ajouter le package

'fr.lip6.move.coloane.core.interfaces' dans la section 'Exported Packages'.

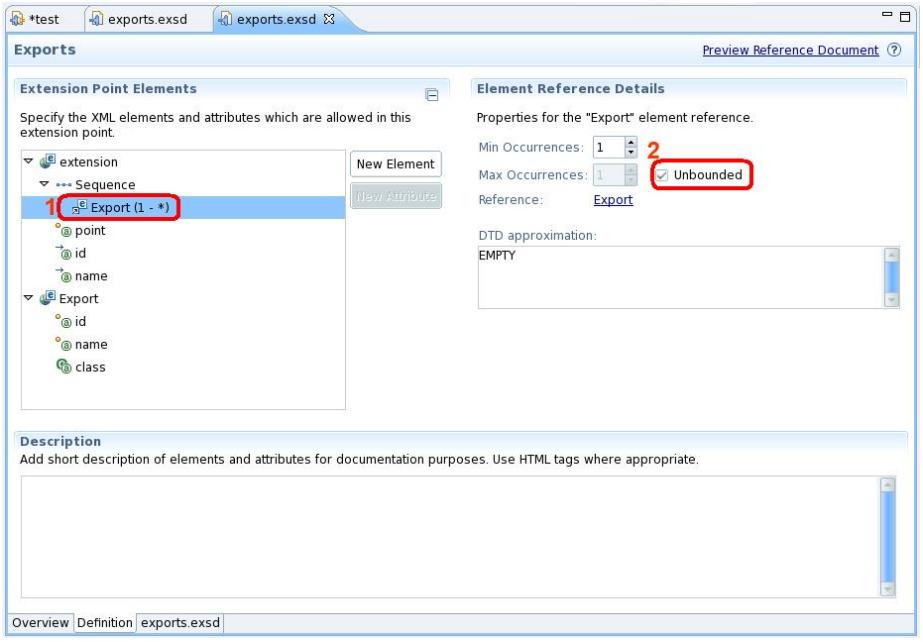
La grammaire d'un point d'extension se compose systématiquement d'un élément parent nommé 'extension'. Il nous faut indiquer la relation entre l'élément 'extension' et notre élément 'Export'. Dans notre cas l'élément 'extension' peut contenir plusieurs sous-éléments 'Export' (un plugin peut fournir plusieurs types de formats d'exportations), pour définir ce lien il faut utiliser le menu contextuel sur l'élément extension et sélectionner 'New - Compositor - Sequence' :



Ouvrir ensuite le menu contextuel sur la séquence et sélectionner 'New - Reference - Export' :



Sélectionner la référence vers l'élement 'Export' et cocher la case 'Unbounded' dans la partie de gauche :

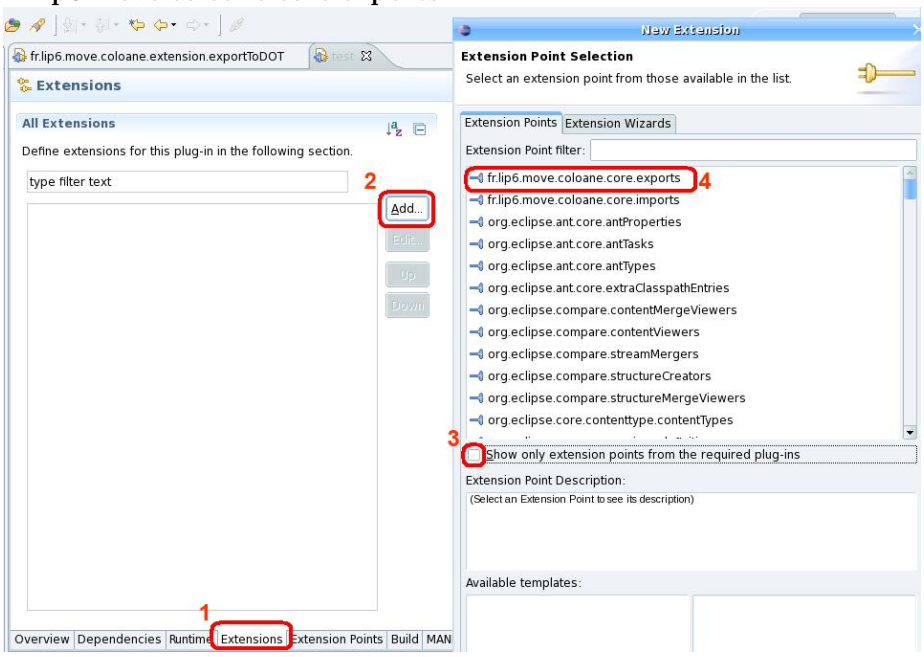


7 Réalisation d'une extension

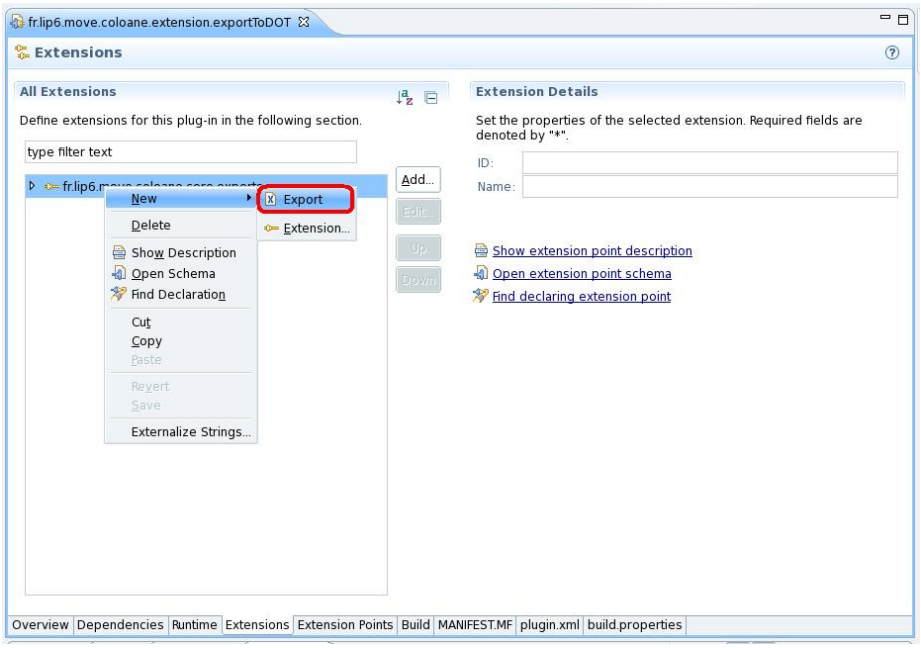
Pour réaliser des extensions pour Coloane offrant de nouveaux services d'imporations ou d'exportations, il suffit d'utilisé, les points d'extensions que nous avons définie, précedement.Ici, aussi grâce au, PDE, nous pouvons créer facilement des extensions. Nous allons vous présenter la définition d'un extension permettant d'exporter au format DOT, en utilsant le PDE.

7.1 Définition de l'extension exportToDOT

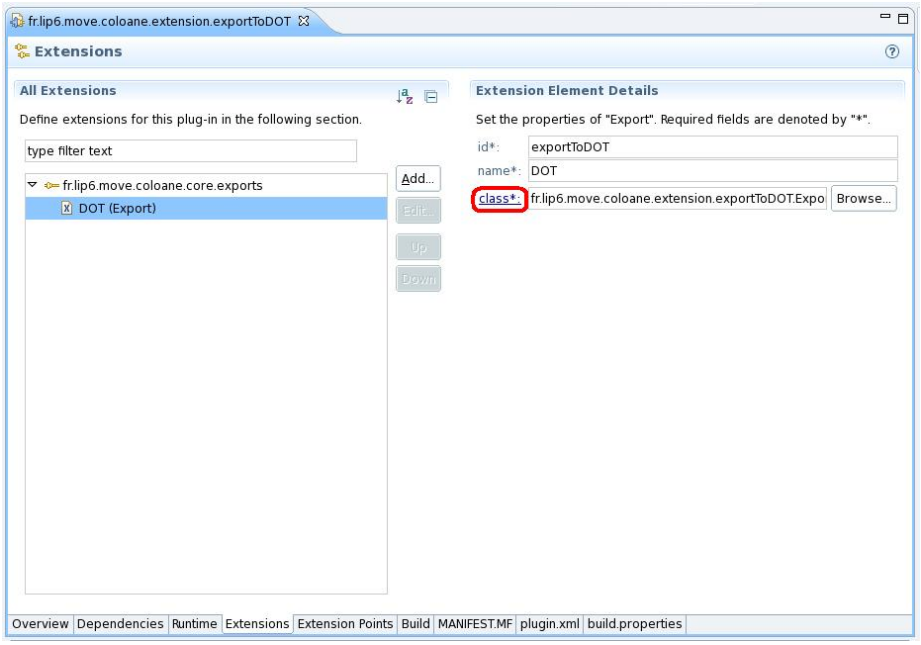
Dans l'éditeur de fichiers manifestes de ce plugin, sélectionner l'onglet 'Extensions'. Dans cet onglet cliquer sur le bouton 'Add...' : la liste de tous les points d'extension est présentée. Dans l'assistant décocher la case 'Show only extension points from required plugins' et sélectionner le point d'extension **fr.lip6.move.coloane.core.exports** :



L'objectif de l'onglet 'Extensions' est de générer la définition XML de l'extension dans le fichier plugin.xml. Sélectionner le point d'extension et utiliser le menu contextuel pour ajouter un des éléments XML déclarés dans la grammaire associée au point d'extension, dans notre cas un élément de type 'Exports' :



La partie de gauche de l'éditeur permet de voir les attributs associés à cet élément XML et d'indiquer leurs valeurs :



La définition de notre extension est terminée, le code XML correspondant a été généré dans le fichier plugin.xml :

```
<extension
         point="fr.lip6.move.coloane.core.exports">
      <Export
            class="fr.lip6.move.coloane.extension.exportToDOT.ExportToImpl"
            id="exportToDOT"
            name="DOT">
      </Export>
</extension>
```

7.2 Classe associée à l'extension

L'étape suivante consiste à créer la classe indiquée par l'attribut **class**. Le PDE fournit une aide sympathique : le libellé de l'attribut ' **class** est un lien hypertexte capable d'ouvrir l'assistant de création de classe et de pré-remplir les champs (notamment le nom de l'interface à implémenter et/ou de la classe à étendre) :

Faire créer la classes 'ExportToImpl' en sélectionnant le lien hypertexte 'classe*'. Et la compléter pour implementer la fonction export :

```
package fr.lip6.move.coloane.extension.exportToDOT;

import java.util.Vector;
import fr.lip6.move.coloane.core.exceptions.ColoaneException;
import fr.lip6.move.coloane.core.interfaces.IExportTo;
import fr.lip6.move.coloane.core.ui.model.IModelImpl;

public class ExportToImpl implements IExportTo {

        public ExportToImpl() {

        }

        public void export(IModelImpl modeleCourant, String filePath)
                        throws ColoaneException {
                DotTranslator translator = new DotTranslator();
                Vector<String> model = translator.translateModel(modeleCourant.g
                translator.saveModel(model, filePath);
        }

}
```

8 Bibliographie

Livre :
- **"Eclipse : Principes, patterns et plug-in"**, par Erich Gamma, Kent Beck, et Olivier Engler (Broché - 13 septembre 2006)
- **"Eclipse 3 pour les développeurs Java : Développez des plug-in et des applications"** par Berthold Daum et Claude Raimond (Broché - 1 janvier 2005)

Liens :
- `http://http://www.eclipsetotale.com`, site trés bien fait qui fortement inspirer les parties 6 et 7.
- `http://www.developpez.com`, contenant plein d'informations sur SWT.
- `http://www.graphviz.org`, contenant plein d'informations sur DOT.