

CHAPITRE 4

GENERATION DES OUTILS POUR LA TRANSFORMATION DES DC VERS RDP

4.1 Introduction

Avant d'aborder les techniques de transformation de modèle et les règles de la transformation, il est nécessaire de définir le méta-modèle des deux outils de modélisation « les diagrammes de classe UML » et les « Réseau de Pétri », et après en passe à les règles de transformation de ces modèles.

Pour définir et réaliser les deux méta-modèles on va utiliser le standard EMF qu'on a déjà abordé dans le chapitre 02.

4.2 Génération des outils pour « les diagrammes de classe » et « les Réseaux de Pétri »

En se basant sur ces deux méta-modèles EMF nous permet de générer des outils pour la création des exemples « des diagrammes de classe » et des « Réseaux de Pétri ».

4.2.1 Un méta-modèle pour le diagramme de classe

Pour définir un méta-modèle (ou une ecore) EMF utilise le diagramme de classe. Donc, une DC (comme le montre le méta-modèle de la figure 4.1) est composée de sept classes : *Package, Association, Entity, Feature, Attribute, Method, Parameter*.

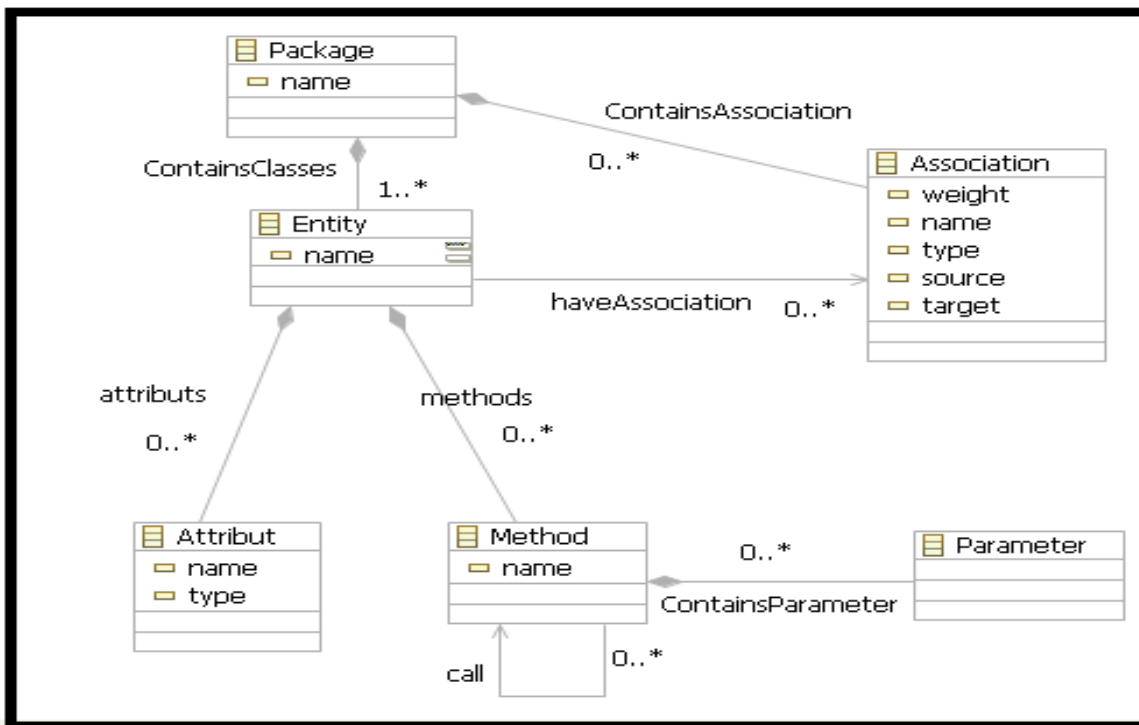


Figure 4.1 : le méta-modèle de diagramme de classe

4.2.2 La génération d'un outil pour le diagramme de classe

En se basant sur le méta-modèle (voir la figure 4.1) on peut générer un outil qui nous permettra de créer des exemples d'architectures logicielles (des instances) avec les étapes suivantes :

- **Création d'un projet EMF vide (File → New → Project...)**

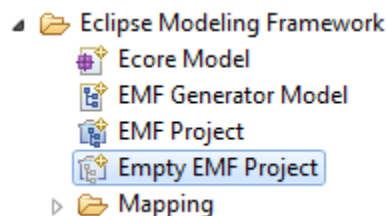


Figure 4.2 Crée projet EMF vide.

➤ **Création d'un diagramme Ecore (New → Other →)**

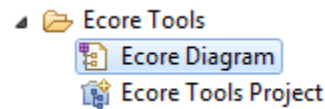


Figure 4.3 Crée diagramme Ecore.

➤ **Défini les éléments de notre méta-modèle de DC**

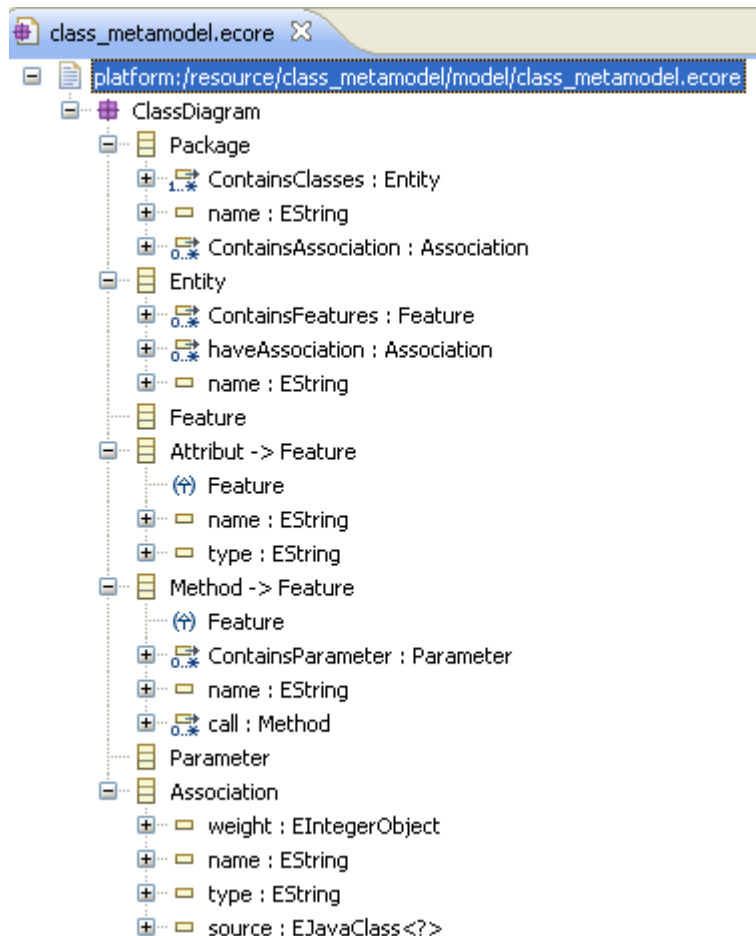


Figure 4.4 : les éléments de méta-modèle de DC.

➤ **Création de *genmodel***

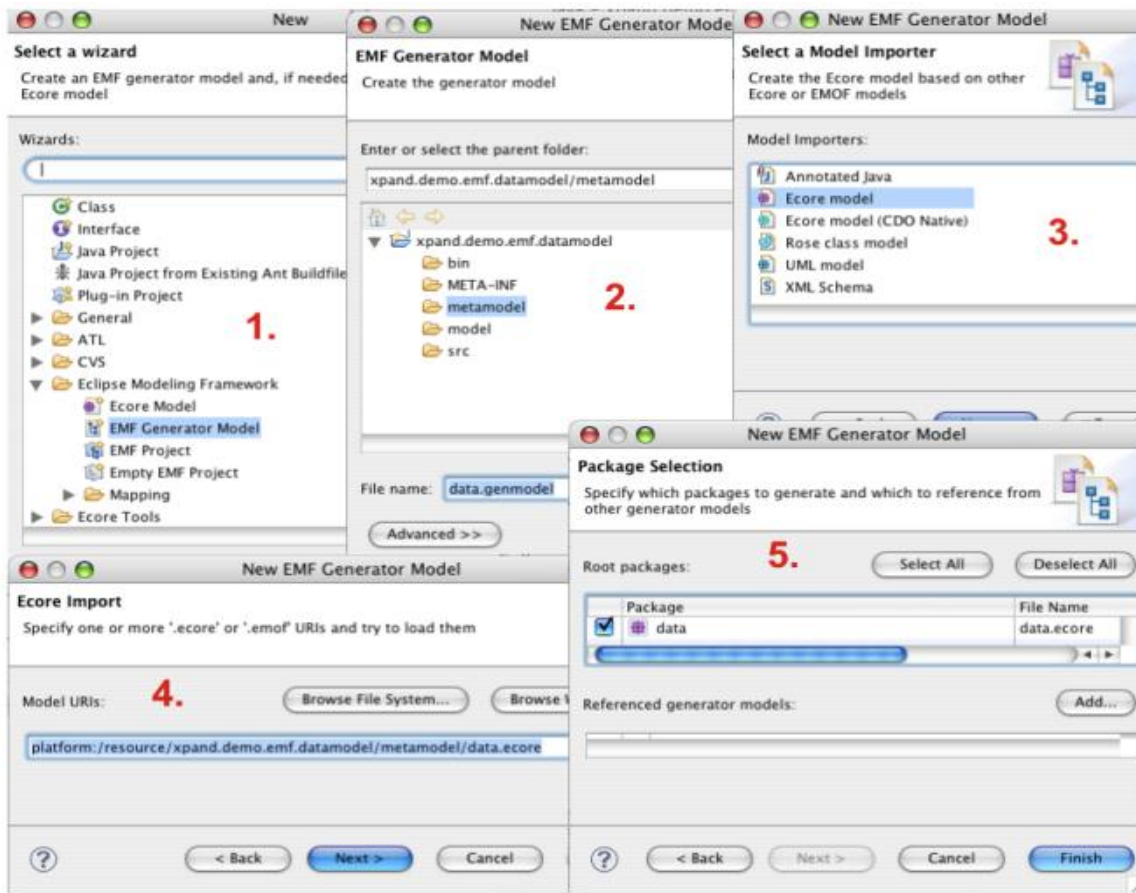


Figure 4.5 : génération de *genmodel*

➤ **défini Les éléments de génération de notre model DC**

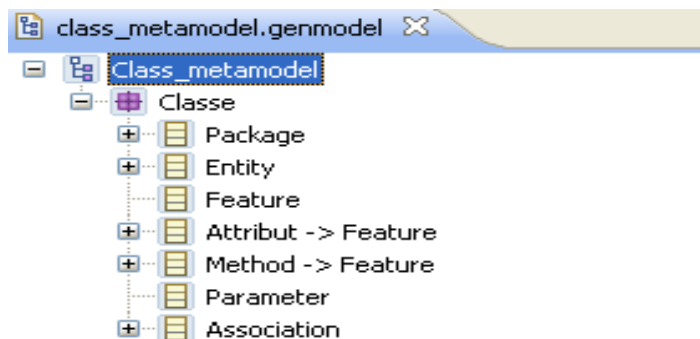


Figure 4.6 : les éléments de *genmodel* de DC.

➤ **Généré les projets `édit`, `editor`, et `tests`**

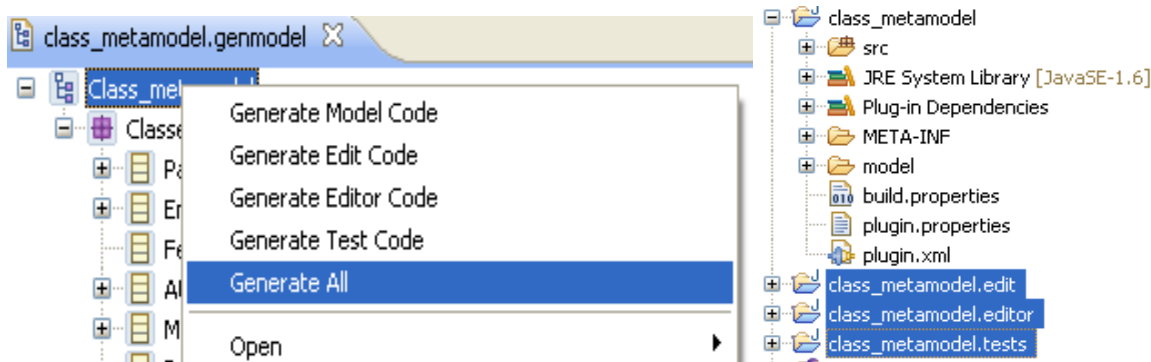


Figure 4.7 : génération des projets `.edit`, `.editor`, `.tests`

➤ **Exécuter les plugins :**

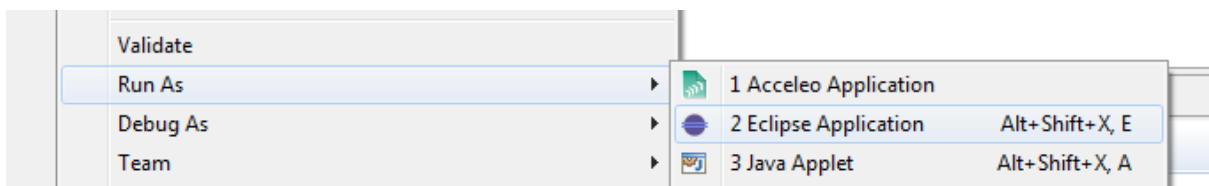


Figure 4.8 : exécuter les plugins

Pour tester les nouveaux plugins, une deuxième instance d'Eclipse doit être lancée, et pour vérifier que les plugins déjà produit, amenez le « Help/About Eclipse plate-forme » dialogue, clique sur le bouton « plug-in Détails », et vérifie.

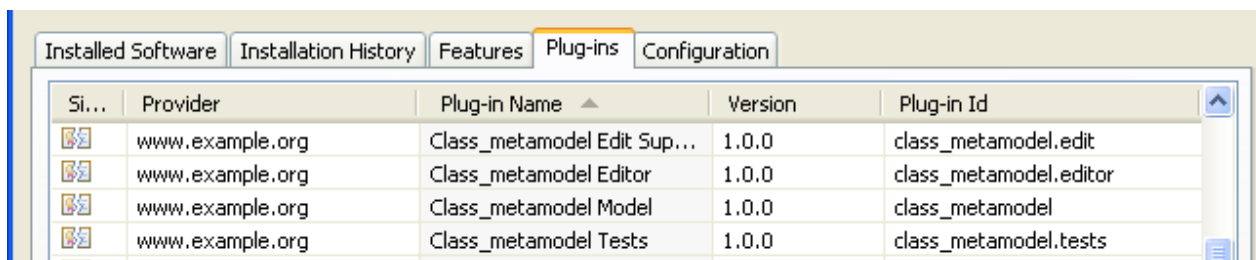


Figure 4.9 : plugin détails de classe modèle

➤ **Création d'une nouvelle modèle de diagramme de classe**

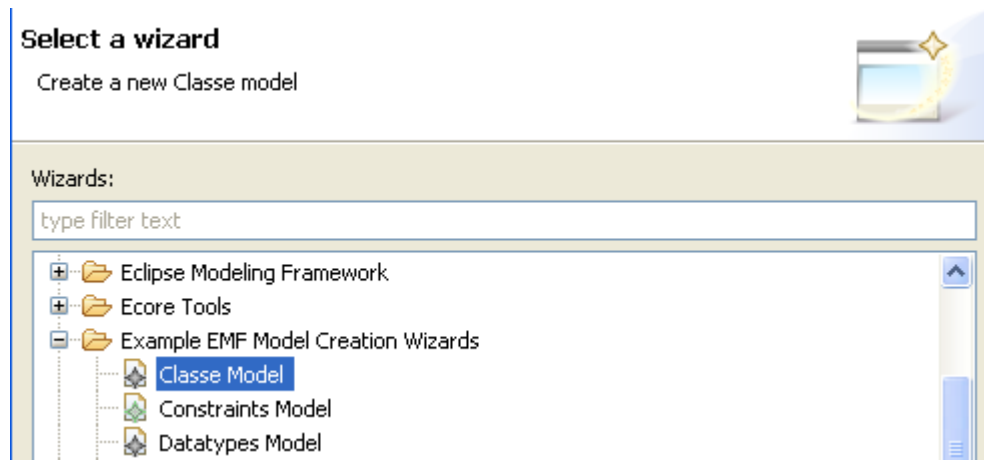


Figure 4.10 : création d'une classe modèle

4.2.3 Un méta-modèle pour le réseau de pétri

Notre méta-modèle de réseau de pétri est composé à Cinq classes : **PetriNet**, **Place**, **Transition**, et **Arcs**.

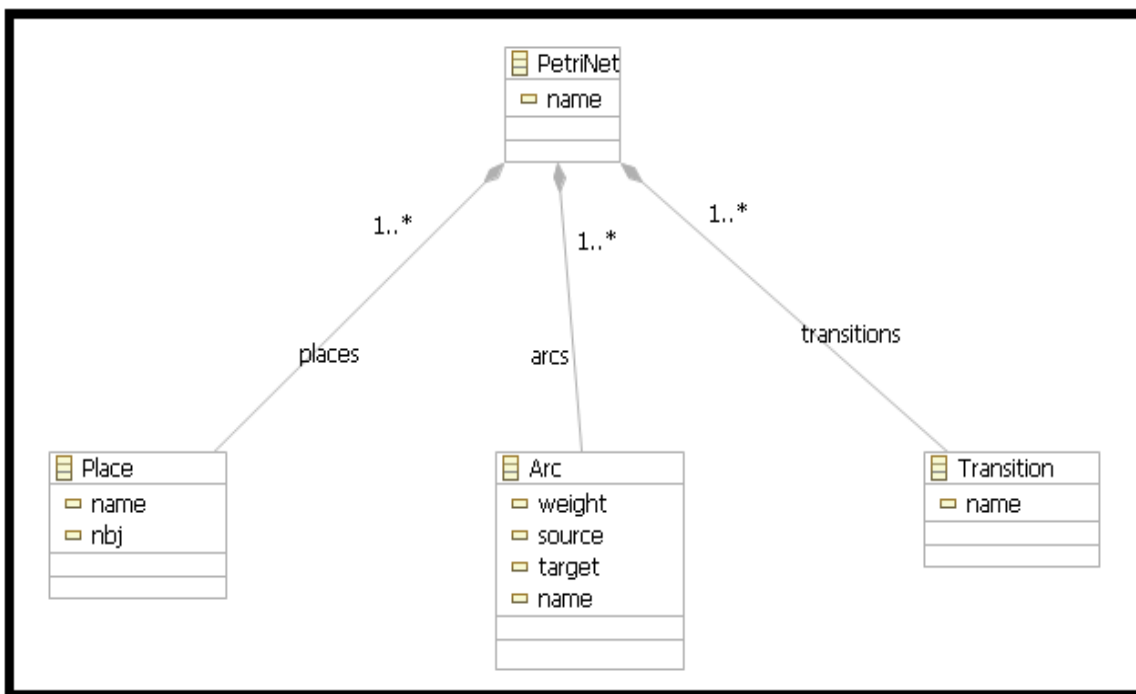


Figure 4.11 : le méta-modèle de réseau de pétri

4.2.4 La génération d'un outil pour le réseau de pétri

Pour générer un outil de réseau de pétri, on va passer à la même manier qu'on a vue avec la génération d'un outil pour le diagramme de classe.

Donc on va présenter sauf les étapes spécifiques au modèle réseau de pétri.

➤ **Défini les éléments de notre méta-modèle de réseau de pétri**

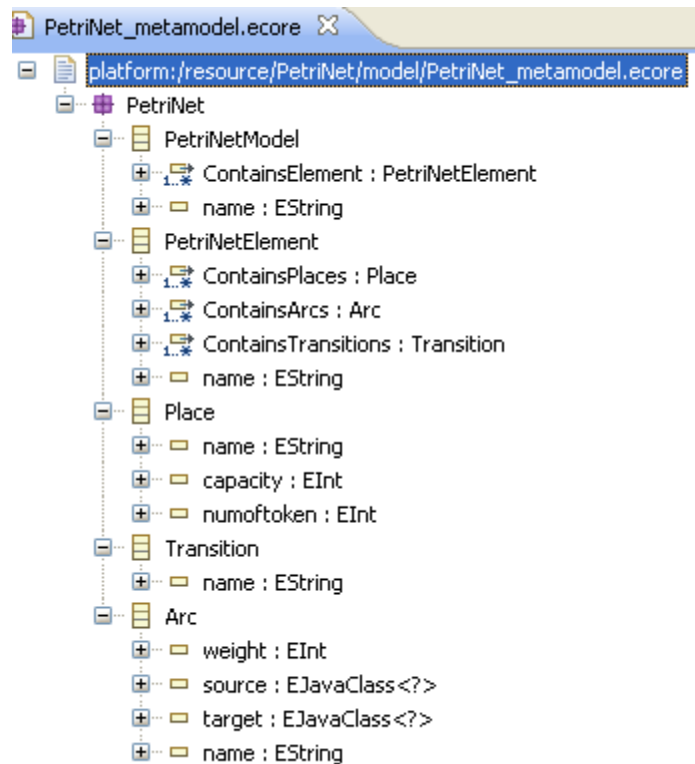


Figure 4.12 : les de méta-modèle de réseau de pétri

➤ **Défini Les éléments de génération de notre model réseau de pétri**

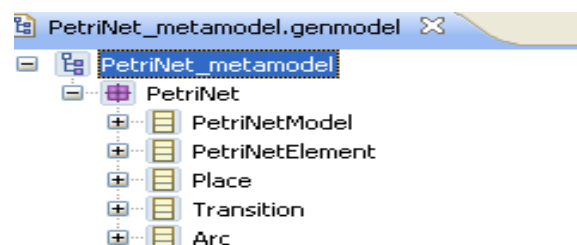


Figure 4.13 : les éléments genmodel de réseau de pétri

➤ **Généré les projets édit, editor, et tests**

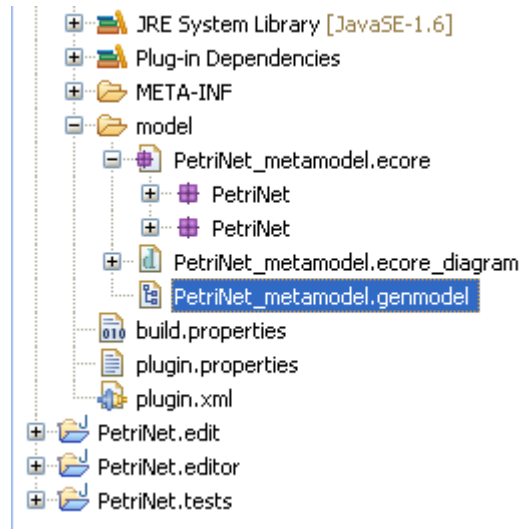


Figure 4.14 : génération des projets .edit, .editor, .tests

➤ **Exécuter les plugins :**

Installed Software Installation History Features Plug-ins Configuration				
Si...	Provider	Plug-in Name ▲	Version	Plug-in Id
	www.example.org	PetriNet_metamodel Edit ...	1.0.0	PetriNet.edit
	www.example.org	PetriNet_metamodel Editor	1.0.0	PetriNet.editor
	www.example.org	PetriNet_metamodel Model	1.0.0	PetriNet
	www.example.org	PetriNet_metamodel Tests	1.0.0	PetriNet.tests

Figure 4.15 : plugin détails de PetriNet modèle

➤ **Création d'une nouvelle modèle de réseau de pétri**

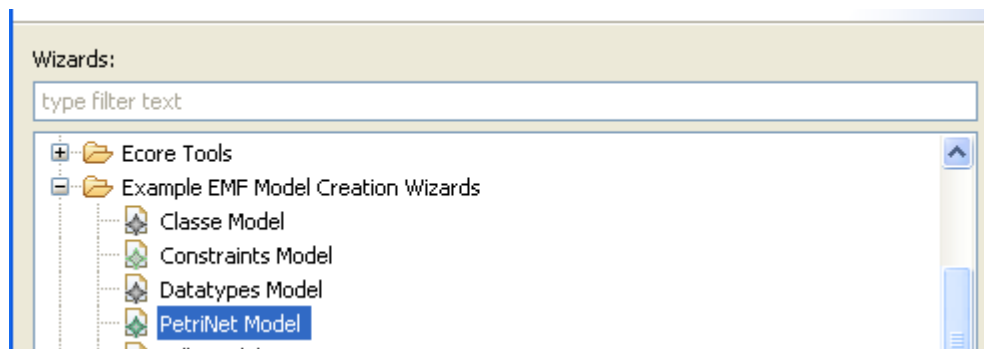


Figure 4.16 : création d'un réseau de pétri modèle

4.2.5 La génération d'un outil pour la transformation d'un DC vers Rdp

➤ **Création d'un ATL projet (File → New → Project...)**

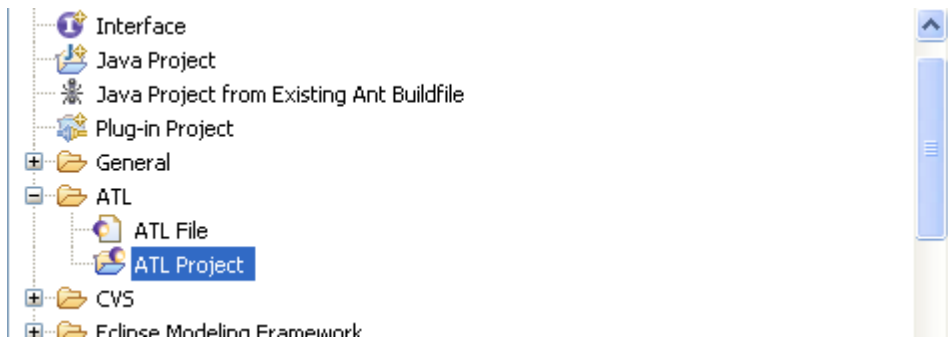


Figure 4.17 : créer un nouveau projet ATL

➤ **Création les dossiers (metamodels, models, transformations)**

Pour l'organisation de travail on va créer trois dossiers dans le projet ATL.

Clic droit sur le projet puis (New → Other ...).et choisi folder.



Figure 4.18 : création des dossiers

➤ **Création les méta-modèles Ecore**

Nous allons créer les méta-modèles de DC et de Rdp Ecore.

On a déjà vue comment faire ça.

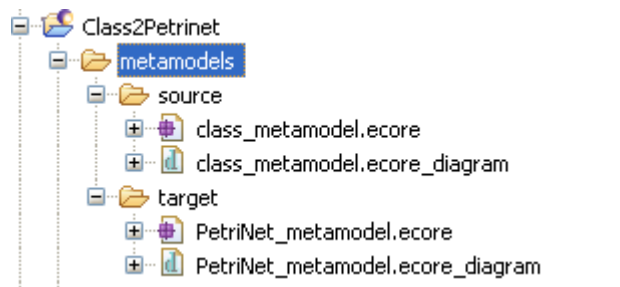


Figure 4.19 : création les méta-modèles

➤ **Génération de modèle conforme à méta-modèles de DC**

On va générer le modèle conforme à la méta-modèle DC.

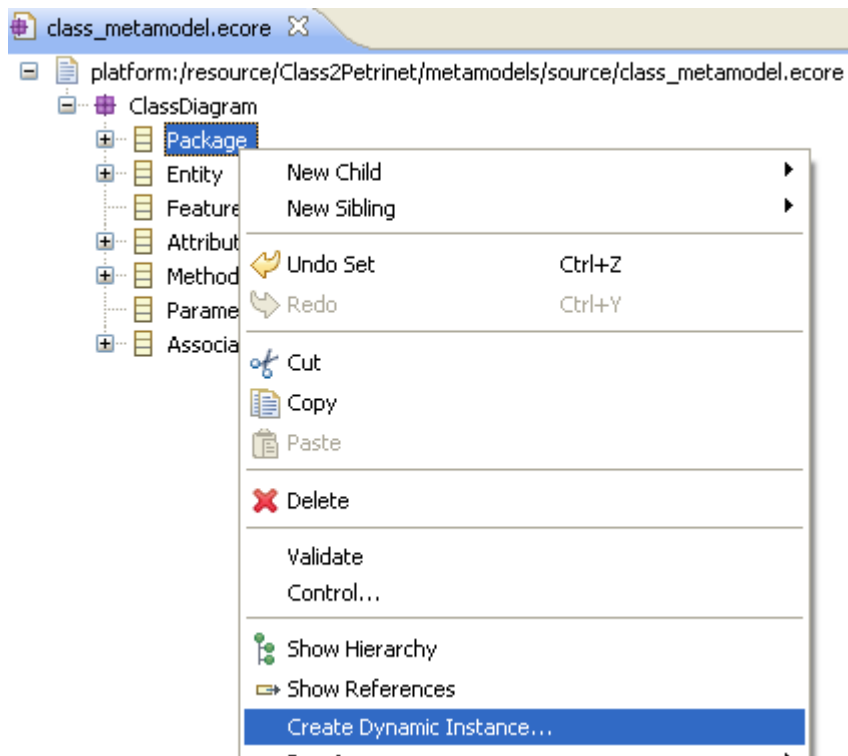


Figure 4.20 : création des instances dynamiques

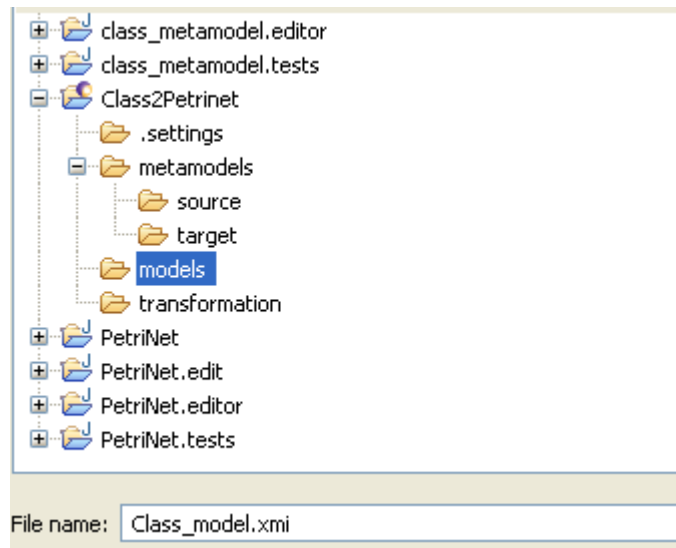


Figure 4.21 : choisir le nom de modèle et le dossier de l'emplacement



Figure 4.22 : modèles générés

➤ Création ATL file

Pour un ATL file on va clic droit sur le dossier transformations, On a créé dans notre projet ATL, et choisi (New → Other → ATL file...).

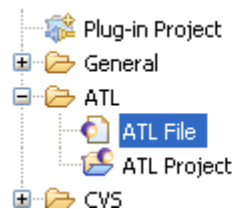
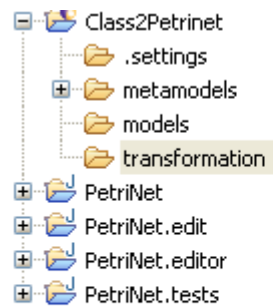


Figure 4.23 : création ATL file



File name: Myrules.atl

Figure 4.24 : sélection le nom d'ATL file

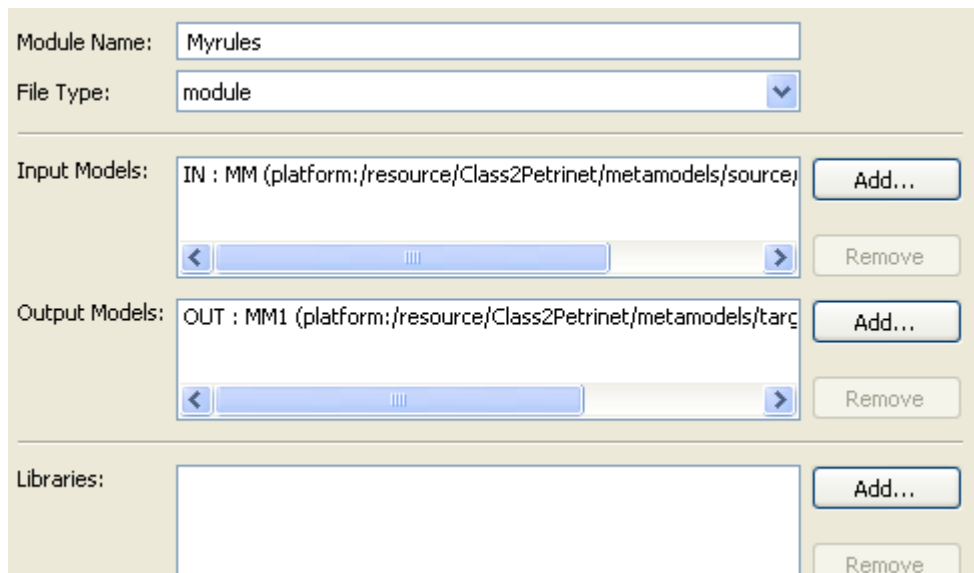


Figure 4.25 : sélection les méta-modèle source et cible

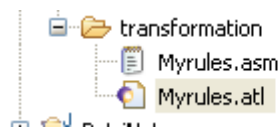


Figure 4.26 : ATL file généré

4.3 Transformation des diagrammes de classe vers les réseaux de Pétri

4.3.1 Les règles de transformation

Afin de produire les méta-modèles générés par notre outil (EMF), nous avons proposé des règles de transformation avec le langage ATL qui seront exécutées dans un ordre ascendant. L'application de ces règles au diagramme de classe créé par notre outil conduit à la génération du réseau de Pétri équivalent

✓ Règle 1 : transformation de package à PetriNetModel

Cette règle permet de transformer la classe package à la classe PetriNetModel, dans cette étape on va affecter les attributs de la classe package à des attributs de la classe PetriNetModel, ainsi les contenants classes de la classe package à la classe PetriNetModel.

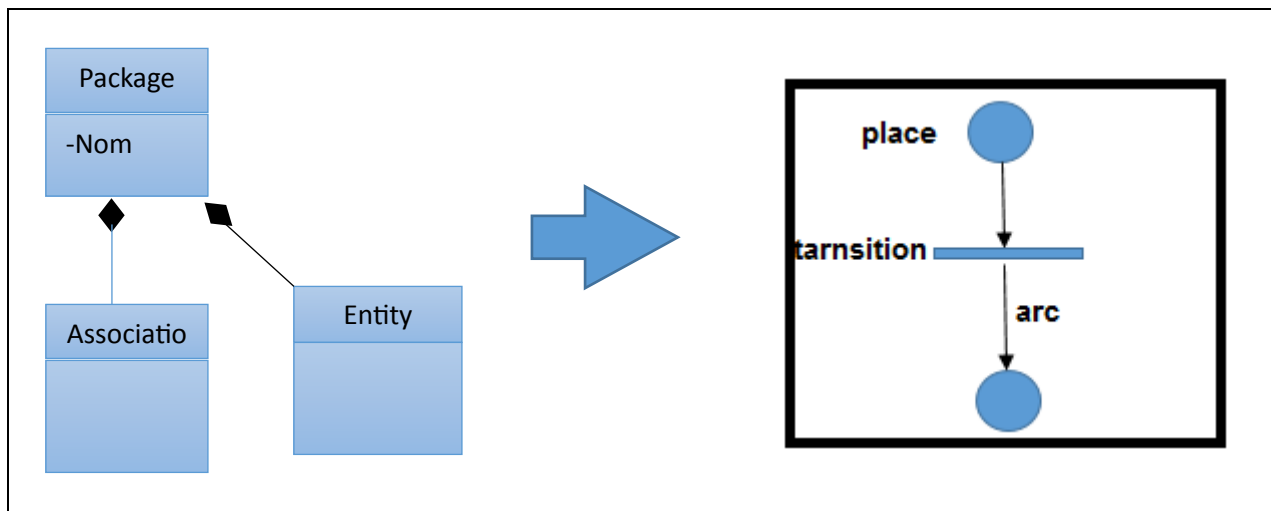


Figure 4.27 : transformation de Package vers PetriNetModel

✓ Règle 2 : transformation de Class à PetriNetElement

Cette règle permet de transformer la classe class à la classe PetriNetElement, donc faire une équivalence entre les classes class et PetriNet, on va transformer tous les éléments de la classe class qui correspondez à les attributs de PetriNet et les classes qui relient au. La figure suivant décrit cette règle.

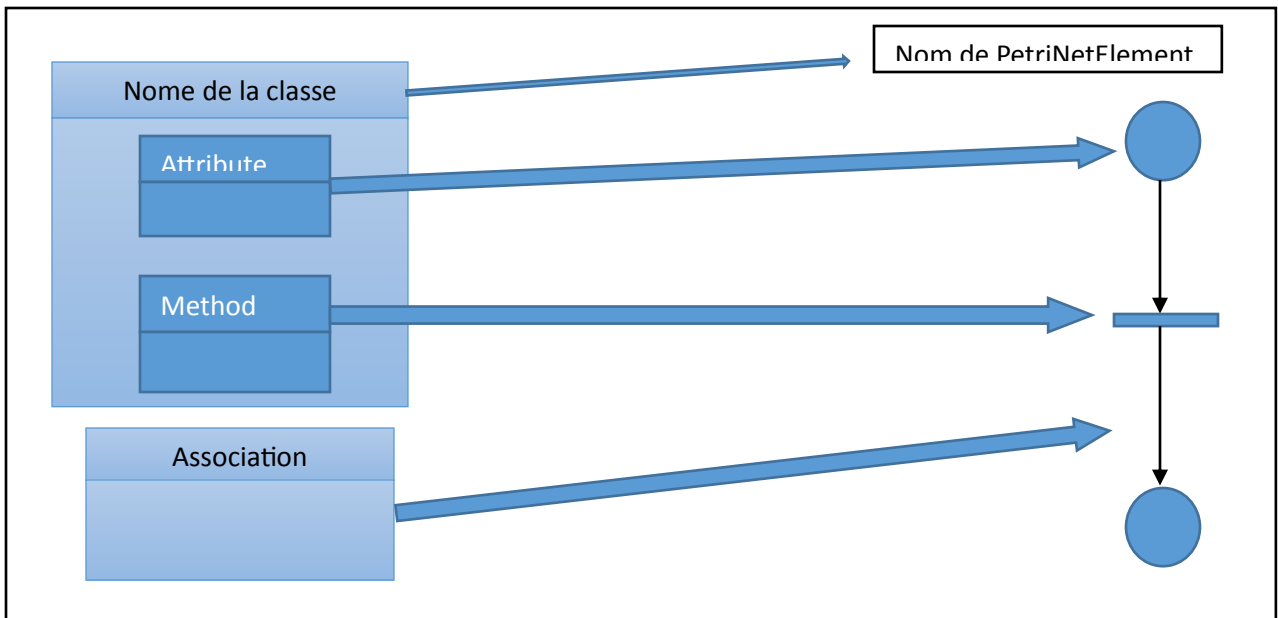


Figure 4.28 : transformation class vers PetriNetElement

✓ **Règle 03 : transformation attribut à place**

Dans cette règle l'équivalence est entre la classe Attribute et la classe Place, c'est –à-dire transformer tous les éléments de la classe Attribute à des éléments correspondant de la classe Place, et la figure 4.29 représenter cette règle.

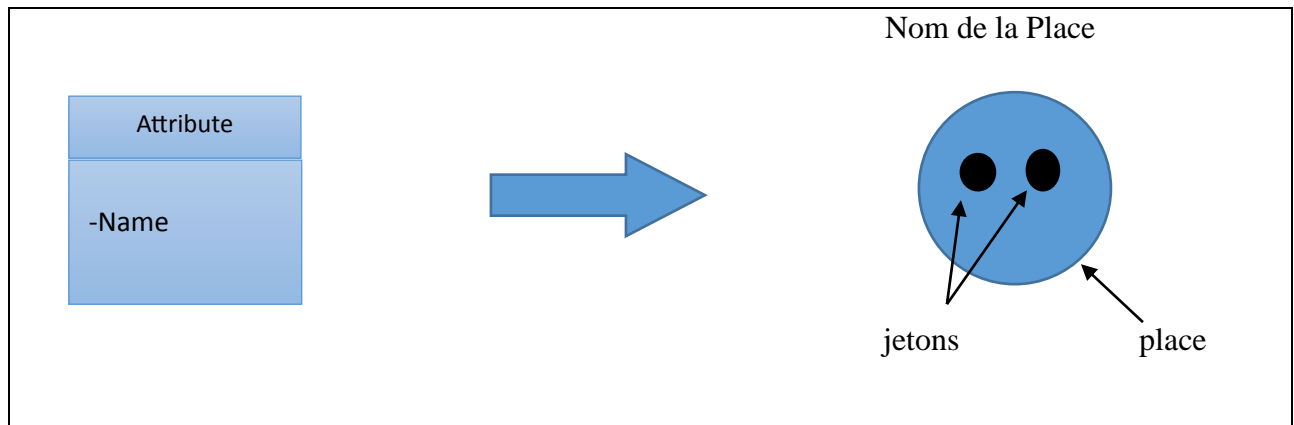


Figure 4.29 : transformation Attribute vers Place

✓ **Règle 04 : transformation Method à Transition**

Cette règle représente la transformation des éléments de la classe Method à des éléments de la classe Transition tel que le nom de la classe, la figure 4.30 représente cette règle.

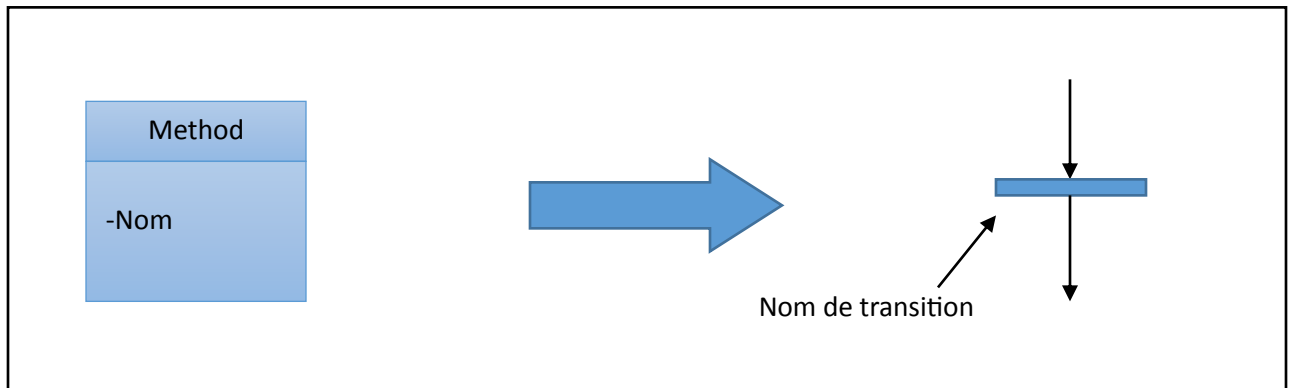


Figure 4.30 : transformation Method vers Transition

✓ **Règle 05 : transformation Association à Arcs**

Cette règle permet de transformer une association entre les classes Attribute et Method à un arc entre place et transition, donc on va transformer tous les éléments de la classe Association vers des éléments correspondant à la classe Arcs, la figure 4.31 représente la.

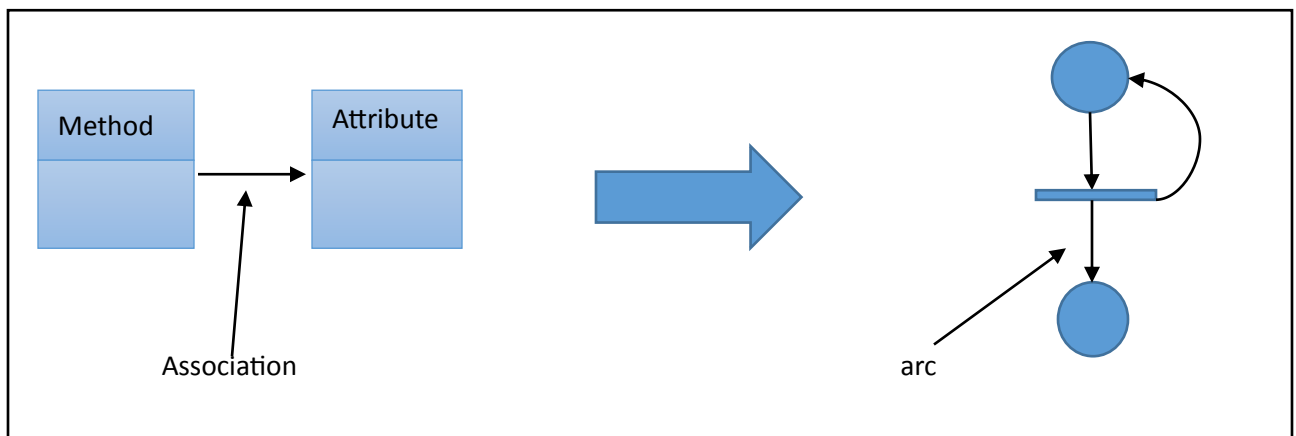


Figure 4.31 : Transformation Association vers Arcs

✓ **Règle 06 : Transformation Association à Transition**

Cette règle permet de transformer une association entre deux classes à une transition, donc l'association la fait une action entre les classe ou bien une appelle méthode de certain classe à une autre classe, la figure 4.32 représente la.

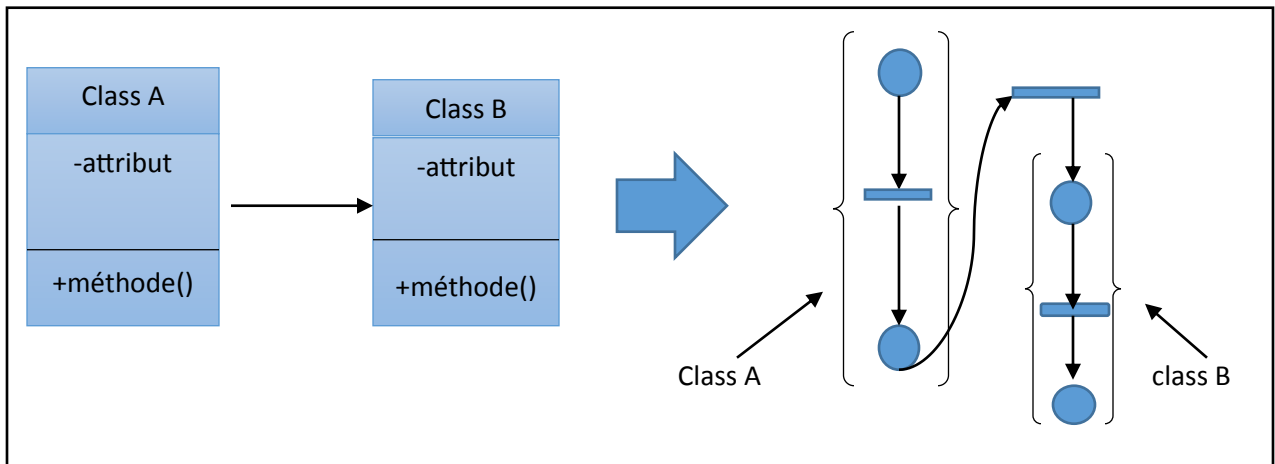


Figure 4.32 : transformation Association vers Transition

4.4 Exemple

On prouve l'utilité de notre démarche.

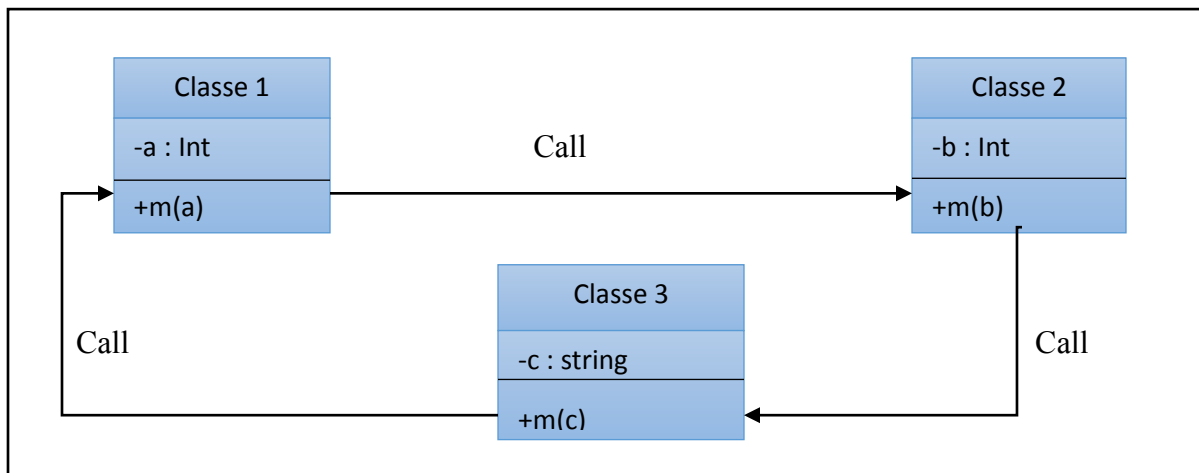


Figure 4.33 : exemple d'une instance de diagramme de classe

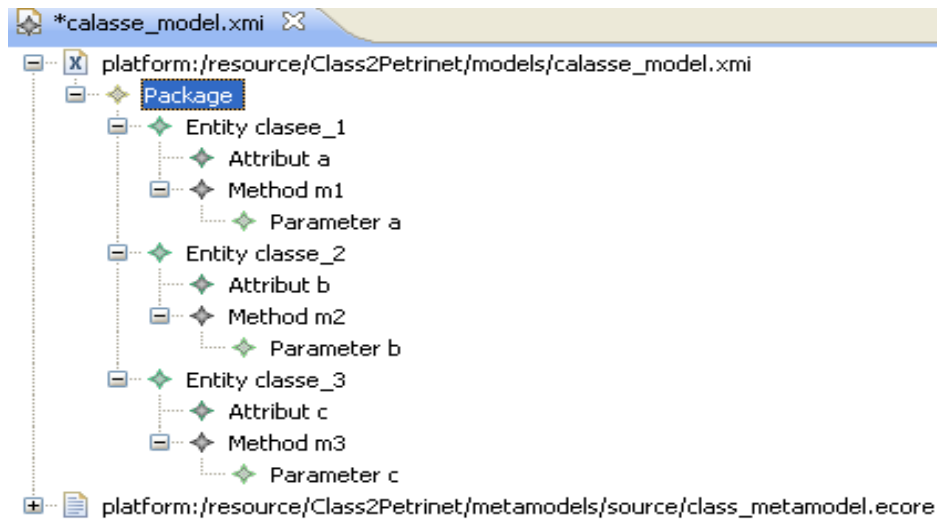


Figure 4.34 : Diagramme de classe simple par l'outil généré

Nous supposons que cet exemple est partie d'un diagramme de classe complet.

On a proposé une extension « call » qui représente une relation d'appel entre deux méthodes

Après avoir transformé ce diagramme de classe à réseau de Pétri on a obtenu le résultat suivant qui montre bien qu'il y aura un inter-blocage dans le système modélisé.

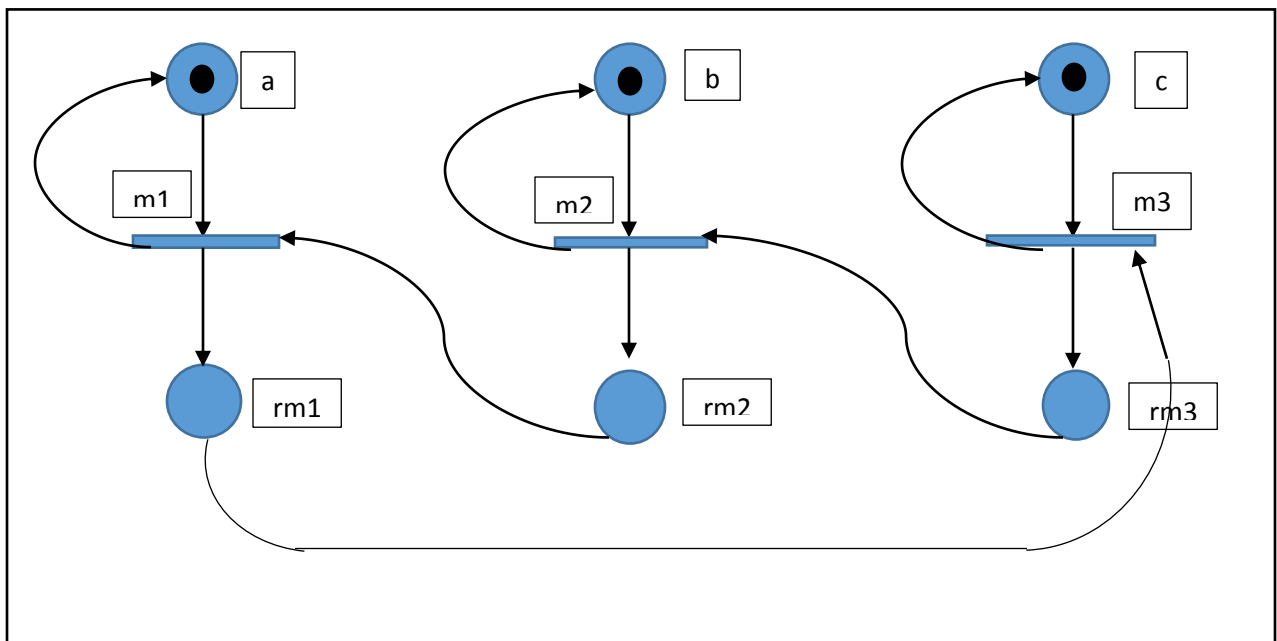


Figure 4.35 : réseau de Pétri généré pour montre un inter-blocage d' système modélisé

4.5 l'utilisation d'outil INA

Le but de transformer les processus métiers vers les RdPs est de pouvoir les analyser. Pour ce faire, nous avons choisi l'outil d'analyse INA (Integrated Net Analyser) présenté dans le chapitre 03

Un des plus grands avantages des RdPs est leur représentation graphique des modèles.

Cependant, l'outil INA n'est pas graphique ce qui nous allons faire une représentation textuelle qui peut être décrire le modèle de réseau de pétri que nous avons généré.

```

Fichier  Edition  Format  Affichage  ?
P    M    PRE,POST  NETZ  0:StateEquation
  1  1    1, 3
  2  1    2, 1
  3  1    3, 2

@
place nr.                name capacity time
    1: rm1                oo      0
    2: rm2                oo      0
    3: rm3                oo      0

@
trans nr.                name priority time
    1: m1                 0      0
    2: m2                 0      0
    3: m3                 0      0

@
|

```

Figure 4.36 : exemple INA fichier .pnt

4.5.1 le resultat obtenu

Après l'exécution l'outil INA sur notre modèle généré, nous avons obtenu le resultat suivant :

```
C:\Users\aimade\AppData\Local\Temp\yZip\yZip10256\yZip28347\INAwins32.exe
```

```
>>>>>>>>> Welcome to the Integrated Net Analyzer! <<<<<<<<<<<  
Version 2.2                               Jul 31 2003           Peter Starke, Berlin  
  
Current net options are:  
token type: black              (for Place/Transition nets)  
time option: no times  
fixing rule: normal  
priorities : not to be used  
strategy   : single transitions  
line length: 80  
  
Do You want to  
edit ? .....E  
fine ? .....F  
analyse ? .....A  
reduce ? .....R  
read the session report ? .....S  
delete the session report ? .....D  
change options ? .....O  
quit ? .....Q  
choice > _
```

Figure 4.37 : les choix d'exécution

```

urrent name options are:
  transition names to be written
  place names to be written
.....Reset options? Y/N N
e net is statically conflict-free.
e net is dynamically conflict-free.
e net is pure.
e net is ordinary.
e net is homogenous.
e net is conservative.
e net is structurally bounded.
e net is bounded.
ere are no proper semipositive T-surinvariants.
e net is subconservative.
e net is a state machine.
e net is extended free choice.
e net is extended simple.
e net is free choice.
e net is marked.
e net is not marked with exactly one token.
e net is a marked graph.
e net has a non-blocking multiplicity.
e net has no nonempty clean trap.
e net has no transitions without pre-place.
e net has no transitions without post-place.

```

Figure 4.38 : resultat représente le conflit de notre réseau

4.6 Conclusion

Dans ce chapitre nous avons proposé une approche automatique pour transformer les Diagrammes de classe vers les Réseaux de Pétri. La méthode proposée se base sur langage de transformation ATL et utilise l'outil de modélisation et de méta-modélisation EMF.

Afin de réaliser cette méthode, nous avons proposé un méta-modèle (ecore EMF) qui nous a permis de générer un outil visuel pour la réalisation (écriture) des Diagrammes de classe et des Réseau de Pétri.

Enfin, nous avons donné un exemple très riche pour illustrer notre démarche, et montre le probleme de conflit à partir de l'outil d'analyse INA.