UNIVERSITE DE NANTES – MASTER 1 ALMA

IR - COLOSS

Compte-rendu et avancement du projet

BEGAUDEAU Stéphane et BRUN Clémentine

Compte-rendu sur l'avancement du projet d'IR de M1 ALMA au sein de l'équipe du COLOSS.

BEGAUDEAU Stéphane et BRUN Clémentine – M1 ALMA

Sommaire

Introduction

Les applications actuellement développées sont de tailles de plus en plus importantes. La modélisation UML permet de représenter le comportement d'un système placé au niveau des classes. Aujourd'hui, la capacité d'abstraction des modélisations n'est plus suffisante. C'est ainsi que la programmation par composants est apparue.

Cette méthode de développement amène avec elle de nombreuses nouvelles interrogations, de nouveau standard à définir. L'équipe du COLOSS travaille à apporter des réponses à certaines d'entre elles.

I. Environnement de travail

1. Principe du projet

Nous sommes deux étudiants de master 1 ALMA de l'Université de Nantes : Stéphane Begaudeau et Clémentine Brun. Dans ce cursus nous avons un module d'initiation à la recherche ayant pour but de nous faire intégrer un d'une équipe de recherche. Il y a deux aspects essentiels de l'enseignement : la découverte d'une équipe de recherche et la participation à un projet.

Pour la bonne réalisation de ces objectifs, la journée du jeudi est entièrement consacrée à ce module. Les évènements actuels au sein de l'Université de Nantes ne nous permettent pas toujours de travailler le jeudi, mais nous conservons tout de même le principe d'une journée entière par semaine dédiée à ce travail.

2. L'équipe du *COLOSS*

Le *COLOSS*[1] est une des onze équipes de recherche du laboratoire du *LINA*[2]. Ce dernier est spécialisé en « sciences et technologie du logiciel » et ses axes de recherches sont les architectures logicielles distribuées et les systèmes d'aide à la décision.

L'équipe *COLOSS* est dédiée à la création de méthodes et outils à destination des développeurs de logiciels. Ces outils permettront d'assurer notamment la correction des composants de systèmes informatiques complexes.

Au sein de cette équipe nous serons encadrés par Gilles Ardourel (Responsable adjoint de l'équipe *COLOSS*) et Pascal André. Tous les deux sont enseignants chercheurs (?) à l'UFR sciences de l'Université de Nantes. Ils font également partie de l'équipe *COLOSS*.

3. Le projet *ECONET*

Parmi les différents travaux de l'équipe *COLOSS*, nous participerons au projet *ECONET*. Il n'est pas l'exclusivité de l'équipe *COLOSS*, mais résulte de partenariats internationaux. Nous pouvons citer les équipes *DSRG* (République Tchèque), *LCI* (Roumanie) et *OBASCO* (France - EMN[3]).

L'objectif d'ECONET est de créer des liens entre le code java, les différents types de modèle, et le méta modèle.

4. Le process B

Le projet ECONET est découpé en trois sous projets :

- ✓ Métamodels : à partir d'un des deux types de modèles, pouvoir retrouver le méta modèle.
- ✓ Process A : à partir du modèle abstrait structurel, pouvoir retrouver le modèle abstrait de comportement.
- ✓ Process B : à partir de modèle à composant (EJB, …) et du code java, pouvoir retrouver le modèle abstrait structurel ainsi que le code java annoté. C'est le sous projet auquel nous avons été affectés.

[1] COmposants et LOgiciels SurS.
[2] Laboratoire d'Informatique de Nantes Atlantique.
[3] Ecole des Mies de Nantes

II. Quelques notions essentielles

La première phase de notre « stage » fut l'apprentissage d'un certain nombre de concepts qui nous étaient totalement nouveaux. Afin d'avoir de solides bases, nous avons donc réservé trois semaines à cet effet. Cela nous permet par la suite de progresser plus rapidement.

1. Modèles et composants

Les applications d'aujourd'hui sont devenues de taille tellement conséquente, que le niveau d'abstraction proposé par la modélisation UML n'est plus suffisante. Les développeurs ont donc à leur disposition la représentation d'un système par un modèle, constitué de composants.

a. Les modèles

Initialement les modèles avaient une vision entité-relation d'un système, à l'image d'UML par exemple. Désormais les modèles permettent également d'obtenir une représentation simplifiée et abstraite de la structure et du comportement d'un système.

Il existe plusieurs types de modèle de composants utilisés dans l'industrie : EJB[4], .NET[5], CCM[6]. Ils offrent alors des cadres de création de composants.

b. Les composants

La programmation orientée composant permet de développer des logiciels extrêmement modulables et facilite donc la maintenance des systèmes.

Un composant est une entité possédant des fonctionnalités accessibles depuis l'extérieur. On dit qu'il a des *services fournit* et que les programmes les utilisant sont des *clients*. Un composant peut également avoir besoin de fonctionnalités qu'il ne possède pas, on appelle cela les *services requis*. Les communications avec le monde extérieur peuvent se faire au travers d'*interfaces*. Les points de connexion sont appelés des *ports*.

Les composants sont réutilisables, ce qui permet de diminuer les couts de production d'un logiciel. Ils peuvent être eux-mêmes composables, c'est-à-dire qu'ils peuvent contenir d'autres composants. Ils sont également auto descriptif, ils possèdent en effet des mécanismes d'introspection.

Voici une représentation couramment utilisée dans les diagrammes de composants :

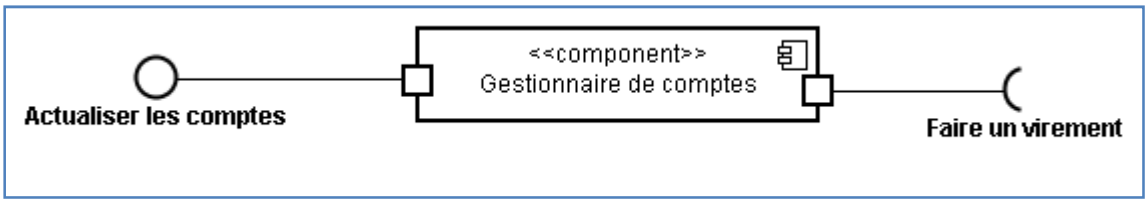

Figure 1 : Représentation d'un composant

Les cercles représentent les interfaces des services requis, tandis que les demi-cercles symbolisent les interfaces des services offerts. Les carrés à cheval représentent les ports.

[4] Enterprise Java Bean. Ce modèle est spécialement dédié au monde Java et à la technologie J2EE.
[5] Ensemble de produits développé par Microsoft, notamment basé sur les composants.
[6] Corba Component Model.

2. Architecture à plugins

La définition d'un plugin donnée par Wikipedia est « un plugin est un programme informatique qui interagit avec un logiciel principal pour lui apporter de nouvelles fonctionnalités ». Les plugins peuvent être utilisés pour constituer les bases d'une application. On parle alors d'architecture à plugins. L'exemple le plus connu de ce type d'architecture est la plate forme Eclipse RCP[7].

Un plugin sera dépendant d'autres, il faut alors un définir leur mode de communication. Ce sont les points d'extensions qui sont chargés de ce rôle.

Les propriétés d'un plugin sont principalement définies dans deux fichiers.

a. META-INF/MANIFEST.MF

C'est le manifest qui contient diverses informations concernant le plugin:

- ✓ Son nom : `Bundle-name MonPlugin`
- ✓ Son identifiant : `Bundle-SymbolicName MonPluginId`
- ✓ Les plugins requis : `Require-Bundle org.eclipse.search`
- ✓ …

b. Plugin.xml

Plugin.xml est le fichier permettant de décrire le plugin, notamment ses points d'extensions. Voici un exemple de ce fichier :

```
<?xml verison="1.0" encoding="UTF-8"?>
    <plugin>
        <extension point="org.eclipse.ui.ActionSets">
            <actionSet  id = "MonMenu"
                        label = "Cliquez ici"
                        visible = "true">
                <menu   id = "MonMenu.Menu1"
                        label = "Mon&ampMenu"
                    <action class = "MonMenu.maClasseAction"
                            label = "SousMenu1"/>
                </menu>
            </actionSet>
        </extension>
    </plugin>
</xml>
```

Il décrit un plugin ajoutant un menu à Eclipse, appelé `MonMenu`, et ayant un sous menu `SousMenu1`.

[7] Rich Client Application : Client proposant la mise à disposition d'un framework de développement et de composants de base, dans le but de faciliter le travail des développeurs.

III. Etude du projet

Comme nous l'avons dit précédemment, nous travaillons sur le sous projet Process B, du projet ECONET de l'équipe COLOSS. Nous allons décrire ici les différents éléments du projet, ses problématiques et les solutions que nous comptons apporter.

1. La boite à outils (« toolbox »)

La base du process B est une boite à outils contenant toutes les opérations nécessaires à la retro-ingénierie de code java en modèle abstrait structurel. Voici les principaux d'entre eux :

- ✓ Model from annotation : cet outil nécessite en entrée le code java déjà annoté. Il permet ensuite de générer le modèle.
- ✓ Annotation writer from modèle : c'est l'inverse de l'outil précédent. Celui-ci annote le code Java à partir du modèle.
- ✓ Clustering tool : regroupe les classes dans différents composants, ainsi que les composites dans les composants.
- ✓ Distribution analyser : étudie les programme distribués afin de détecter les composants.
- ✓ Model transformation
- ✓ …

Afin de tirer le maximum d'informations d'un programme Java, il sera nécessaire d'itérer plusieurs fois sur des outils différents.

Chacun de ces outils est, ou sera, un plugin connecté à un plugin central, que l'on peut considérer comme la boite à outils. C'est cette dernière que nous devons développer.

2. Les problématiques

Il y a plusieurs aspects du projet que nous devons préciser ou définir avant de commencer son développement.

a. L'interface utilisateur

Comment représenter les itérations sur les outils que l'utilisateur désire ?

Nous choisissons la solution du fichier texte. En effet, tous les plugins de la boite à outils ne sont pas encore implémentés. Certains nécessiterons peut être des options, des informations complémentaires. Nous voulions laisser la possibilité d'étendre facilement la description des itérations. Un fichier texte accompagné d'un parseur nous a semblé la solution la plus satisfaisante en étant la plus simple à réaliser.

L'utilisateur défini une séquence d'actions à réaliser avant de lancer l'exécution. Il peut alors revenir sur ces choix et modifier la séquence précédemment écrite. Deux possibilités apparaissent alors :

- ✓ L'utilisateur ajoute simplement de nouvelles itérations. Le programme ne doit pas relancer toute l'exécution, mais simplement la poursuivre à partir de la dernière étape déjà réalisée.
- ✓ L'utilisateur modifie ou supprime des itérations déjà écrite. Le programme ne peut alors pas repartir de la dernière étape, nous choisissons alors de ré exécuter l'ensemble du programme.

b. *Les sauvegardes*

L'utilisateur défini une séquence d'exécution. L'étape n utilise donc le résultat de l'étape n – 1. Comment sauvegarder ces différentes étapes ?

Nous n'avons pas encore répondu à cette question. Nous avons pour le moment simplement quelques pistes de réflexion :

- ✓ Une solution extrême : la sauvegarde complète à chaque étape. C'est une solution très couteuse en espace mémoire et donc non satisfaisante.
- ✓ Une autre solution extrême : la sauvegarde uniquement de l'étape n – 1. L'utilisateur peut vouloir très certainement regarder le résultat de chaque étape, afin de corriger ou d'améliorer sa séquence d'exécution suivante. Cette solution n'est donc pas satisfaisante non plus.
- ✓ Une solution intermédiaire : utiliser le principe d'un outil de gestion de version, type SVN. On ne sauvegarde que les modifications réalisées par l'étape. Cette solution mixte parait satisfaisante. Nous n'avons pour le moment pas de solution quand à sa mise en œuvre.

Bien entendu, il faut dans tous les cas conserver le projet initial.

c. *La détection de dépendances*

Certains plugins de la boite à outils sont dépendants les uns des autres. Nous pouvons par exemple avoir des contraintes du type : Pour exécuter le plugin 3, il faut que le plugin 2 ait été lancé.

Pour le moment, nous avons simplement évoqué le principe d'un parseur vérifiant la cohérence de la totalité de la séquence entrée pare l'utilisateur.

d. *L'ajout de nouveau plugin*

Cette fonctionnalité ne s'adressera qu'à un public de développeurs averti. Nous pouvons donc nous contenter d'une ergonomie relativement sommaire. Nous avons pour le moment uniquement imaginé une sorte de fichier XML, sans plus approfondir.

IV. La maquette de la boite à outils

Afin de faire valider dans un premier temps l'interface de notre projet, nous avons réalisé une maquette. Celle-ci nous a permis de tester la mise en œuvre des plugins Eclipse, tout en commençant à répondre à certaines problématiques exposées précédemment.

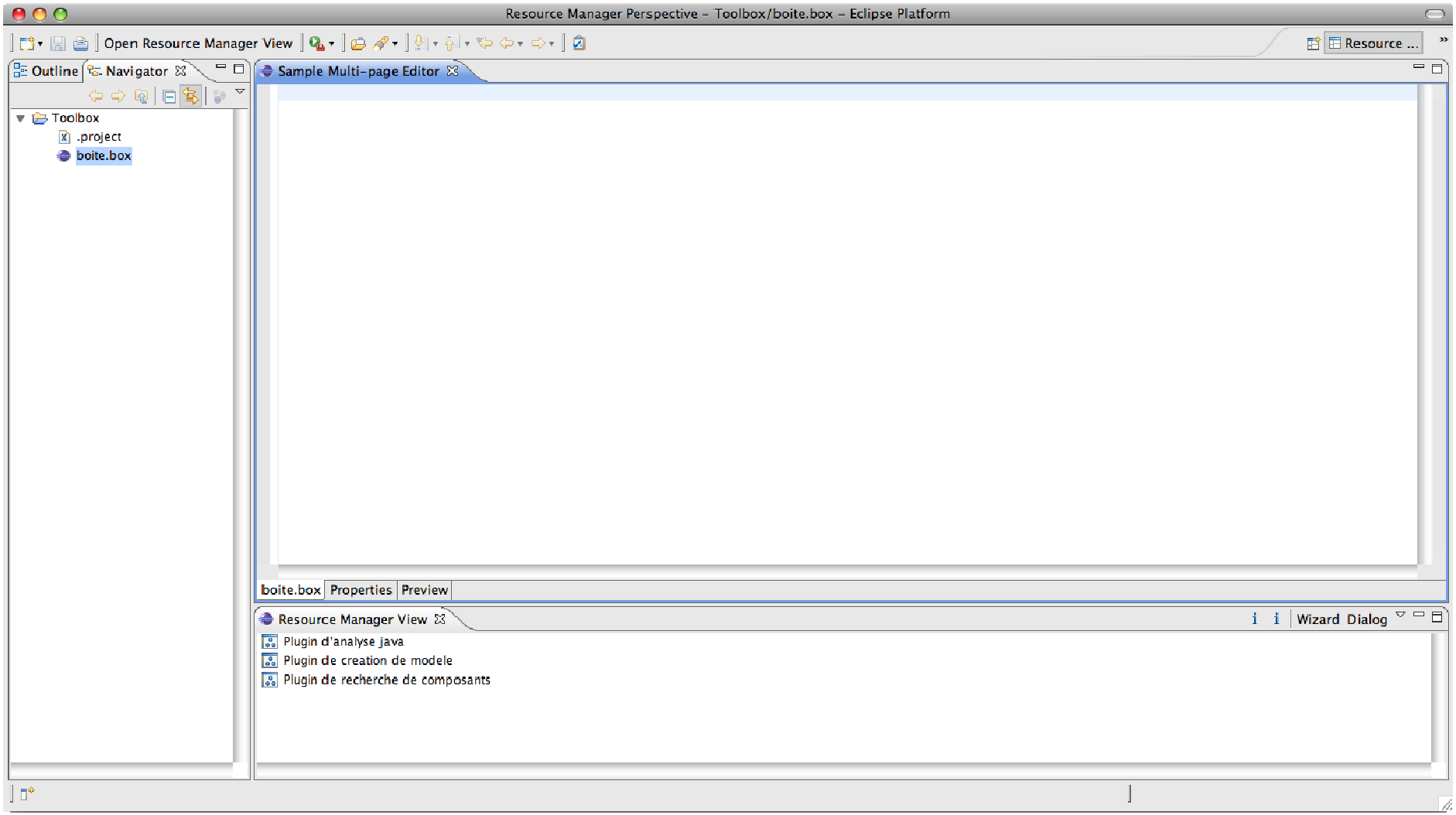

Figure 2 : Interface initiale. Un projet toolbox a été créé, dans lequel il y a un nouveau fichier nommé boite.box. Cela déclenche l'ouverture de l'éditeur de texte.

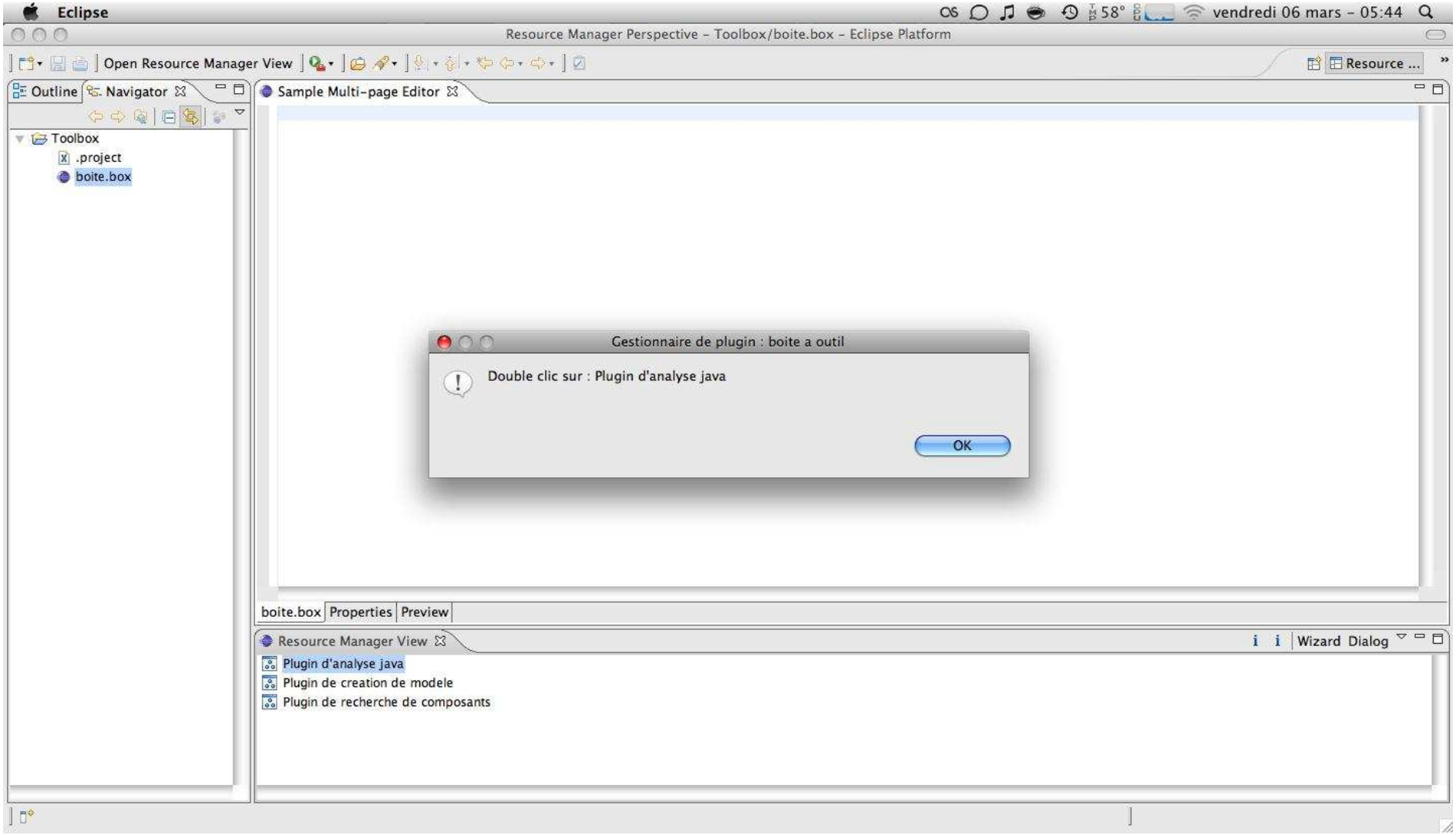

Figure 3 : Ouverture d'une pop-up affichant les options d'un plugin grâce à un double-clic.

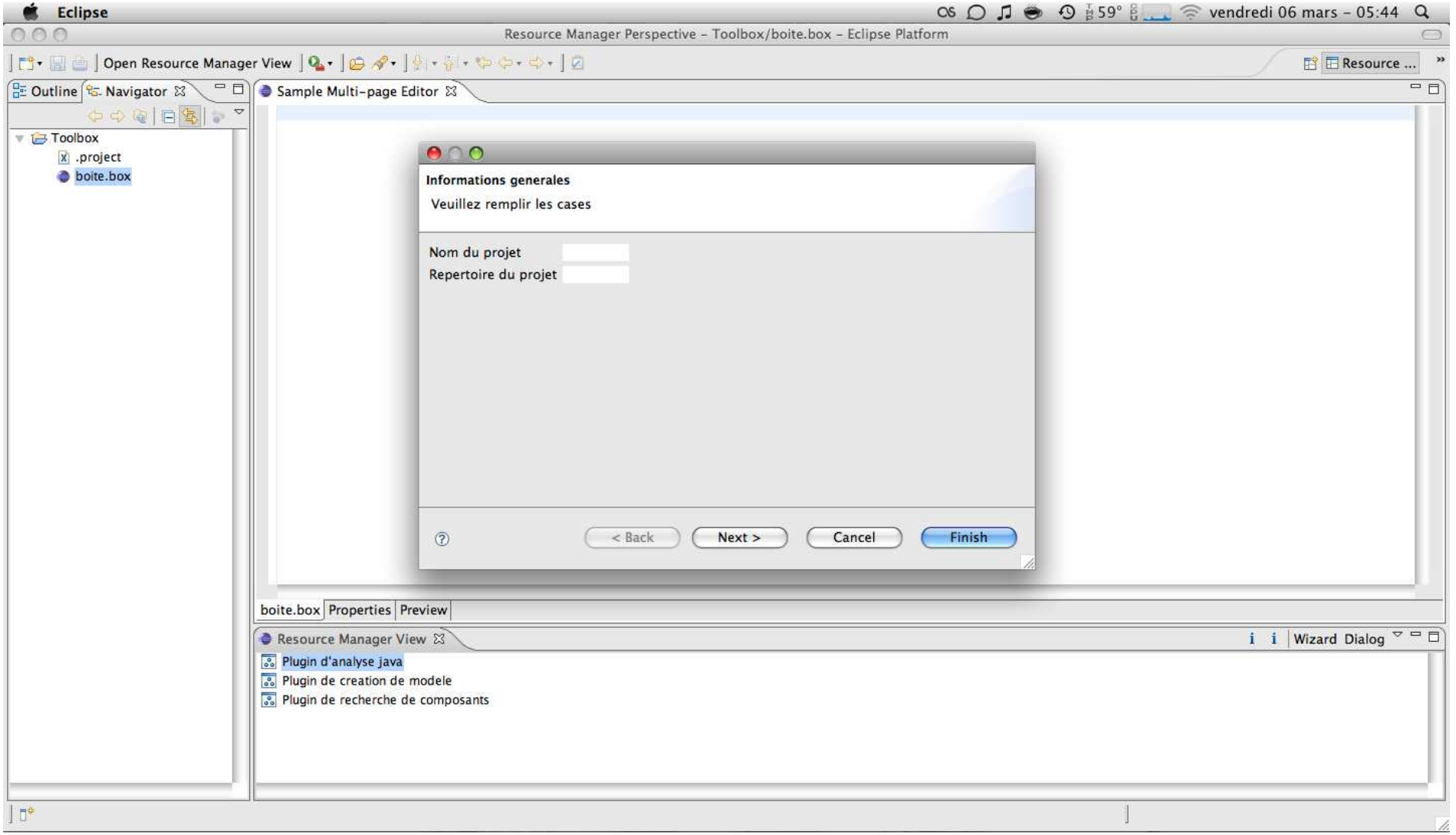

Figure 4 : La future boite de création d'un projet (très simplifiée pour le moment) page 1

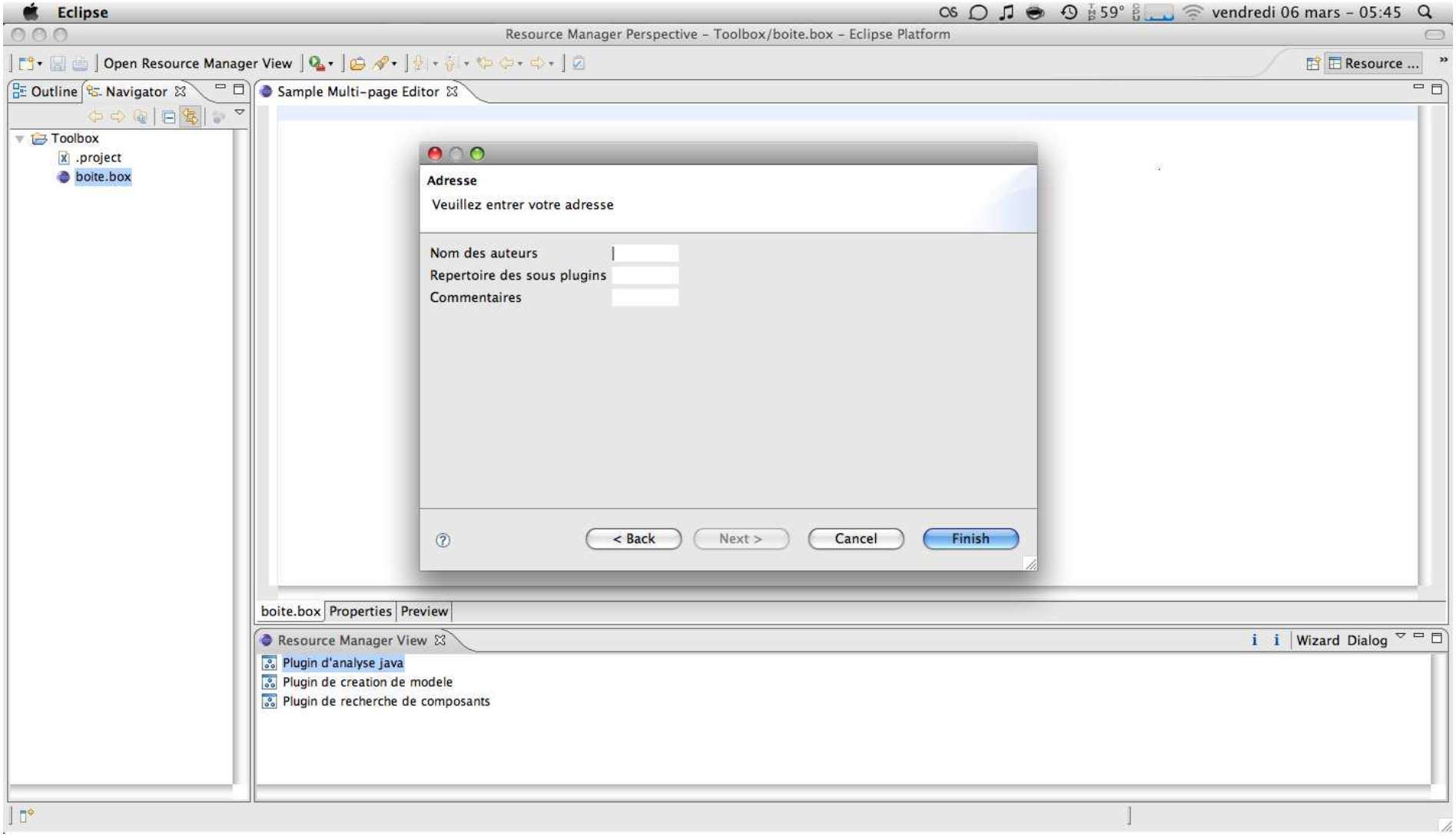

Figure 5 : la future boite de création d'un projet (très simplifiée pour le moment) page 2

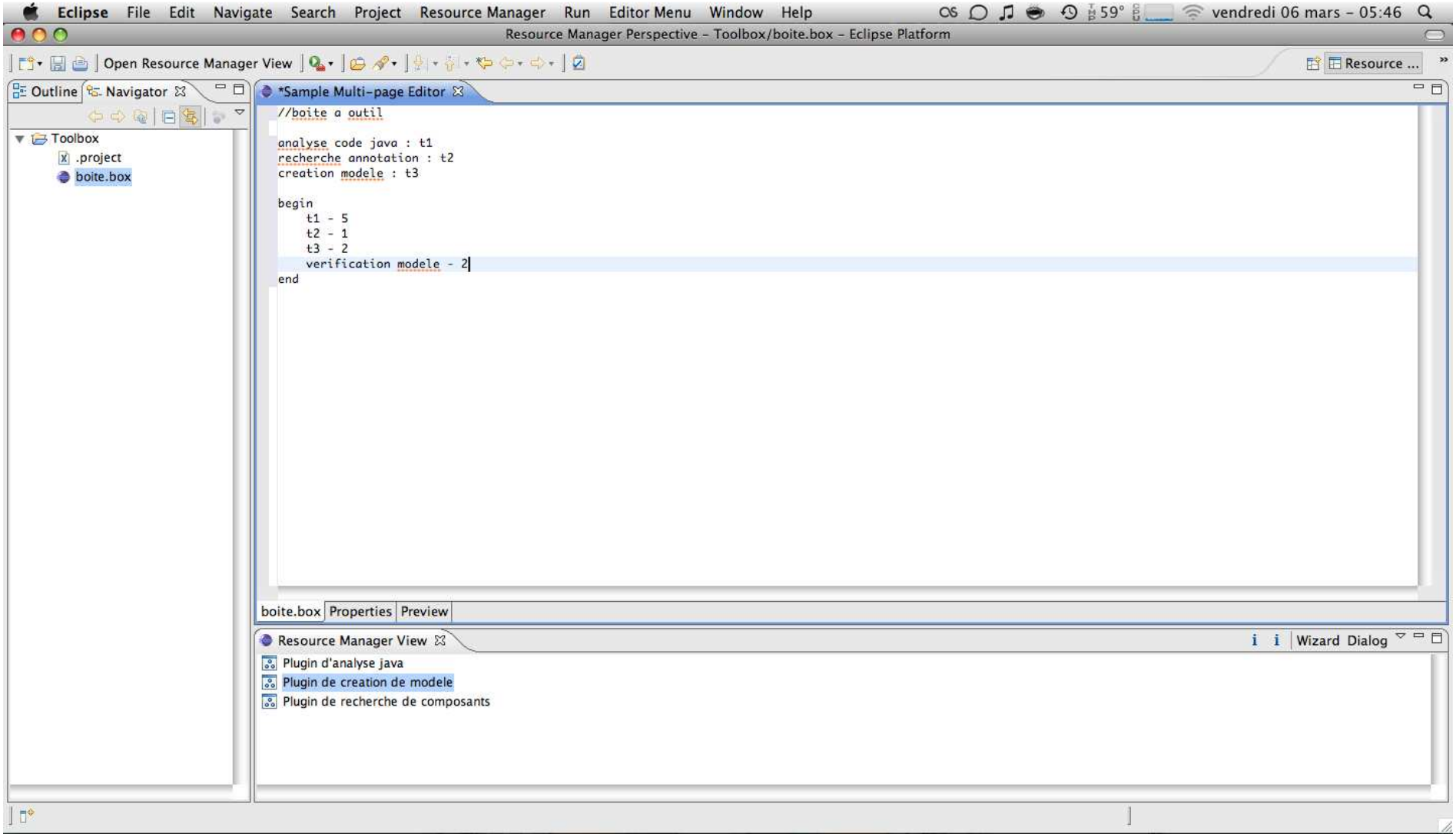

Figure 6 : Exemple de séquence comme l'utilisateur pourra la définir. La première ligne est un commentaire, les trois suivantes servent au renommage des plugins, la suite est la séquence d'exécution.

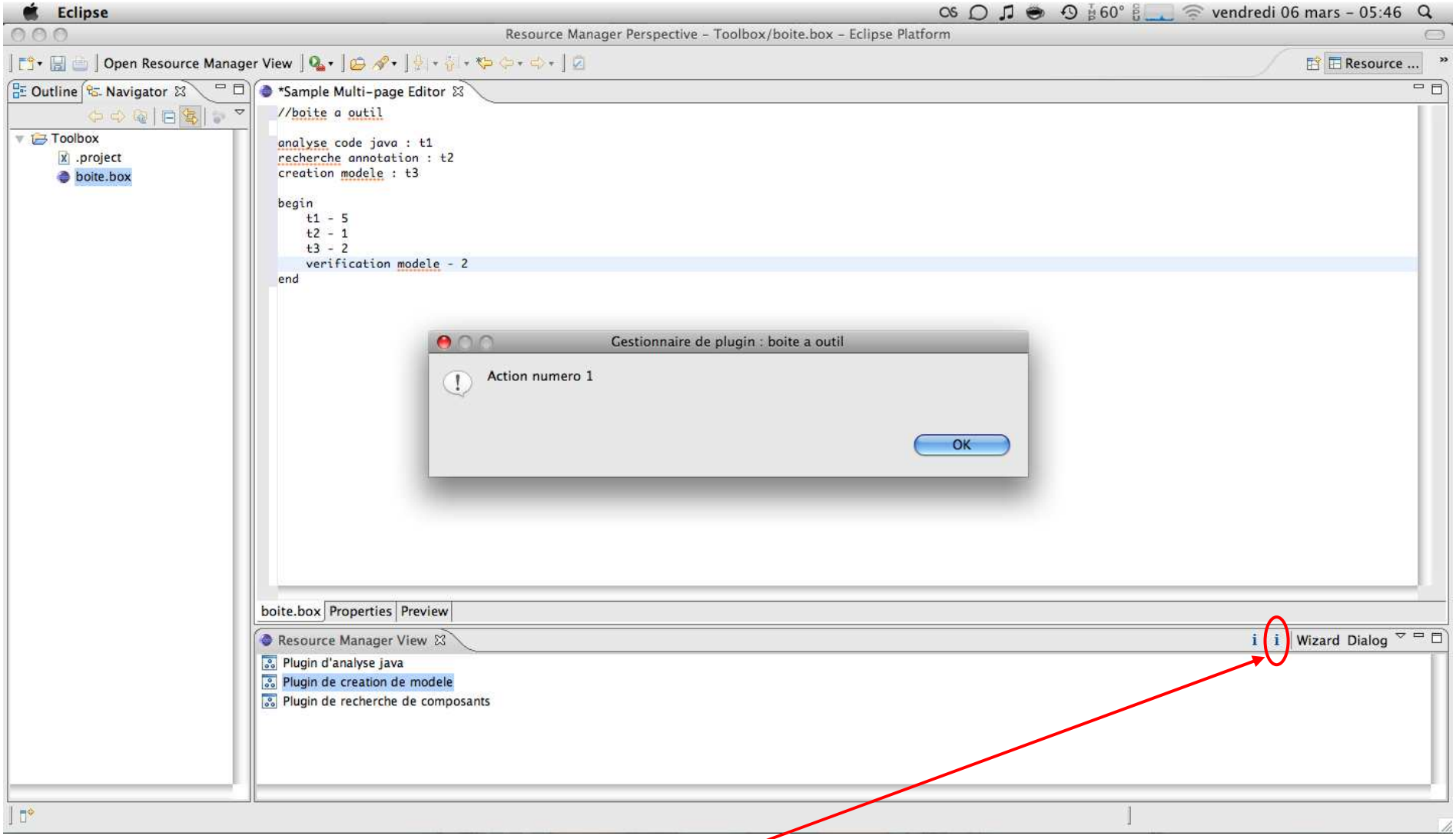

Figure 7 : Un exemple d'action déclenché par le clic sur une icône. Ici un clic sur le « i » bleu.

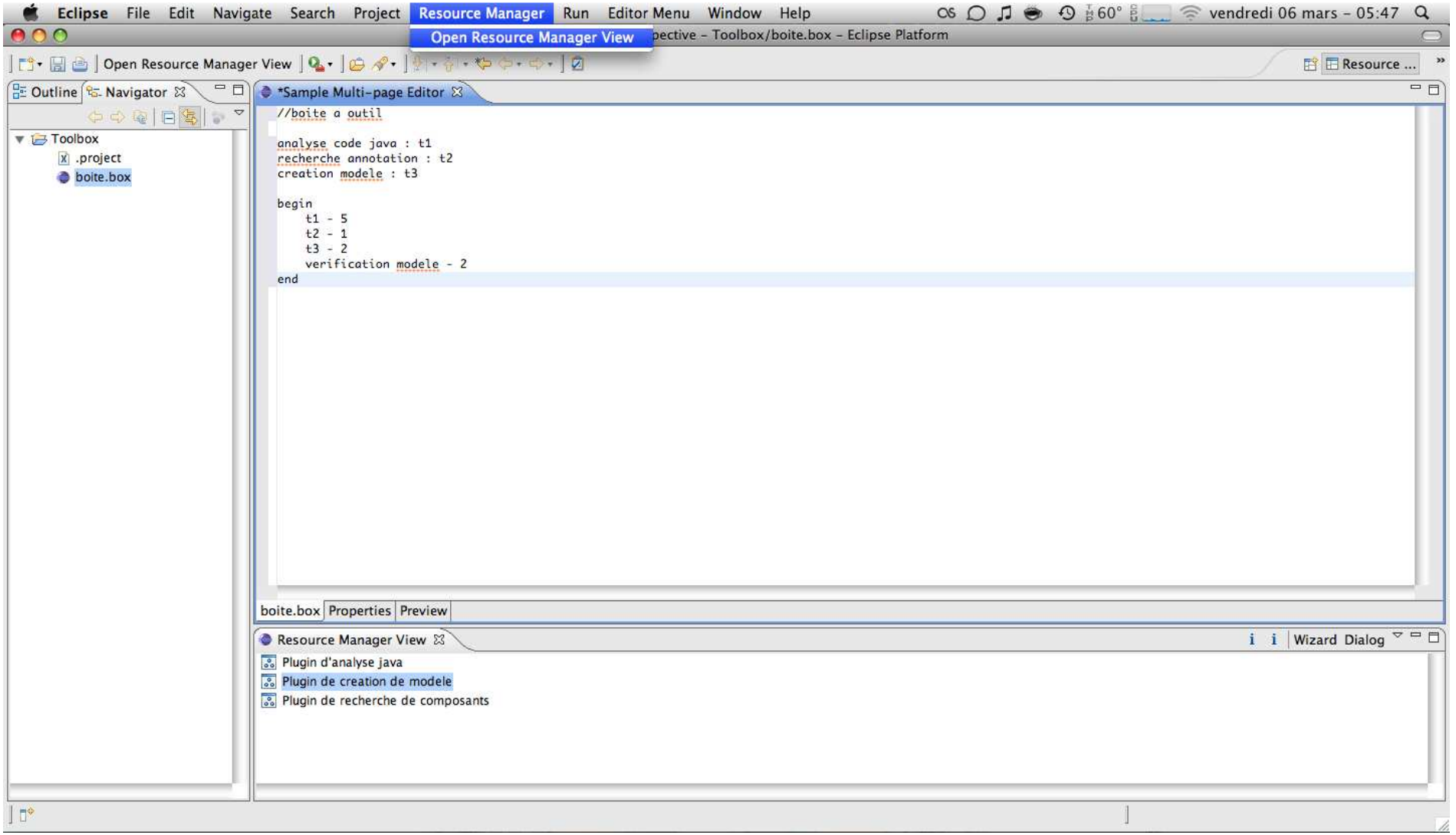

Figure 8 : Menu permettant de déclencher la modification de la perspective actuelle pour y afficher le menu du bas avec les plugins.

Ressources

http://fr.wikipedia.org/

http://www.pajbam.com/wp-content/uploads/2008/04/cours_plugin_eclipse.pdf

Ingénierie des composants : Concepts, techniques et outils, Oussalah, Mourad, Collectif, Editions Vuibert, Collection Génie Logiciel, 2005, ISBN 2-7117-4836-7