|  |
| --- |
| Auteur : Wei JIANG  6 mars 2015 |

|  |
| --- |
| **Rapport technique**  **Edition intelligente de programme Java pour diatomée** |
| **Encadrants**  Fabien DAGNAT  Antoine BEUGNARD  *Télécom Bretagne*  **Partenaire externe**  Sylvain Guérin  Vincent LEILDE  *Openflexo* |

1, rue de l’Institut

75000 PARIS – France

Tél. +33 (0)1 00 00 00 00

Siret : 000 000 000 000 00

APE : 0000 A

[**www.mines-telecom.fr**](http://www.mines-telecom.fr)

Mines ParisTech

Mines Albi

Mines Alès

Mines Douai

Mines Nantes

Mines Saint Étienne

Télécom ParisTech

Télécom Bretagne

Télécom SudParis

Télécom Ecole de Management

Télécom Lille1

Eurecom

**

Résumé

Un programme Java contenant un grand nombre de classes est souvent difficile à comprendre et à manipuler. Néanmoins, la compréhension de la structure de code est nécessaire pour travailler sur un programme existant. La représentation graphique est beaucoup plus facile à comprendre que le texte complexe. « Openflexo » est un atelier pour construire des outils de modélisation. Le projet « Edition intelligente de programme Java pour diatomée » consiste donc de rajouter un module pour permettre de construire en mémoire un graphe d'objet représentant un programme Java donné sous la forme d'un ensemble de fichiers Java, concevoir et implémenter un outil de visualisation qui facilite l'exploration d’un programme Java complexe et réaliser la synchronisation entre les fichiers sources et l’outil de visualisation. Dans ce rapport, je vous explique la façon de construire un graphe d’objet abstrait à partir des fichiers sources, la réalisation d’une présentation graphique et la synchronisation entre des fichiers sources et la présentation graphique.

Sommaire

1. introduction 3

1.1 Contexte 3

1.2 Problématique et objectif 3

2. Analyse Du domaine 3

2.1 Java meta modele 4

2.2 Outil de visualisation 6

3. déscription technique 6

3.1 Développement du prototype 6

3.1.1 Architecture d’implémentation 6

3.1.2 Module de construction de graphe d’objet abstrait (javaconnector) 7

3.1.3 Module de visualisation (javaconnector-ui) 10

3.1.4 Démonstration du prototype 12

3.2 Test et resultat 14

4. points d’Amélioration 14

5. references 16

# introduction

## Contexte

Les systèmes complexes et les grands systèmes sont difficiles à décrire et à concevoir et ils nécessitent l’intervention de nombreux spécialistes de diverses disciplines. Pour tous ce type de système, la synchronisation de l’intervention de métiers et d’experts varies est indispensable. L’infrastructure Openflexo [[1]](#r1) propose des composants et une méthodologie pour construire des ateliers de modélisation afin de résoudre ce problème. Dans Openflexo, tout est considéré comme un modèle. L’ingénierie dirigée par les modèles est une branche particulière de l’informatique qui s’intéresse aux techniques de capture, de transformation et de représentation des modèles. Openflexo n’est pas une application simple, c’est un logiciel d’infrastructure. La stratégie diatomée [[2]](#r2) est de relier les composants Openflexo et le modèle métier afin de construire un outil personnalisé aux utilisateurs. La plateforme Openflexo se compose avec des composants logiciels génériques pour que des entreprises puissent construire leurs propres outils de modélisation par assemblage et configuration de composants.

## Problématique et objectif

Openflexo contient déjà tous les composants pour manipuler et sérialiser des modèles graphiques, pourtant il ne supporte pas encore la manipulation du texte complexe. Par exemple pour un projet Java, Openflexo est capable de traiter les modèles UML, mais est incapable de traiter les fichiers sources de Java. L'outil fonctionne en maintenant en mémoire un graphe d'objet (Java) qui est manipulé par les éditeurs graphiques de l'outil et maintenu en synchronisation avec la version sérialisée des modèles (fichier dont la forme dépend du domaine, par exemple, XML, EMF, ...).

Le projet « Edition Intelligente de programme Java pour diatomée » consiste donc à implémenter un module permettant de construire en mémoire un graphe d'objet représentant un programme Java à partir des fichiers Java, maintenir la synchronisation entre des fichiers sources et le graphe d’objet en mémoire. Ce projet accordait aux utilisateurs d’Openflexo une possibilité de manipuler plus facilement un projet Java complexe contenant un grand nombre de classes.

# Analyse Du domaine

Le projet « Edition Intelligente de programme Java pour diatomée » inclut principalement deux modules à développer :

* Un module permettant de construire en mémoire un graphe d’objet abstrait à partir d’un ensemble des fichiers Java et maintenir la synchronisation entre les fichiers sources et le graphe d’objet. Un graphe d’objet représentant un programme Java est composé par différents éléments dans un projet Java. Le [Java Meta Model](#metamodel) joue le rôle du graphe d’objet.
* Un module réalise une nouvelle forme de représentation d’un programme Java à partir du graphe d’objet abstrait en mémoire et maintenir la synchronisation entre le graphe d’objet et la présentation graphique. L’objectif est de fournir un [outil de visualisation](#visualisation) permettant de faciliter la compréhension et la modification d’un programme Java.

## Java meta modele

Un langage de programmation peut avoir plusieurs Méta Modèles. Chaque Meta Model est plus ou moins approprié selon de différents besoins. Dans le cas de ce projet, j’utilise le code d’Openflexo comme exemple de travail. Autrement dit, le Java Meta Model que je propose est capable de représenter tous les éléments du projet Openflexo. Openflexo utilise « maven » [[3]](#r3) pour faire la gestion et l'automatisation de production des projets logiciels Java. L’outil de gestion de versions est « Git » [[4]](#r4). Le générateur « SableCC » [[5]](#r5) est aussi utilisé dans des modules d’Openflexo. Le fait de construire un Méta Modèle est de définir tous les éléments dans une application Java et les relations entre eux. Les éléments contiennent des éléments physiques (ex : fichier, répertoire, etc.) et des éléments logiques (ex : package, classe, interface, etc.). Il existe des relations entre deux éléments logiques (un package contient des classes), deux éléments physiques (une répertoire contient des fichiers). Il peut aussi y avoir des relations entre un élément physique et un élément logique (une classe est un fichier java).

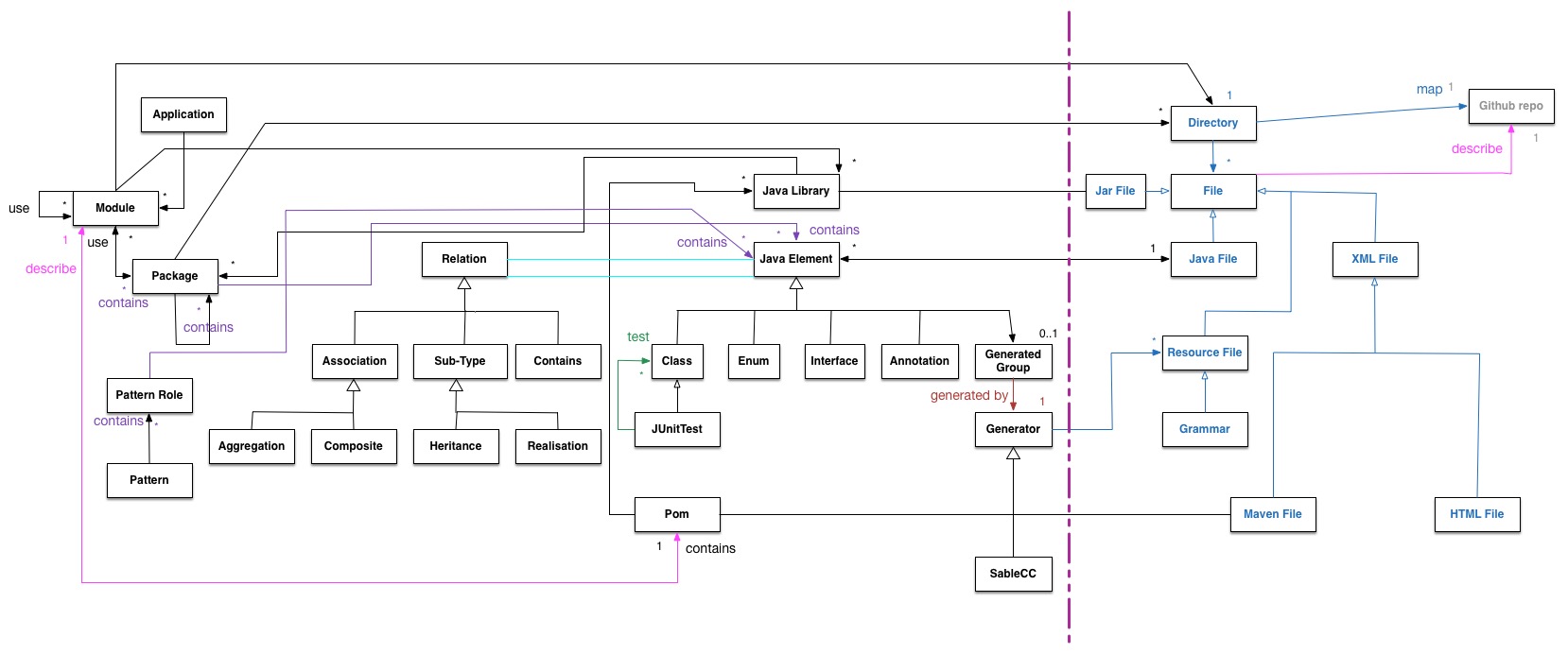
La [figure 1](#f1) montre le Java Méta Modèle en vue globale. La partie logique et la partie physique sont séparées. La partie droite est la partie physique donc contient des éléments physiques. La partie gauche est la partie logique donc consiste en éléments logiques.

Figure 1 : Java Méta Modèle global

La [figure 2](#f2) montre la structure plus en détail au niveau de classe. Une classe se compose avec des membres qui sont des attributs et des méthodes. Les attributs ont des différents types, les méthodes ont des paramètres et un type de retour.

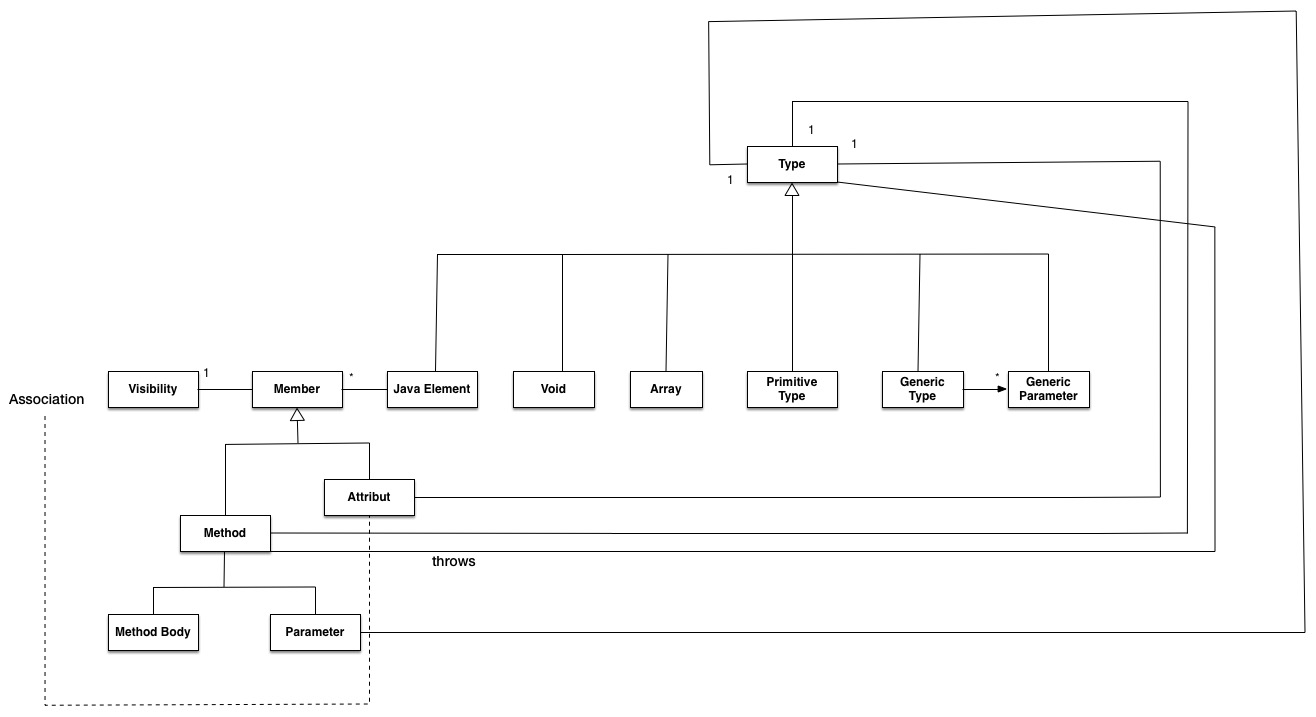


Figure 2 : Java Meta Model au niveau d’une classe

Openflexo utilise souvent des annotations qui eux aussi portent des informations. La [figure 3](#f3) présente le Méta Modèle au niveau des annotations. « Annotation Instance » est la définition d’un type d’annotation et « Annotated Element » donc est les éléments annotés.

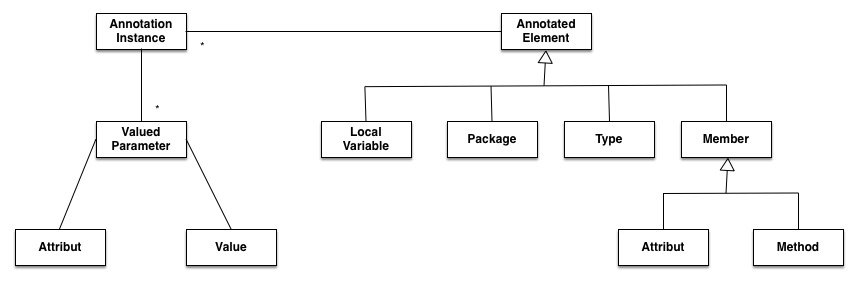


Figure 3 : Java Méta Modèle au niveau des annotations

## Outil de visualisation

Pour ce projet, la partie visualisation de graphe est la base pour représenter un programme Java complexe sous forme graphique. Il existe déjà des logiciels (ex : Sonar [[6]](#r6), Eclipse) qui sont capables de faire la représentation de programme. J’ai expérimenté des outils différents afin de trouver des idées pour finalement concevoir la structure la plus adaptée aux utilisateurs ce qui sont les développeurs. Finalement j’ai choisi une forme de représentation en cercle.

La [figure 4](#f4) est une présentation sous forme des packages. Le package « p » est la racine. Les packages « p1 », « p2 », « p3 », « p4 » sont des sous-packages de « p », ainsi de suite. En choisissant un package (ex : « p1 »), il devient la nouvelle racine, la graphe montre que les sous-packages de « p1 ». De cette manière, nous pouvons naviguer du niveau plus haut vers le niveau plus bas. Si le programme est très complexe, nous pourrions choisir d’afficher le nombre de niveaux.

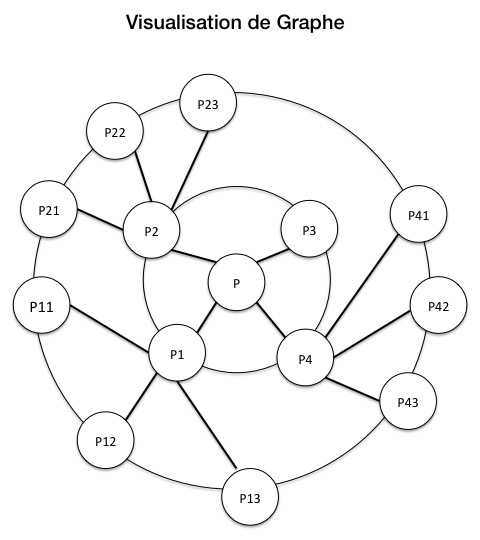


Figure 4 : Présentation d’un programme Java sous forme cercle

# déscription technique

## Développement du prototype

### Architecture d’implémentation

Comme ce projet est dans le cadre du logiciel Openflexo, j’ai analysé l’architecture d’Openflexo. La [figure 5](#f5) ci-dessous montre l’architecture de l’implémentation de ce projet au plateforme Openflexo. La partie en haut est l’espace de travail d’Openflexo. Openflexo construit en mémoire un graphe d’objet, chaque élément de ce graphe d’objet pointe vers des éléments dans le « source center » (la partie droite). « JAVATechnologyAdapter » est un connecteur permettant aux éléments du diagramme de pointer vers les sources et de se mettre en synchronisation avec les sources. L’outil de visualisation à gauche représente le projet Java sous forme graphique. Avec ce diagramme d’objet. L’outil est capable de représenter le programme. La synchronisation entre l’outil de visualisation et le diagramme d’objet est aussi indispensable.

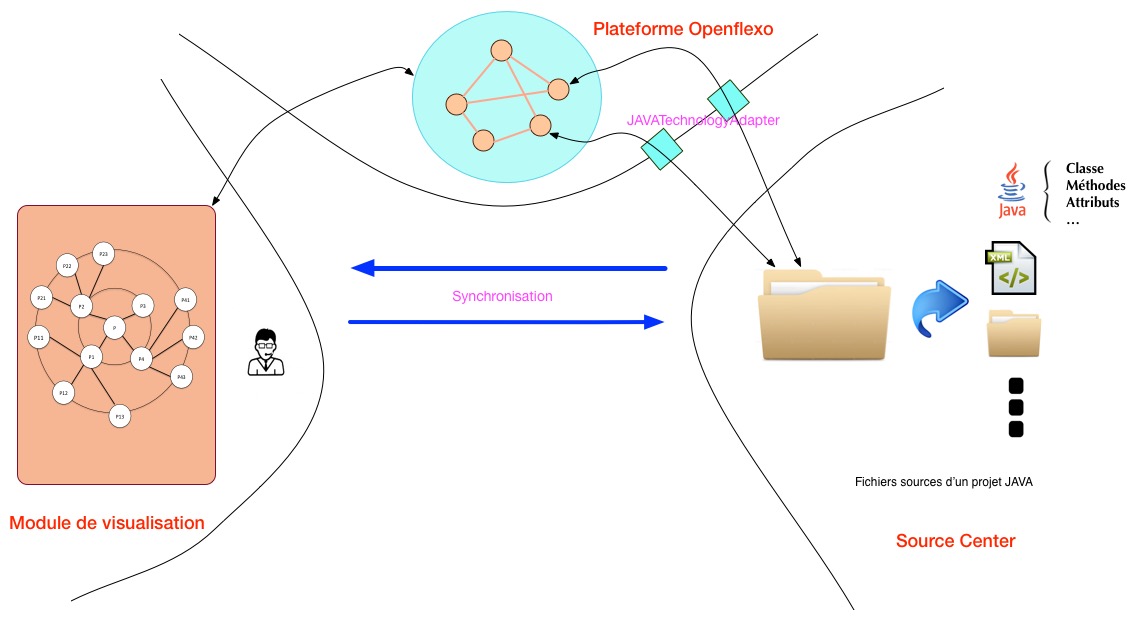


Figure 5 : Architecture de l’implémentation du projet dans openflexo

Les skeletons des modules que j’ai développés sont générés par un archétype donné dans « Openflexo repository ». L’avantage de développer des modules à partir d’un archétype est que la structure des modules est déjà définie. L’archétype a généré deux modules :

* « javaconnector » : ce module doit être réalisé pour construire un graphe d’objet à partir des fichiers sources et maintenir la synchronisation entre les fichiers sources et le graphe d’objet abstrait.
* « javaconnector-ui » :ce module doit être réalisé afin de fournir une représentation graphique d’un programme Java sous forme en cercle et maintenir la synchronisation entre le graphe d’objet abstrait et la représentation graphique.

### Module de construction de graphe d’objet abstrait (javaconnector)

Dans une vision hiérarchique, La racine contient 4 classes (« JAVATechnologAdapter », « JAVATechnologyContextManager », « JAVAModelSlot », « JAVAModeSlotInstanceConfiguration ») et 4 packages (« model », « fml », « rm », « utils »).

* Package « model »

Le package « model » contient des éléments prédéfinis dans un programme Java. Selon le Java Méta Modèle que j’ai présenté dans ce rapport, chaque élément doit être défini dans ce package. Pour l’instant, j’ai définir les éléments suivants : « JAVAFileModel », « JAVAClassOrInterfaceModel », « JAVAFieldModel », « JAVAMethodModel ».

La [figure 6](#f6) est le diagramme de classe représente le package « model ». J’ai fait une distinction entre la partie logique et la partie physique.

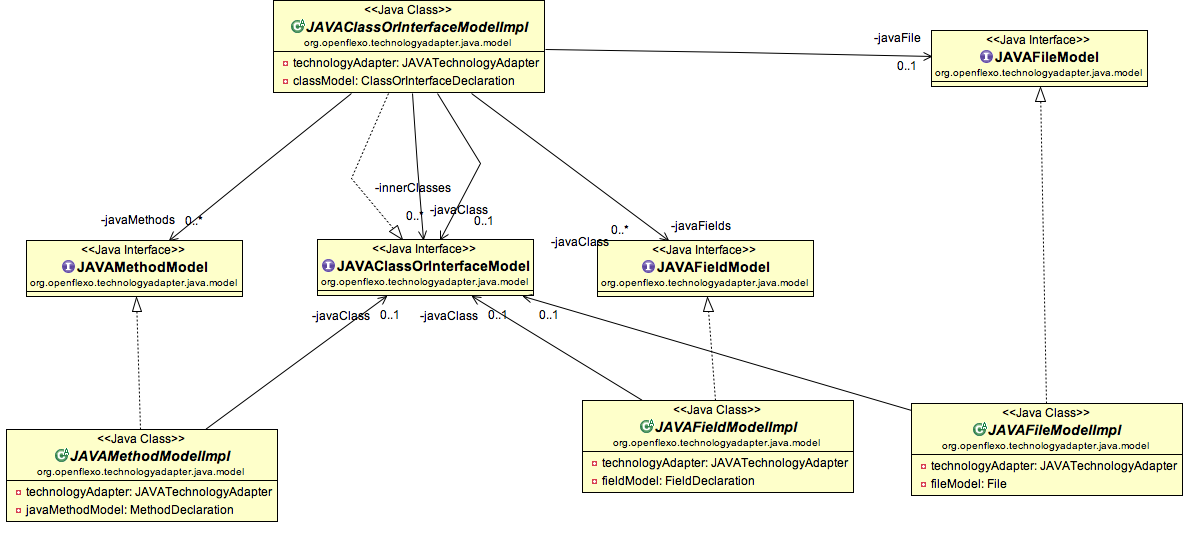


Figure 6 : diagramme de classe package « model »

* Package « fml »

Le nom « fml » signifie « Free Modelling Language ». Dans ce package, les différents rôles de fédération sont définis. Le projet « Edition intelligente de programme Java pour diatomée » n’a pas réalisé ce package.

* Package « rm »

Le package « rm » contient 2 classes et 1 interface pour gérer la ressource. La classe « JAVAResourceRepository » représente le « repository » qui stocke les ressources. La [figure 7](#f7) montre la structure de l’interface « JAVAResource » et sa classe d’implémentation « JAVAResourceImpl ». La classe contient toutes les méthodes qui seront utilisés pour gérer les ressources, comme « save », « load », « make», « retrieve » etc. La donnée de « JAVAResource » est « JAVAFileModel ». La méthode « loadResourceData » est définie pour obtenir la donnée « JAVAFileModel ».

### WeiWei:Users:wei:Travail:projet S5:structure code:rm.png

Figure 7 : diagramme de classe package « rm »

* Package « utils »

Le package « utils » définit des classes pour réaliser toutes les fonctionnalités utilitaires. Dans ce package, j’ai créé une classe « JAVAFileParser » qui utilise une librairie « JavaParser » [[7]](#r7) pour extraire les éléments dans un fichier .java. Ci-dessous ([figure 8](#f8)) est le diagramme de classe de « JAVAFileParser ».

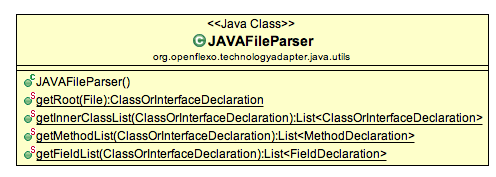


Figure 8 : diagramme de classe « JAVAFileParser »

* Classe « JAVATechnologyAdapter »

La classe « JAVATechnologyAdapter » ([figure 9](#f9)) gère les ressources dans le « ResourceCenter ». Chaque fois l’utilisateur lance le plateforme d’Openflexo, la méthode « initializeResourceCenter » est appelée. Le système Openflexo est capable de détecter l’ajout, la suppression le renommage des ressources. Les méthodes « contentsAdded », « contentsDeleted » affectent la modification sur le graphe d’objet.

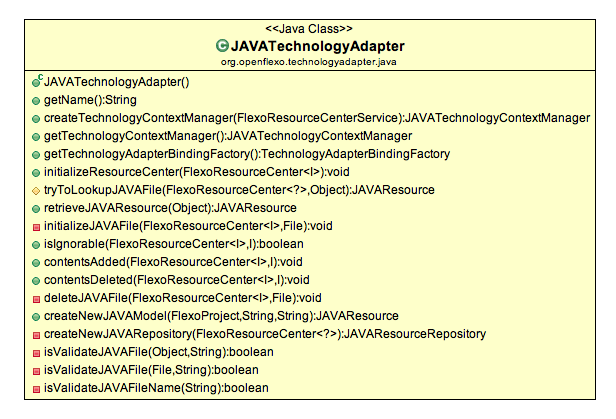


Figure 9 : diagramme de classe « JAVATechnologyAdapter »

### Module de visualisation (javaconnector-ui)

Le module « javaconnector-ui » contient 3 packages : « controller », « library » et « view ». Le package « controller » contient 2 classes : « JAVAAdapterController » et JAVATechnologyPerspective ». La classe « JAVAAdapterController » contrôle la création des « ModuleView ». J’ai défini un « JAVAperspective » propre au « JAVATechnology ». La classe « JAVAIconLibrary » dans le package « library » définit des icons utilisé dans le « perspective ». Le package principale qui réalise la représentation en cercle est « view ». Au début, j’ai utilisé « RadialTreeLayout » de librairie « Jung » pour réaliser cette représentation. Dans l’objectif de mieux intégrer le projet dans Openflexo, la librairie « Diana » fournie par Openflexo est utilisée en deuxième temps. Cette version n’est probablement pas la dernière version car un « layout » sera implémenté dans « Diana » pour réaliser la présentation d’un graphe en forme des cercles.

* Structure du graphe

La [figure 10](#f10) ci-dessous est la structure d’un graphe en cercle. « JAVAGraph » contient tous les « Node » dans ce graphe. Un « Node » est sûrement dans un « Graphe ». Chaque « Node » a des « inputEdge » et des « outputEdge ». Chaque « Edge » relie un « startNode » et un « endNode ». Les classes « JAVAGraph », « JavaNode », « JAVAEdge » utilisent le mécanisme « HasPropertyChangeSupport » défini par Openflexo. Par exemple, dans la classe « JAVANode », si les attributs « name », « inputEdges » et « outputEdges » changement, tous les objets à l’écoute sont notifiés. Grâce à ce mécanisme, la synchronisation entre le graphe d’objet et la représentation graphique est assurée.

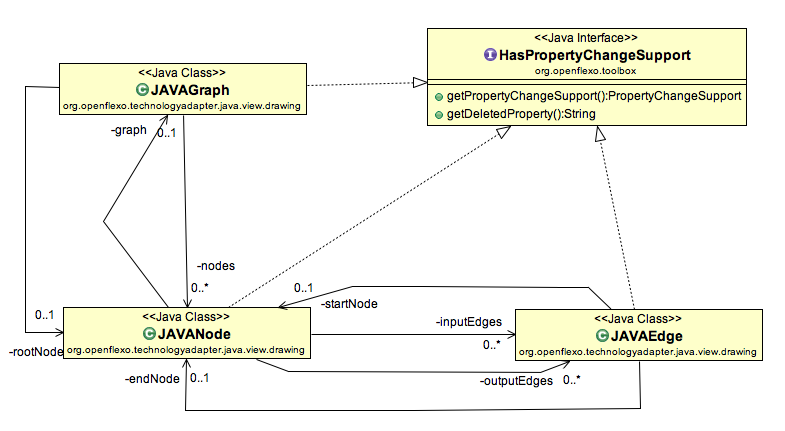


Figure 10 : diagramme de classe de la structure d’un graphe

* Construction d’un « ModuleView »

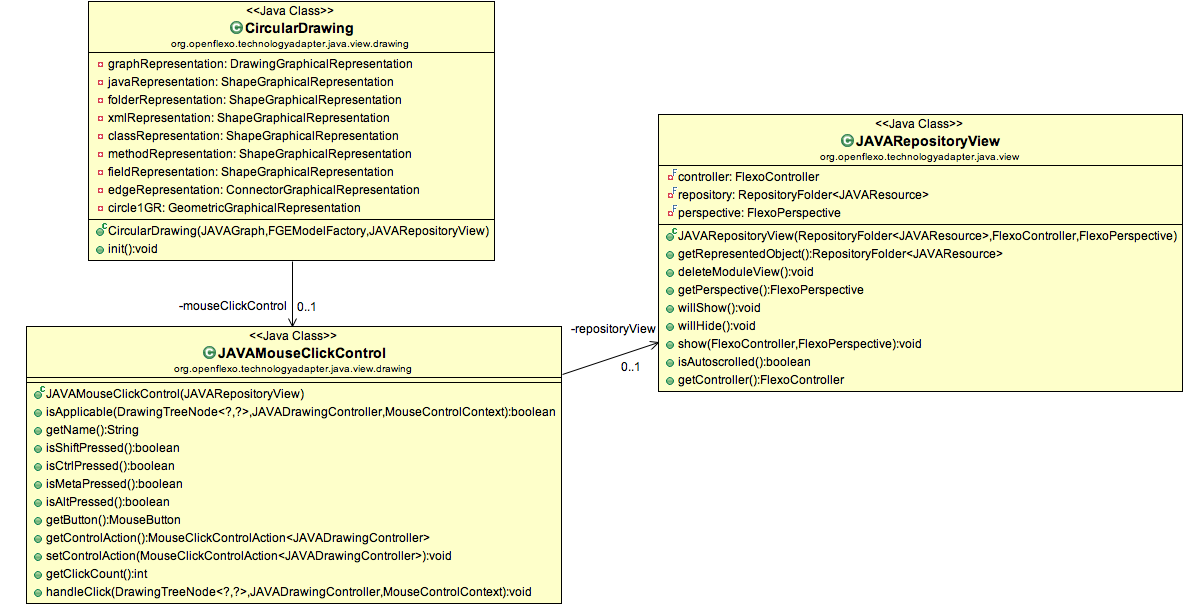
La [figure 11](#f11) montre les classes qui construisent un « RepositoryView ». La classe « CircularDrawing » hérite la classe « DrawingImpl » de « Diana ». La seule méthode à implémenter est « init ». Dans cette méthode, j’ai défini les « DynamicPropertyValue » pour observer le changement du nom et de la position des « node ».

nodeBinding.setDynamicPropertyValue(GraphicalRepresentation.***TEXT***, **new** DataBinding<String>("drawable.name"), **true**);

nodeBinding.setDynamicPropertyValue(ShapeGraphicalRepresentation.***X***, **new** DataBinding<Double>("drawable.circularX"), **true**);

nodeBinding.setDynamicPropertyValue(ShapeGraphicalRepresentation.***Y***, **new** DataBinding<Double>("drawable.circularY"), **true**);

La classe « JAVAMouseClickControl » réalise la fonctionnalité de naviguer du niveau le plus vers le niveau le plus bas.

Figure 11 : diagramme de classe de construction d’un « ModuleView »

### Démonstration du prototype

Le projet « Edition intelligente de programme Java pour diatomée » est facile à intégrer dans Openflexo. La [figure 12](#f12) est l’interface d’Openflexo module. En double-cliquant sur un « repository », par exemple « apache\_tomcat », une fenêtre s’ouvre avec une représentation en cercle d’une arborescence d’un répertoire ([figure 13](#f13)).

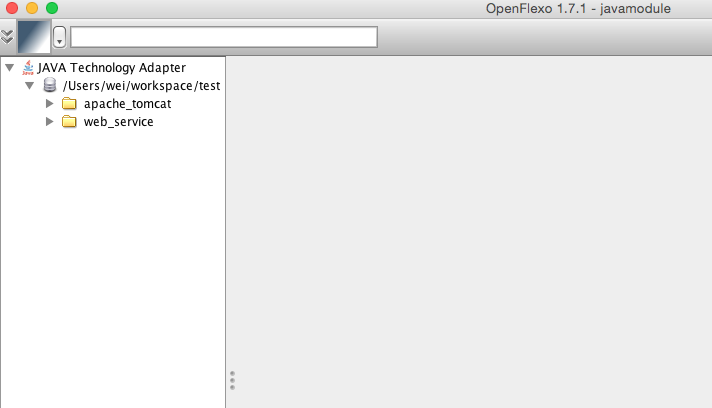


Figure 12 : plateforme Openflexo

Le répertoire « apache\_tomcat » est la racine. Les répertoires « test », « monMagasin » et « consommation » sont des sous-répertoires de « apache\_tomcat » et ainsi de suite.

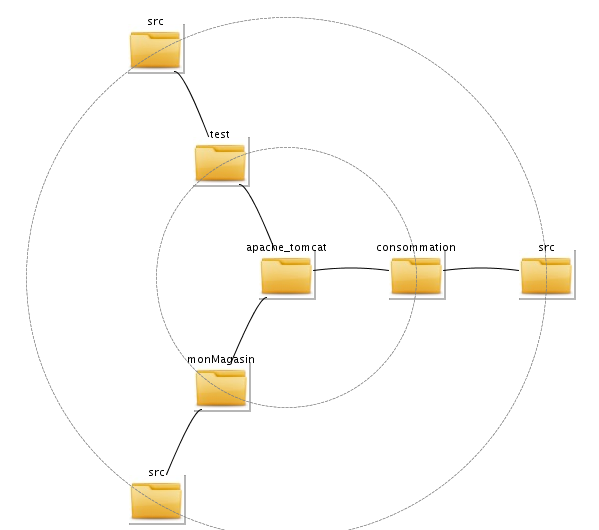


Figure 13 : représentation en cercle 1

Cliquer sur un répertoire en appuyant sur la touche « control », le répertoire choisi devient la nouvelle racine ([figure 14](#f14)).

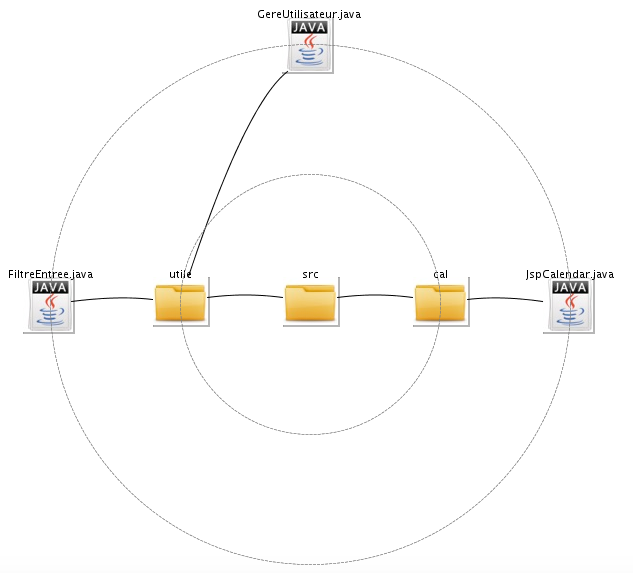


Figure 14 : représentation en cercle 2

La [figure 15](#f15) extrait les attributs, les méthodes d’une classe. Les ronds en jaune sont des méthodes et ceux qui sont en bleu sont des attributs.

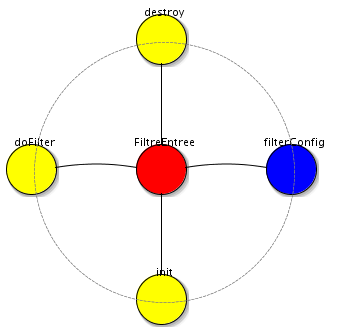


Figure 15 : représentation en cercle 3

## Test et resultat

Comme j’utilise le code source d’Openflexo comme exemple d’un programme Java pour construire le Java Méta Modèle, j’utilise aussi le code source d’Openflexo pour tester les fonctionnalités du projet. Grâce aux tests que j’ai fait, les fonctionnalités ci-dessous sont bien réalisées :

* Lire l’arborescence d’un répertoire et représenter la structure du répertoire sous forme en cercle.
* Extraire la structure d’un fichier .java (classe interne, attribut, méthode) et la représenter sous forme en cercle.
* Modifier le nom d’un « java file » ou déplacer un fichier d’un répertoire vers un autre.

# points d’Amélioration

Le but de ce projet est de construire un ensemble d'outils pour manipuler plus facilement un projet Java complexe contenant un grand nombre de classes. Tous ce que j’ai implémenté est un premier pas. Ci-dessous sont des points d’amélioration que j’ai prévu pour finalement réaliser un produit final :

* Construction d’un graphe d’objet complet

Selon le Java Méta Modèle ([figure 1](#f1), [figure 2](#f2), [figure 3](#f3)) que j’ai construit, un programme Java contient beaucoup d’éléments. Pour l’instant, je n’ai que implémenté certains éléments : répertoire, fichier, classe, attribut, méthode. Pour rendre un outil avec des fonctionnalités assez complètes et maintenir la synchronisation entre les fichiers sources et le graphe d’objet, il faut prévoir d’implémenter tous les éléments que j’ai définis dans le Java Méta Modèle. Une fois que tous les éléments sont implémentés et la synchronisation est réalisée, il faut aussi arriver à réaliser la partie visualisation. Le module de visualisation doit donc être capable de représenter tous les éléments d’un Java Méta Modèle sans oublier la synchronisation.

* Modification du programme Java

Comme l’objectif final du projet est de proposer un ensemble d’outils pour que un programme Java deviennent facile à comprendre et à modifier. La modification du nom de fichier, le déplacement du fichier d’un package vers un autre est réalisé. Mais le fait de modifier un fichier source Java, il reste des points à réfléchir :

1. La modification du fichier n’entraîne pas d’erreurs dans le code. Autrement dit, après la modification, le programme doit être compilable.
2. La notion « refactoring » [[8]](#r8) est respectée. La maîtrise des modifications apportée au code est importante.

* L’extension à d’autres langages de programmation

Le projet est concentré sur le langage de programmation Java, mais il y a toujours des points en commun entre de différents langages de programmation (C, C++, Python, etc.). Une fois que ce produit fonctionne bien pour les programmes Java, il faut aussi réfléchir à l’extension des autres langages de programmation.

# references

[1] <https://openflexo.org/tiki/Discover+Model+Driven+Engineering>

[2] [https://openflexo.org/tiki/Diatomée](https://openflexo.org/tiki/Diatom%C3%A9e)

[3] <http://maven.apache.org>

[4] <http://git-scm.com>

[5] <http://sablecc.org>

[6] <http://nemo.sonarqube.org>

[7] <https://code.google.com/p/javaparser/>

[8] <http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Freference%2Fref-menu-refactor.htm>

|  |  |
| --- | --- |
|  | Technopôle Brest-Iroise CS 83818 29238 Brest Cedex 3 France  +33 (0)2 29 00 11 11  **www.telecom-bretagne.eu** |
| INSTITUT-MINES-TELECOM_Logo_Buro_petit | |