Projet Econet
-
Système à base de règles
Spécifications des règles

Aurélia COUVRAND
Mathieu VÉNISSE

16 mars 2009

Table des matières

Chapitre 1

Introduction

L'architecture du plug-in étant construite, il est désormais temps de spécifier les règles nous permettant d'extraire une structure à composants d'un projet Java.

Par manque de temps, nous n'utilisons que les règles déjà définies. Ainsi, ce rapport sera relativement similaire à celui sur l'étude du SBR (Système à Base de Règles) existant, à la nouvelle architecture du plug-in près.

Une fois les spécifications faites, nous pourrons partager le travail pour implémenter ces règles.

Chapitre 2

Spécifications des règles

2.1 Récupération des types d'intérêt

Nom : *getTypesOfInterest*
Type de règle : *SimpleRule*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : ensemble de *IType*
Explications : Elle correspond à la méthode *getTypesOfInterest* du plugin existant. Nous aurons besoin de reprendre la méthode *getUnitsOfInterest* dans la classe ASTActionDelegate pour trouver les unités de compilation du projet ; en effet, seuls les types définis dans ces unités nous intéressent. Elle retourne un ensemble de IType ; il nous faudra choisir la structure utilisée (vecteur, tableau, hashmap, liste...). Elle permettra l'instanciation de l'attribut *typesOfInterest* de la classe *ASTActionDelegate* en parcourant chaque unité de compilation du projet analysé et en récupérant le type de haut niveau de cette unité. Nous avons pensé à ne rien la faire retourner, mais étant donné que *typesOfInterest* sera un attribut privé de *ASTActionDelegate* et qu'on veut disposer de ces types, nous sommes obligés de renvoyer l'ensemble des types.

2.2 Marquage des data types

Nom : *flagDataTypes*
Type de règle : *GroupOfRules*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : *void*
Explications : Elle permet de reconnaître les data types ; nous devrons étudier une expérimentation pour voir si certains restent. Détaillons ses sous-règles.

2.2.1 À partir des paramètres des méthodes non statiques autres que les constructeurs

Nom : *flagMethodsParametersType*
Type de règle : *SimpleRule*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : *void*
Explications : Elle permet de parser les méthodes, autres que les constructeurs et les méthodes statiques, de chaque élément de l'ensemble *typesOfInterest* et de marquer comme data type le type de leurs paramètres.

2.2.2 À partir des paramètres des méthodes statiques

Nom : *flagMainMethodParametersType* ou *flagStaticMethodsParametersType*
Type de règle : *SimpleRule*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : *void*
Explications : Elle permet de parser les méthodes statiques du projet. Après avoir cherché dans la documentation sur JDT, nous nous sommes rendus compte que nous n'avons pas moyen d'identifier les méthodes statiques autres que le main ; ça veut dire que la précédente règle considère déjà les potentielles méthodes statiques, le main excepté. Nous allons donc devoir ajouter une hypothèse supplémentaire, à savoir qu'il n'y a qu'une seule méthode statique par projet : le main. À noter que le rapport entre les méthodes statiques et une approche à composants n'a toujours pas été éclairci ; la méthode sera implémentée au cas où l'utilisateur en ait besoin.

2.2.3 D'après les getters et setters

Nom : *flagFromGettersAndSetters*
Type de règle : *SimpleRule*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : *void*
Explications : Elle permet de marquer comme data types les types possédant plus de 70% d'attributs dipsosant d'accesseurs et de modifieurs, le seuil ayant été fixé avec les encadrants lors d'une réunion ; les expérimentations nous révèleront si ce seuil doit être modifié.

2.2.4 D'après le type des attributs

Nom : *flagFromNonComponentAttributes*
Type de règle : *SimpleRule*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : *void*
Explications : Elle permet de marquer comme data types les classes dont le type des attributs soit n'appartient pas à l'ensemble *typesOfInterest*, soit est un data type. Pour valider cette règle, il faudra étudier le réseau des communications lors d'une expérimentation.

2.2.5 Énumérations et structures de données

Nom : *flagEnumAndDataStructures*
Type de règle : *SimpleRule*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : *void*
Explications : Elle permet de marquer les énumérations et les classes implémentant des structures de données comme types de données.

2.2.6 D'après la visibilité des attributs

Nom : *flagFromAttributesVisibility*
Type de règle : *SimpleRule*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : *void*
Explications : De la même façon que concernant les accesseurs et modificateurs, un pourcentage seuil est introduit, tel que si le pourcentage d'attributs publics est supérieur à ce seuil, alors le type est marqué comme type de données. Le seuil est initialisé à 70%, mais est appelé à être modifié si les expérimentations ne sont pas concluantes.

2.3 Marquage des composants

Nom : *flagComponents*
Type de règle : *GroupOfRules*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : *void*
Explications : Elle permet de reconnaître les composants. Détaillons ses sous-règles.

2.3.1 D'après les interfaces (au sens Java)

Nom : *flagFromJavaInterfaces*
Type de règle : *SimpleRule*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : *void*
Explications : Elle permet de marquer comme composants les types implémentant des interfaces Java ou dont les champs référencent de telles interfaces.

2.3.2 D'après la déclaration d'un type

Nom : *flagFromTypeDeclaration*
Type de règle : *SimpleRule*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : *void*
Explications : Nous avons extrait cette règle d'après l'implémentation de la règle P3 du plug-in existant : les types non encore marqués comme composants ou types de données sont parsés ; s'ils ont déclarés comme classes, ils sont marqués comme composants. L'expérimentation sur CoCoME n'ayant pas été satisfaisante, cette règle est appelée à être modifiée, voire peut-être à disparaître ; d'autres expérimentations nous permettront de déterminer ce qu'elle deviendra.

2.4 Extraction des structures composites

Nom : *AnalyseCompositeStructures*
Type de règle : *SimpleRule*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : *void*
Explications : Cette règle permet d'analyser la structure composite des composants. Elle part du programme principal puis, par les différents appels aux constructeurs, permet de savoir quels sont les composants inclus dans les autres. Bien entendu, on ne prend en compte que les constructeurs appelant des types identifiés comme étant des composants. La règle marquant les composants doit donc avoir obligatoirement été appliquée avant.

2.5 Extraction des communications inter-composants

Nom : *findComponentsLinks*
Type de règle : *SimpleRule*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : *void*
Explications : Cette règle permet d'analyser les messages envoyés par les méthodes. On considère qu'il y a communication entre un composant A et un composant B si A dispose d'une méthode M et que B effectue un appel à cette méthode dans le corps de l'une des siennes. Cette règle permet d'établir les différents liens entre les composants identifiés. Tout comme la précédente, cette règle devra forcément être appelée après celle marquant les composants.

2.6 Identifications des interfaces requises et fournies

Nom : *findInterfaces*
Type de règle : *SimpleRule*
Paramètre(s) d'entrée : *ASTActionDelegate*
Paramètre(s) de sortie : *void*
Explications : Cette règle permet d'identifier les interfaces requises et fournies des composants. Tout comme les deux précédentes, cette règle devra être appelée après celle marquant les composants. Elle prend en entrée un *ASTActionDelegate* et parcourt les méthodes des composants identifiés. Il faut alors déterminer si les méthodes représentent des services fournis ou requis. On s'intéresse au méthodes de visibilité *public* ou *package*.

Chapitre 3

Conclusion

La spécification des règles étant maintenant faite, le travail peut être partagé entre plusieurs membres du groupe. Voici une première répartition (tous les membres concernés n'ayant pas été concertés, elle pourrait changer dans les jours qui viennent) :

- **Récupération des types d'intérêt** : Mathieu V.
- **Marquage des data types** : Aurélia
- **Marquage des composants** : Mathieu V.
- **Extraction des structures composites** : Matthieu A.
- **Extraction des communications inter-composants** : Matthieu A. (ou Mathieu V.)
- **Identifications des interfaces requises et fournies** : Jérémie