FISE A2 - TAF DCL
LERQUEMAIN Yann - *yann.lerquemain@imt-atlantique.net*
RABILLOUD Quentin - *quentin.rabilloud@imt.atlantique.net*

Méthodes avancées de programmation et de développement logiciel
Rapport final

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Diagramme de classe

Conception initiale

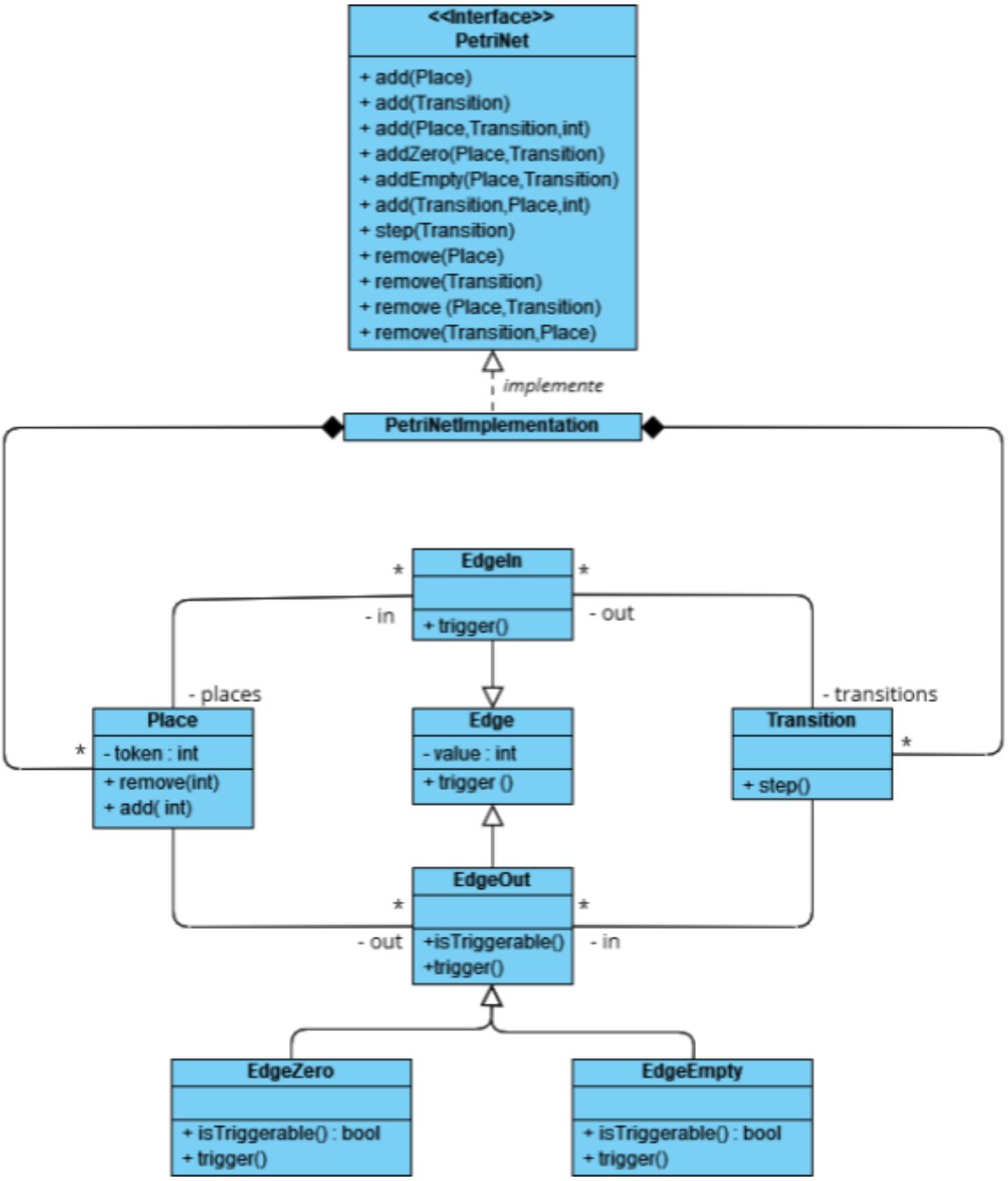

Figure 1 : Diagramme de classe de la conception initiale

Rétro-ingénierie de la conception finale

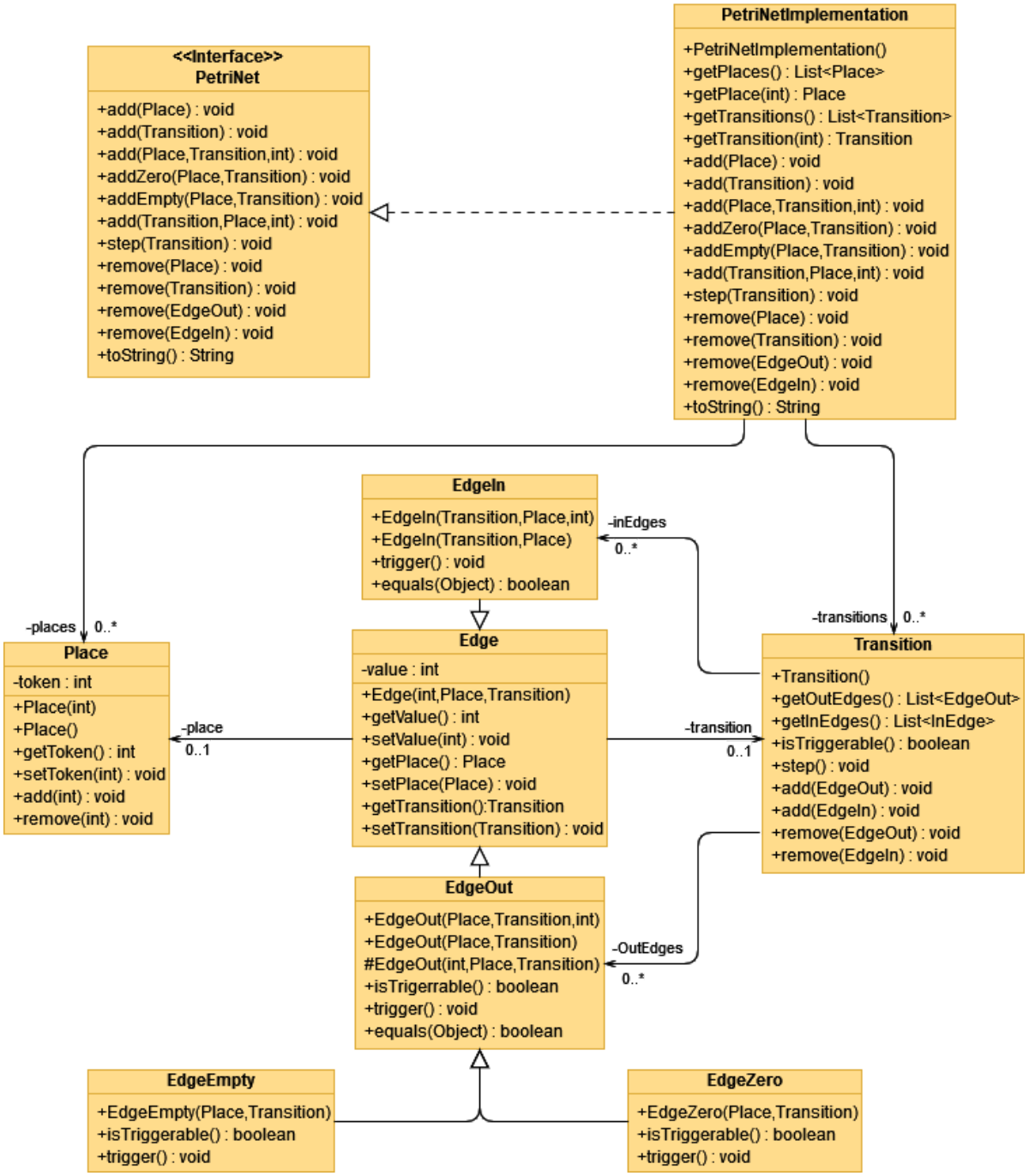

Figure 2 : Diagramme de classe de la conception finale

Justification des modifications

Structure et attributs

Entre ces deux diagrammes, la différence principale de structure est l'association entre la classe **Edge** et la classe **Place**, qui remplace les associations **EdgeIn**-**Place** et **EdgeOut**-**Place**. Ce choix de modification permet de mieux exploiter le polymorphisme des classes qui héritent de **Edge**. En effet, l'interaction entre **Place** et **Edge** est la même, que cela soit dans le cas d'un **EdgeIn** et d'un **EdgeOut** : Il s'agit dans les deux cas de modifier l'attribut *token*, ce qui se fait via la classe **PetriNetImplementation**.

A présent, les instances de **Edge** contiennent en attribut la **Place** (attribut *place* de cardinalité 0..1) et la **Transition** (attribut *transition* de cardinalité 0..1) qu'ils relient. Cela permet de modifier, directement, au niveau de l'instance de **Edge**, son origine et sa destination, via les méthodes setPlace() et setTransition(), et également d'accéder à ces informations via les getPlace() et getTransition().

Ainsi, et compte tenu de ces modifications, il n'est donc plus nécessaire de contenir les attributs *in* et *out* pour la classe **Place**, cette information étant stockée dans les instances de **Transition** et **Edge**.

Méthodes

Pour ce qui est des méthodes de classes, il a été décidé d'ajouter des constructeurs additionnels, selon les cas d'usage. Ainsi, pour le cas des classes **EdgeIn** et **EdgeOut**, un constructeur sans argument *int* a été ajouté, afin d'avoir un constructeur prenant *value*=1 par défaut. Il en va de même pour **Place**, prenant alors *token*=0 par défaut.

Un constructeur additionnel supplémentaire, passé en protected, a été ajouté à **EdgeOut** pour contourner une exception quant à la valeur des **EdgeZero** et **EdgeEmpty**, qui prennent une *value* placeholder de -1 par défaut, ce qui levait une exception dans **Edge** à leur création.

On vérifie également à présent à plusieurs reprises, pour assurer la robustesse du modèle, si les instances ont isTriggerable()=True. On a ainsi ajouté à **Transition** cette méthode.

Les options de manipulation de *token* ont été changées pour **Place** : il ne s'agit donc plus de add() ou remove(), qui fonctionnaient uniquement de façon incrémentale ou décrémentale unitaire, mais de modifier l'attribut *token* via des quantités choisies, ou même, avec setToken(), d'indiquer la valeur de l'attribut directement.

Analyse statique avec STAN

Voici les résultats globaux obtenus après analyse statique par le plug-in STAN.

Count	
Libraries	1
Packages	5
Units	15
Units / Package	3
Classes / Class	0
Methods / Class	6.2
Fields / Class	0.53
ELOC	1064
ELOC / Unit	70.93
Complexity	
CC	2.34
Fat - Libraries	0
Fat - Packages	11
Fat - Units	58
ACD - Library	0.00%
ACD - Package	55.00%
ACD - Unit	53.81%

Robert C. Martin	
D	-0.33
|D|	0.49
Chidamber & Kemerer	
WMC	14.53
DIT	1.73
NOC	0.53
CBO	1.87
RFC	11.33
LCOM	22.4

De ces métriques, on peut distinguer des points qui permettent de décrire le programme, et également les potentiels axes d'amélioration.

En effet, on remarque tout d'abord que le programme est relativement simple, comme indiqué par la complexité cyclomatique (**CC**), située entre 1 et 10. Cela signifie que le nombre de chemins de décisions est plutôt faible, ce qui facilite la maintenance, la relecture, et le test du programme. De plus, allant dans ce sens, la profondeur moyenne des arbres d'héritage (**DIT**) est relativement faible, facilitant la maintenance et la modification du code. Enfin, Une distance moyenne (**D**) négative, ici à -0.33, suggère une certaine stabilité du code, facilitant encore une fois la maintenance de celui-ci. Mais penchons-nous davantage sur cet indicateur.

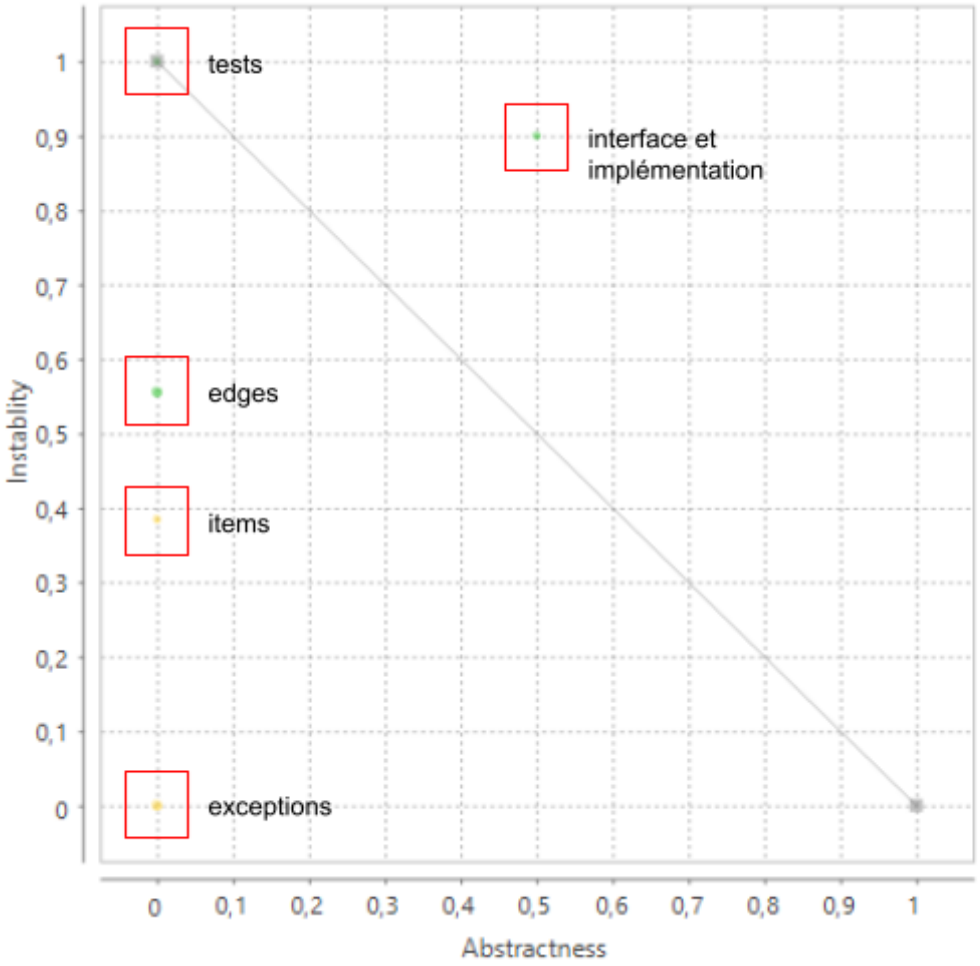

Figure 3 : Graphe Instabilité/Abstraction fourni par STAN

On remarque immédiatement que les exceptions sont les classes ayant la plus grande distance à la séquence principale, ce qui est cohérent, car elles ne contiennent aucune abstraction, et n'ont aucune dépendance.

On remarque également que la classe la plus proche de la séquence principale est celle des tests. En effet, elles ne font appel à aucune abstraction, et n'ont, bien sûr, que des dépendances, donc une instabilité maximale.

Un point d'attention à soulever concerne surtout l'interface **PetriNet** et son implémentation **PetriNetImplementation**, qui ont trop de dépendance pour leur haut niveau d'abstraction.

Cette analyse est à mettre en lien avec la métrique de dépendance moyenne entre les packages (**ACD - Package**), qui est trop élevée : les packages ont, selon cette métrique, trop de dépendances entre eux. Dans l'idéal, cette métrique se situerait en dessous de 50%.