***REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE***

***Ministère de l'Enseignement Supérieur et de la Recherche Scientifique***

***Université El Hadj Lakhdar – BATNA***

**Faculté des Sciences**

**Département d'Informatique**

N° d'ordre :. . .. . .. . .

Série :. . .. . .. . .

**Mémoire**

**Présenté en vue de l'obtention du diplôme de**

## Magister en Informatique

Option : **Système d'Information et de Connaissance *(SIC)***

Sujet du mémoire :

# Une Approche de Transformation des Diagrammes d'Activités d'UML Mobile 2.0 vers les Réseaux de Petri

Présenté le / /

Par : **GUERROUF FAYÇAL**

Composition du jury :

| | | |
|---|---|---|
| Mr. BELATTAR Brahim | **Président** | (Maître de Conférence à l'Université de Batna) |
| Mr. CHAOUI Allaoua | **Rapporteur** | (Maître de Conférence à l'Université de Constantine) |
| Mr. BILAMI Azzeddine | **Examinateur** | (Maître de Conférence à l'Université de Batna). |
| Mr. KAZAR Okba | **Examinateur** | (Maître de Conférence à l'Université de Biskra). |

*À mes très chers parents avec tout mon amour et ma gratitude...*
*À mes frères et mes sœurs...*
*Je dédie ce travail*

***Remerciements***

***Je tiens à remercier :***

*Mr. Chaoui Allaoua d'avoir accepté de m'encadrer, de gérer ce travail et pour ces précieux conseils.*

***Je remercie également les membres du jury :***

*Mr.OKBA Kazar Maître de conférences à l'université de Biskra*
*Mr. BELATTAR Brahim Maître de Conférence à l'Université de Batna*
*Mr. BILAMI Azzeddine Maître de Conférence à l'Université de Batna*

*D'avoir accepté l'évaluation de ce travail.*

*J'exprime ma profonde reconnaissance à tous ceux qui m'ont aidé à l'élaboration de ce mémoire de près ou de loin.*

# Table des matières

# Table des figures

# Liste des tableaux

# Introduction Générale

Le progrés technologique dans le domaine des réseaux informatiques et la disponibilité du matériel performant ont rendu possible l'utilisation d'une catégorie spéciale de programmes appelés *"agent mobile"*. Ils ont les capacités des agents intelligents et les caractéristiques des applications distribuées. Un agent mobile peut décider comment atteindre son but et de migrer d'un nœud dans le réseau vers un autre pour compléter ses objectifs. Cela lui permet d'être autonomes et d'aller où existe ses besoins. Ils sont utilisés dans de divers domaines tels que le commerce électronique, la gestion des réseaux et la recherche d'information.

Les caractéristiques qui ont fait des agents mobiles un outil très puissant, ont engendré des problèmes de modélisation et de vérification. Cependant, pour modéliser et vérifier les agents mobiles, de nouveaux techniques et outils doivent être créés.

Dans le domaine du génie logiciel, il existe deux approches pour représenter les modèles des systèmes. La première approche et traditionnelle est l'approche informelle. Elle utilise une notation graphique et commune pour la représentation. Elle est plus facile à comprendre et à communiquer. La deuxième et importante approche pour spécifier les modèles des systèmes est l'approche formelle. Elle repose sur de solides notations et de preuves mathématiques.

UML (Unified Modeling Language) Le langage de modélisation unifié est considéré comme le langage standard de modélisation visuelle utilisé pour spécifier, visualiser, construire et documenter les artefacts d'un système logiciel. Ainsi, pour être en mesure de modéliser les agents mobiles, l'extension diagramme d'activités mobile a été proposé pour capturer les spécificités dynamiques des agents mobiles. Elle est définie par l'utilisation des mécanismes d'extension d'UML tels que les stéréotypes.

Malgré qu'UML est un langage riche, doté d'une notation ouverte et largement utilisé, les modèles UML restent toujours en besoin d'être vérifiés pour assurer que le comportement spécifié dans ces modèles est correcte et que ce comportement répond exactement aux besoins fonctionnels du système. Ce fait est dû à la nature graphique et semi-formelle du langage UML. Ainsi à sa sémantique qui n'est pas formellement spécifiée.

Par contre, les réseaux de Petri sont un langage de modélisation formel et graphique. Ils sont un outil très puissant pour l'analyse et la vérification. De même que UML, des extensions ont été proposées pour pouvoir les utiliser comme langage de modélisation

des agents mobiles. Les réseaux de Petri imbriqués *(Nested Petri Net)* représentent une extension adéquate pour modéliser les agents mobiles. Les jetons dans ces réseaux de Petri peuvent être des réseaux de Petri eux-mêmes.

Dans ce travail, nous proposons une approche de transformation des diagrammes d'activités mobiles vers les réseaux de Petri imbriqués. L'approche que nous définissons vise à proposer :

1. Un outil de modélisations des diagrammes d'activités mobiles et des réseaux de Petri imbriqués.
2. Une grammaire de graphe pour appliquer la transformation des diagrammes d'activités mobiles vers des réseaux de Petri imbriqués.

La construction d'un outil de modélisation à partir de zéro est une tâche prohibitive. Les approches de méta-modélisations sont utiles pour faire face à ce problème, car elles permettent la modélisation des formalismes eux-mêmes. Un modèle de formalisme qui contient suffisamment d'informations permet la génération automatique d'un outil pour construire des modèles conforme à la syntaxe du formalisme décrit. Pour cela, nous utilisons $AToM^3$ (A Tool for Multi-formalism and Meta-Modelling) qui permet de réaliser ces concepts.

Le reste de ce mémoire est organisé comme suit :

Dans le premier chapitre nous présenterons quelques concepts sur la mobilité, en particulier les agents mobiles. Puis nous verrons brièvement UML et nous terminerons le chapitre par une présentation des diagrammes d'activité mobile.

Le deuxième chapitre sera consacré aux réseaux de Petri. Nous commencerons le chapitre par l'étude des réseaux de Petri ordinaires puis nous présenterons les réseaux de Petri imbriqués.

Dans le troisième chapitre nous commencerons par la présentation des concepts de transformation. Puis nous présenterons l'outil à utiliser, $AToM^3$, qui est un outil de modélisation multi-formalisme et multi-paradigme. Apres, nous détaillerons notre approche de transformation des diagrammes d'activités mobiles vers les réseaux de Petri imbriqués.

Le présent mémoire se termine par une conclusion générale qui résume ce que nous avons réalisé dans notre contribution ainsi que les travaux futurs, inspirés de l'approche proposée

# Chapitre 1

# UML & La Mobilité

L'avènement de la technologie de réseau informatique a incité les chercheurs à développer de nouveaux paradigmes de communication dont les processus peuvent se déplacer sur le réseau. Le développement des logiciels à base de ces paradigmes nécessite des extensions appropriées des méthodes et des concepts traditionnels.

UML [1] *(Unified Modeling Language),* le langage de modélisation unifié, fournit des mécanismes d'extension. Ces derniers permettent d'adapter les spécifications UML à des exigences d'un domaine spécifique. Ainsi, l'extension UML mobile a été proposée pour satisfaire les besoins en modélisation de la mobilité.

Dans ce chapitre, nous rappellerons quelques définitions et quelques concepts sur la mobilité et UML 2.0 puis nous verrons une extension de diagramme d'activités d'UML 2.0 permettant de modéliser un des paradigmes de code mobile.

## 1.1 Paradigmes de Code Mobile

La mobilité de code peut être définie, d'une manière informelle, comme la capacité de changer dynamiquement la relation entre les fragments de code et l'endroit où elles sont exécutées [2].

Alfonso Fuggetta et Giovanni Vigna [2] ont identifié trois principaux paradigmes exploitant la mobilité de code :

1. Évaluation à distance.
2. Code à la demande.
3. Agent mobile.

Ces paradigmes sont caractérisés par [2] :

1. La localisation des composantes *(Code et Ressources)* avant et après l'exécution de code.
2. L'unité d'exécution chargée d'exécuter le code.
3. La localisation où l'exécution de code se déroule en réalité.

Dans le cadre de notre étude, on s'intéresse aux agents mobiles. Ainsi, seulement ces derniers seront vus en détail.

### 1.1.1 Évaluation à Distance

Dans le paradigme d'évaluation à distance, on trouve des serveurs qui exécutent des codes provenant des clients. Après l'exécution, un résultat est retourné au client [2].

Un exemple de ce type de paradigme est celui des interactions des imprimantes *PostScript*. Le code d'une requête *SQL* (Structured Query Language) émis vers un serveur de base de données représente un autre exemple d'évaluation à distance.

La figure 1.1 représente le schéma de l'évaluation à distance

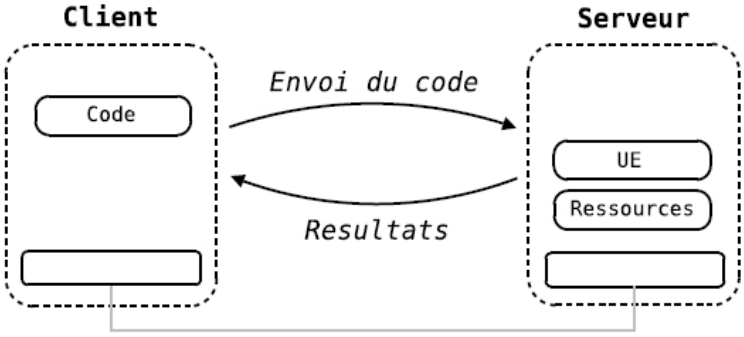



FIGURE 1.1 – Évaluation distante

### 1.1.2 Code à la Demande

Le client dispose de l'unité d'exécution et des ressources mais pas du code nécessaire à la réalisation de service. Ce dernier va être récupéré auprès du serveur. Il s'agit donc de l'inverse du cas précédent. Ainsi, un client adresse une requête uniquement pour récupérer un code précis afin de l'exécuter localement avec ces ressources locale [2].

Cette méthode permet d'étendre les fonctionnalités d'une application directement chez le client sans avoir besoin d'effectuer une nouvelle installation. L'exemple le plus répandu de l'utilisation de cette méthode est le téléchargement d'applet Java [1] à partir d'un serveur Web. Ces applets sont des classes Java présentes sur le site serveur, ils vont être téléchargées d'abords, puis interpréter par la machine virtuelle du site client [2].

La figure 1.2 représente le schéma de code à la demande.

### 1.1.3 Agent Mobile

Les agents mobiles regroupent deux concepts, le premier est celui d'agent issu du monde d'intelligence artificielle, le deuxième est celui de la migration du processus issue des systèmes répartis [3].

1. http ://java.com/

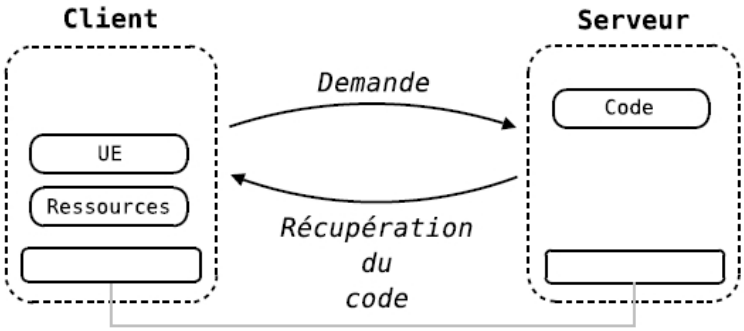



FIGURE 1.2 – Code à la demande

Un agent est, généralement, décrit comme une entité persistante avec un certain degré d'indépendance ou d'autonomie qui effectue un ensemble d'opérations en fonction de ce qu'il perçoit. Un agent est doté, habituellement, d'un certain niveau d'intelligence, comme il doit avoir connaissance de ses buts et ses désirs.

Plusieurs définitions on été donnée aux agents intelligents. Ferber [4] définit un agent comme une entité autonome, réelle ou abstraite, capable d'agir sur elle même et sur son environnement. Un agent dans un univers multi-agents, peut communiquer avec d'autres agents et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents.

Pour Jennings, Sycara et Wooldridge [5], un agent est un système informatique, situé dans un environnement et agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu.

#### 1.1.3.1 Migration d'Agent

La migration est un mécanisme qui vise à transférer un agent d'un site sur un autre pour poursuivre son exécution [2].

Deux types de mécanismes de migration ont été proposés dans les environnements d'agents mobiles ; La migration faible et la migration forte.

***Migration Forte :*** Elle permet la migration du code et de l'état d'exécution en même temps, ce qui permet à un agent de continuer son exécution du point où il a été arrêté pour migrer.

***Migration Faible (clonage) :*** Elle ne permet à un agent que la migration de son code et ses données. A chaque migration l'agent doit s'exécuter du début.

#### 1.1.3.2 Communication entre Agents

Les agents sont créés pour être capables d'atteindre des objectifs qui sont difficiles en agissant individuellement .

Habituellement, un agent coopère avec d'autres agents, il doit donc avoir des capacités sociales et communicatives [6]. Pour communiquer, les agents doivent être en mesure de :

- ❑ ***Envoyer et recevoir des messages :*** À ce niveau, les agents doivent communiquer sur des normes physiques et les couches du réseau pour être en mesure d'envoyer et de recevoir des chaînes de caractères ou des objets qui représentent des messages.
- ❑ ***Analyser les messages :*** Au niveau syntaxique, les agents doivent être en mesure d'analyser les messages, pour le décoder correctement en ses parties, tels que le contenu du message, la langue, l'expéditeur et doit aussi être capable d'analyser son contenu.
- ❑ ***Comprendre les messages :*** Au niveau sémantique, les messages analysés doivent être compris de la même manière, c'est à dire, l'ontologie décrivant les symboles doivent être partagée ou explicitement exprimée et accessible pour être en mesure de décoder les informations contenues dans le message

La communication entre agents peut être de deux types. Elle peut être centralisée en s'effectuant par partage d'information via un tableau noir ou décentralisée par envoi de message

***(a) Communication par partage d'information :*** Ce type de communication s'effectue via un tableau noir. Ce dernier fournit un milieu pour échanger les données, qui sont écrites dessus par les agents participants. N'importe quel agent peut obtenir l'accès aux informations et aux messages signalés sur le tableau noir.

***(b) Communication par envoi de messages :*** Dans ce type de communication, les agents sont liés directement et un message est envoyé directement au destinataire. La communication se fait, soit en mode point à point, soit en mode par diffusion.

1. *mode point à point* : l'agent émetteur du message connaît et précise l'adresse de ou des agent(s) destinataire(s).
2. *mode par diffusion* : le message est envoyé à tous les agents du système.

#### 1.1.3.3 Langages de Communication entre Agents (*ACL*)

Les deux langages les plus utilisés, pour la communication entre Agents sont KQML (Knowledge Query and Manipulation Language) et FIPA [2]-ACL [3], qui sont très similaire. il sont fondé sur la même théorie des actes de langage [7].Ils considèrent que les messages échangés sont des actes communicatifs. Les spécifications de FIPA-ACL se composent d'un ensemble de types de message et de la description. Un message comprend plusieurs éléments, comme l'illustre l'exemple suivant :

2. Organisation Foundation for Intelligent Physical Agents (FIPA) est une des normes *"IEEE Computer Society organisation"* qui promeut la technologie à base d'agents et de l'interopérabilité de ses normes avec d'autres technologies.
3. Agent Communication Languag

```
(inform
      :sender agent1
      :receiver hpl-auction-server
      :content (price (bid good02) 150)
      :in-reply-to round-4
      :reply-with bid04
      :language sl
      :ontology hpl-auction
)
```

Le message ACL contient plusieurs paramètres. Le seul paramètre obligatoire est performatif (acte de communication - tel que informer dans l'exemple précédent), mais la plupart des messages ACL contiennent également un expéditeur, un destinataire et des paramètres de contenu. Comme on peut le voir, le message ACL est indépendant du contenu.

Il existe plusieurs implémentations de la norme FIPA. Probablement l'implémentation la plus populaire est *"Jade*[4]*"* ( Java Agent Development Framework ).

Après avoir évoqué brièvement la mobilité nous verrons dans la suite de ce chapitre comment peut-on modéliser les agents mobiles avec UML.

## 1.2 Langage de Modélisation Unifié (UML)

Le langage de modélisation unifié (UML) est un langage de modélisation visuelle, utilisé pour spécifier, visualiser, construire et documenter les artefacts d'un système logiciel [8].

Il est utilisé pour comprendre, concevoir, naviguer, configurer, maintenir et contrôler les informations sur les systèmes modélisés [9].

UML est un condensé de trois notations importantes *(Booch, OMT*[5]*, OOSE*[6]*)*. Il a eu un impact indéniable sur la façon dont nous considérons le développement des systèmes [10].

La première version d'UML, publiée par l'OMG[7] été *(UML 1.0)* en 1997, puis la version d'UML 1.x *(UML 1.1, UML 1.4 UML 1.5)* jusqu'à 2004 avec la sortie de la version *UML 2.0.*

### 1.2.1 UML et Méta-Modélisation

UML est basé sur l'architecture de méta-modélisation à quatre niveaux. Chaque niveau successif est marqué de *M3* à *M0* et sont, habituellement, nommés méta-méta-modèle,

4. http ://jade.tilab.com/
5. object-modeling technique
6. Object Oriented Software Engineering
7. http ://www.omg.org/, Object Management Group : un consortium d'industrie informatique internationale , ouverte, sans but lucratif.

méta-modèle, diagramme de classes et diagramme d'objets respectivement [1].

Un diagramme de niveau, *Mi,* est une instance d'un diagramme de niveau *Mi +1*. Le diagramme de niveau *M3* est utilisé pour définir la structure d'un méta-modèle. Le Meta Object Facility (*MOF*) appartient à ce niveau. Le méta-modèle d'UML appartient au niveau *M2* [1]. UML est, formellement, défini en utilisant un méta-modèle qui est, lui-même, exprimé en UML [9].

La figure 1.3 illustre cette définition

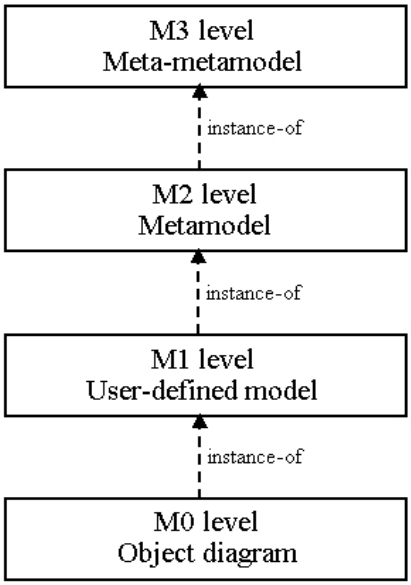



FIGURE 1.3 – Architecture de méta-modélisation à quatre niveaux

Nous verrons le concept de méta modélisation en détail dans la section 3.1 du chapitre 3.

### 1.2.2 Diagrammes d'UML 2.0

La notation UML est décrite sous forme d'un ensemble de diagrammes. La spécification de la version d'UML 2.0 définit 13 diagrammes regroupés dans deux classes principales qui sont [1] :

1. Les diagrammes statiques
2. Les diagrammes comportementaux

#### 1.2.2.1 Diagrammes statiques

Les diagrammes statiques (structurels) illustrent les caractéristiques statiques d'un modèle.

Les diagrammes statiques regroupent [10]

1. les diagrammes de classes,
2. les diagrammes d'objets,

3. les diagrammes de structure composite,
4. les diagrammes de composants,
5. les diagrammes de déploiement,
6. les diagrammes de paquetages.

#### 1.2.2.2 Diagrammes dynamiques

Les diagrammes comportementaux décrivent la façon dont les ressources modélisées dans les diagrammes structurels interagissent.

Les diagrammes dynamiques regroupent [10] :

1. les diagrammes de cas d'utilisation,
2. diagrammes d'activités,
3. les diagrammes d'états-transitions,
4. les diagrammes d'interactions
   (a) Diagramme de séquence
   (b) Diagramme de communication
   (c) Diagramme global d'interaction
   (d) Diagramme de temps

Dans notre projet on s'intéresse seulement au diagramme d'activités, pour cela la section suivante sera consacrée au diagramme d'activités.

### 1.2.3 Diagrammes d'Activités

Les diagrammes d'activités permettent de spécifier la façon dont le système atteint ses objectifs. Ils décrivent un flux séquentiel d'actions pour représenter un processus se produisant dans le système modélisé. Par exemple, on peut utiliser un diagramme d'activités pour modéliser les étapes de la création ou de gestion d'un compte de blog [8].

La figure 1.4 représente un diagramme d'activité d'un processus de gestion des comptes.

#### 1.2.3.1 Notation

Nous présentons dans cette sous section seulement quelques éléments de modélisation de base des diagrammes d'activité [1], pour plus de détail voir [1, 10, 11]

1. ***Action*** **:** Une action est l'unité fondamentale de la spécification de comportement. L'action prend un ensemble d'entrées et les convertit en un ensemble de résultats, l'une ou les deux ensembles peuvent être vide. Une action ne peut être décomposée en actions plus simples.

   Une action peut être, par exemple :

   ❑ une affectation de valeur à des attributs

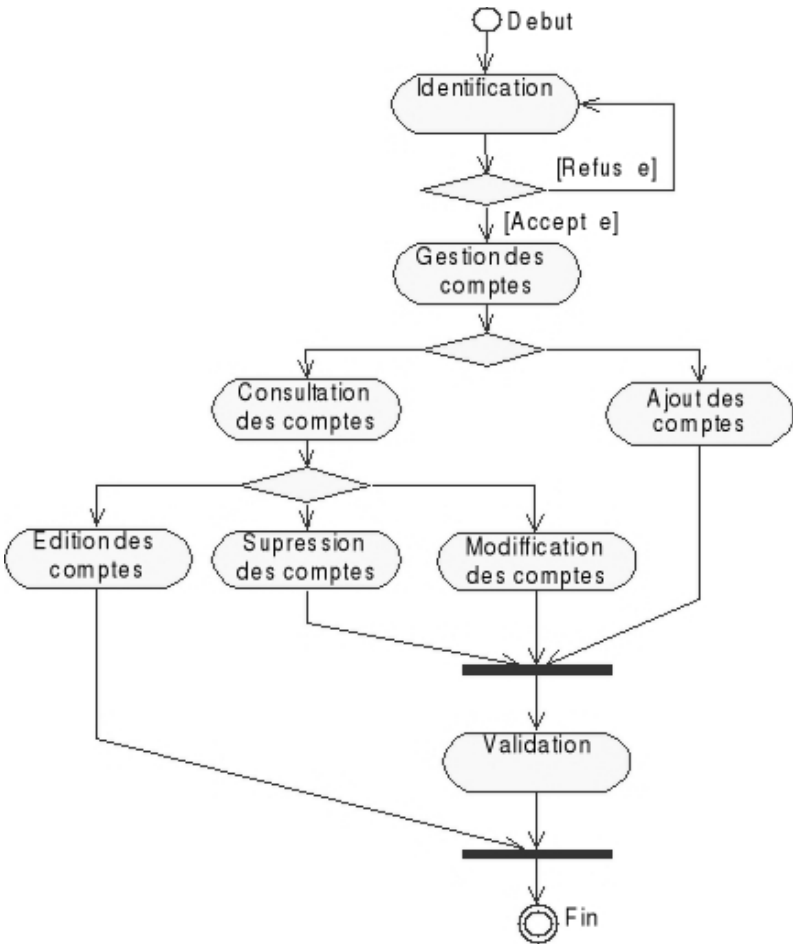



FIGURE 1.4 – Exemple de diagramme d'activité

- la création d'un nouvel objet ou lien
- l'émission d'un signal
- la réception d'un signal
- etc.

La notation d'une action est un rectangle avec des coins arrondis. La figure 1.5 montre trois exemples d'actions.

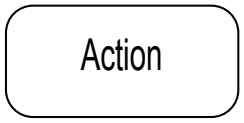



FIGURE 1.5 – Notation d'action

2. ***Activité*** **:** Une activité contient des séquences d'actions et / ou d'autres activités. On utilise les activités pour grouper des séquences d'actions ensemble.
3. ***Nœud de contrôle*** **:** On utilise les nœuds de contrôle pour guider le flux de contrôle (et le flux d'objets) à travers un groupe d'activités et d'actions. Les nœuds de contrôle viennent dans une variété de formes, en fonction de ce qu'on a besoin, ils servent comme un agent de circulation pour le flux de contrôle et les flux d'objets. Les nœuds de contrôle sont les suivants :

- ***Initial*** **:** Un nœud initial est l'endroit où le flux de contrôle commence quand une activité est invoquée. La Figure **1.6** montre la notation d'un nœud initial.

FIGURE 1.6 – Notation de nœud initial

- ***Final*** **:** Un nœud final est un nœud de contrôle dans laquelle un ou plusieurs flux au sein d'une activité donnée s'arrêtent. Il existe deux types de nœuds finaux :
  (a) nœud de flux final.
  (b) nœud d'activité final.

(a) Nœud d'activité final

(b) Nœud de flux final

FIGURE 1.7 – Notation de noeud final

La figure 1.7a et 1.7b montre la notation de nœud final.

- ***Décision*** **:** offre un choix entre deux ou plusieurs activités sortantes, dont chacune a une expression booléenne qui doit résoudre à *Vrai* avant la prise de chemin. Un nœud de décision apparaît comme un diamant, comme le montre la figure1.8

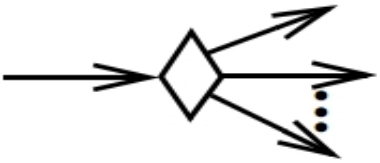


FIGURE 1.8 – Notation de noeud de décision

- ***Fusion (Merge)*** **:** Regroupe des flux multiples.
- ***Bifurcation*** **:** Divise un flux en plusieurs flux simultanés. La figure1.9 montre sa notation.

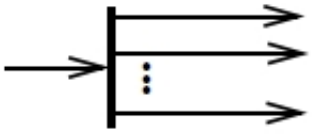


FIGURE 1.9 – Notation de noeud de bifurcation

La figure 1.10 illustre l'utilisation des notations des nœuds de contrôle

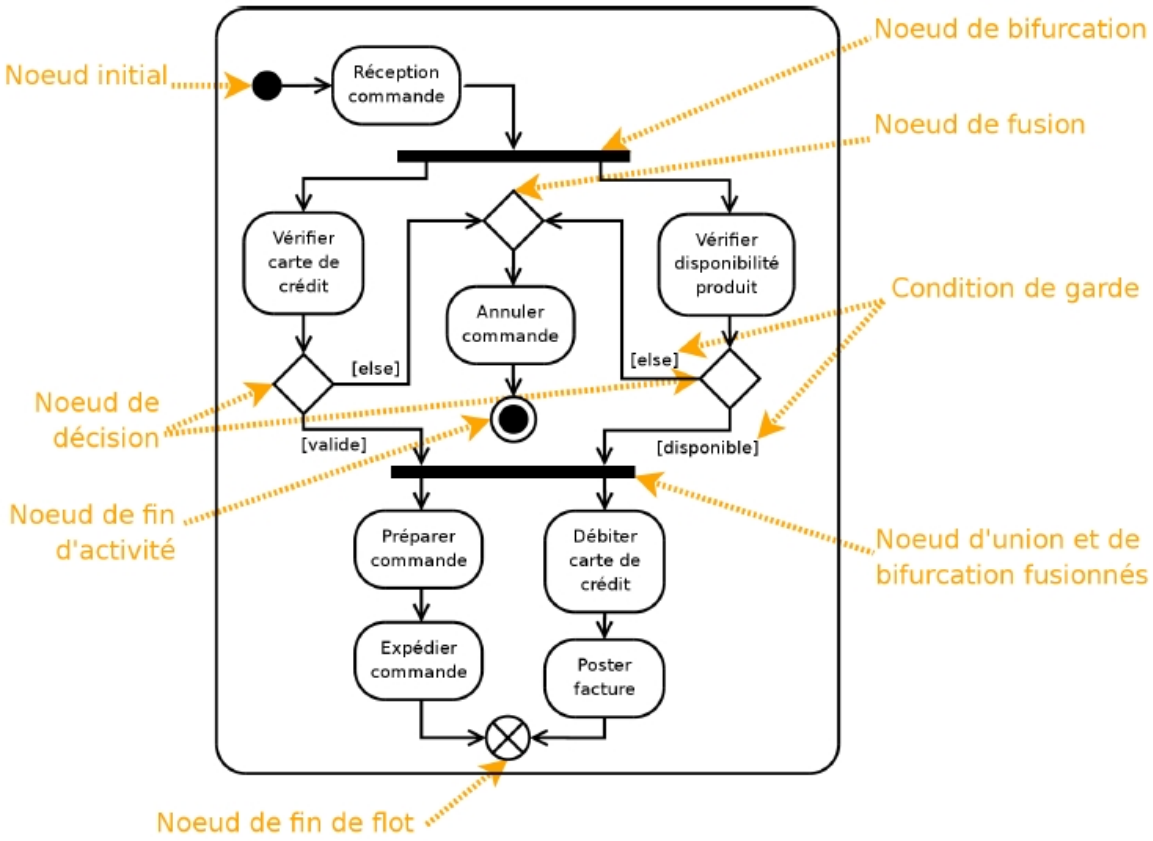



FIGURE 1.10 – Exemple de nœuds de contrôle

4. **Noeud exécutable :** Un nœud exécutable est un nœud d'activité qui peut être exécuté. Un nœud exécutable possède un ou plusieurs gestionnaires d'exception. Chaque gestionnaire spécifie un nœud exécutable à exécuter en cas d'une exception produite pendant l'exécution d'un autre nœud exécutable.

   La figure 1.11 représente un exemple d'une exception.

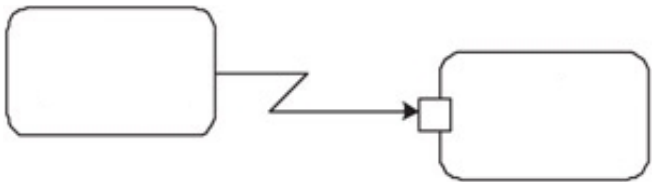


FIGURE 1.11 – Exemple d'exception

5. **Nœud d'objet :** Un nœud d'objet apparaît généralement comme un rectangle avec son nom à l'intérieur. La figure 1.12 montre un exemple de sa notation.

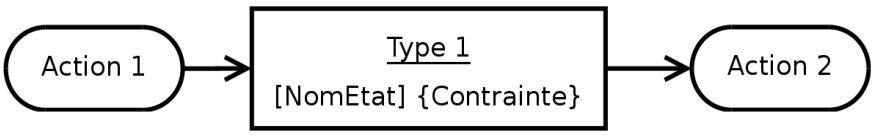



FIGURE 1.12 – Notation de nœud d'objet

6. **Partition :** Les activités peuvent impliquer de différents acteurs, tels que les différents groupes dans une organisation ou un système. pour cela, on utilise des partitions pour montrer quel participant est responsable pour quelles actions. Les Partitions divisent le diagramme en colonnes et/ou en lignes et contiennent des actions qui sont menées par le groupe responsable. Les partitions sont parfois dénommés *swimlanes*. La figure 1.13 illustre la notation de partition.

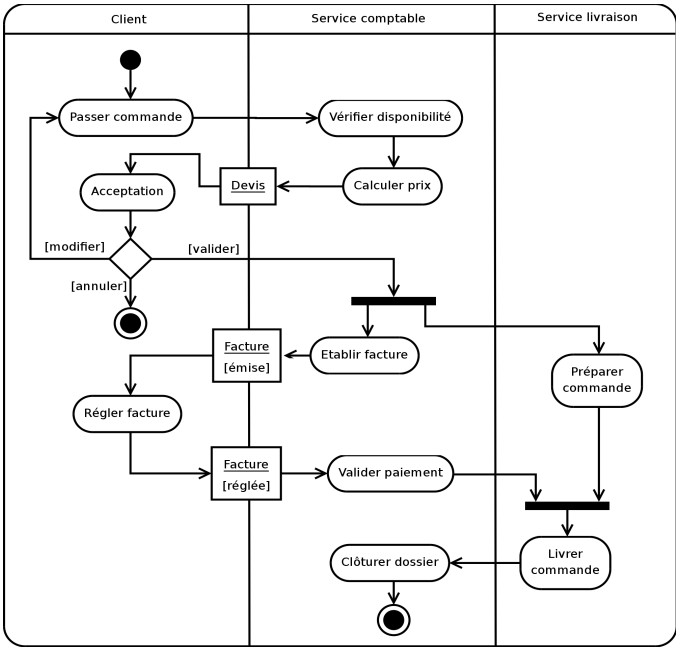



FIGURE 1.13 – Exemple de diagramme d'activité avec partitions

## 1.2.4 Extension d'UML

UML peut être étendu ou adapté à une méthode, organisation, ou utilisateur spécifique.

Il y a trois mécanismes d'extension dans *UML* qui sont [11] :

1. Stéréotypes
2. Valeurs étiquetées (tagged values)
3. Contraintes

Les mécanismes d'extension d'UML ont connu un grand succès, conduisant à une prolifération de nouveaux éléments de modèles et extensions. Les profils fournissent un mécanisme pour gérer ces extensions.

### 1.2.4.1 Stéréotypes

Le mécanisme d'extension stéréotypes définit un nouveau type d'élément de modèle basé sur un élément de modèle existant. Ainsi, un stéréotype est comme l'élément existant,

avec une sémantique et des propriétés supplémentaires qui ne sont pas présents dans l'élément existant.

Graphiquement, un stéréotype est rendu par un nom entouré de guillemets (« » ou par <<>> si les guillemets ne sont pas disponibles) et placé au-dessus de nom d'un autre élément. Un stéréotype peut aussi être indiqué par une icône spécifique [11].

La figure 1.14 représente des exemples de stéréotype.

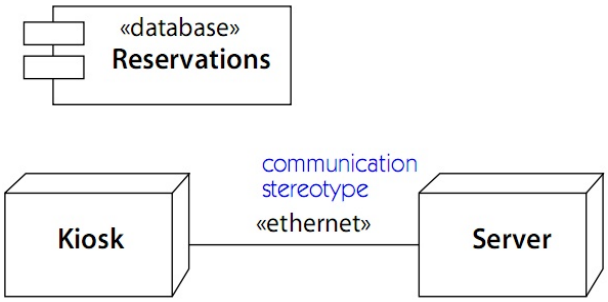



FIGURE 1.14 – Exemples de stéréotype [9]

#### 1.2.4.2 Valeurs Étiquetées (tagged values)

Le but d'une valeur étiquetée est d'attribuer (ajouter) une propriété à un élément de modèle, en plus de ces propriétés déjà définies dans le méta-modèle [10].

Les valeurs étiquetées sont exprimés sous la forme de *"nom = valeur"*, par exemple author="Tom", project_phase=2, ou last_update="1-07-02" [10].

#### 1.2.4.3 Contraintes

Les contraintes définissent les invariants qui permettent de préserver l'intégrité du système. Une contrainte définit une condition qui doit être vraie pour la durée du contexte dans lequel elle est définie. Elle est représentée par un texte placé entre accolades "{}" [10].

La figure 1.15 représente des exemples de contraintes.

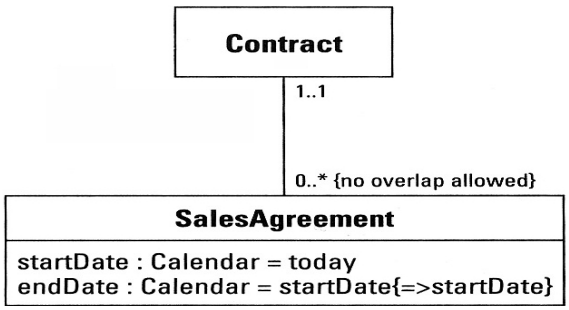



FIGURE 1.15 – Exemple de contraintes [10]

## 1.3 UML Mobile

Les insuffisances d'UML dans la modélisation des agents mobiles ont motivé les auteurs à proposer des extensions pour adapter UML à pouvoir modéliser ce type de système. Ces travaux sont très rares et limités en général. Parmi ces travaux on cite l'extension d'Hubert Baumeister [12], Cornel Klein [13], Kassem Saleh [14] et Haralambos Mouratidis [15].

Dans la suite, nous détaillerons l'extension de Miao Kang [16] que nous utiliserons dans notre projet.

### 1.3.1 Diagramme d'activités Mobile

Le diagramme d'activité mobile est une extension de diagramme d'activité d'UML, en utilisant les Stéréotypes pour modéliser les propriétés des agents mobiles suivantes :

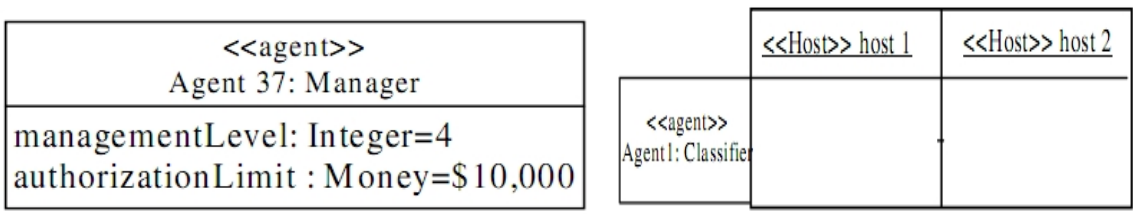
1. Mobilité.
2. Clonage.
3. Communication.

#### 1.3.1.1 Emplacement

On utilise les partitions multi-dimensions où une dimension représente un Emplacement et une autre dimension orthogonale représente un agent.

1. On ajoute le stéréotype ***«Host»*** et un paramètre comme un nom unique (adresse) à la partition dont la dimension est verticale. Cela permet de représenter un emplacement
2. On ajoute le stéréotype ***«agent»*** qui représente un agent comme une classe à la partition dont la dimension est horizontale.

La figure 1.16b illustre l'utilisation des stéréotypes ***«agent»*** *et* ***«Host»*** et la figure 1.16a illustre la classe *agent*.




(a) Classe agent (b) Exemple de notation

FIGURE 1.16 – Classe Agent [16]

#### 1.3.1.2 Mobilité

Pour représenter la mobilité des agents entre les hôtes, on utilise une activité spéciale comme l'activité ***"Go***", donc si un agent doit se déplacer du *"Host1"* vers *"Host2"* il doit

exécuter l'activité ***"Go"***, la figure 1.17 illustre cette représentation.

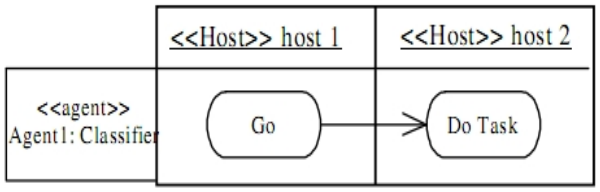



FIGURE 1.17 – Action Go [16]

Dans le cas d'une exception, on utilise la notation d'UML *EXceptionHandler* comme il est illustré par la figure 1.18

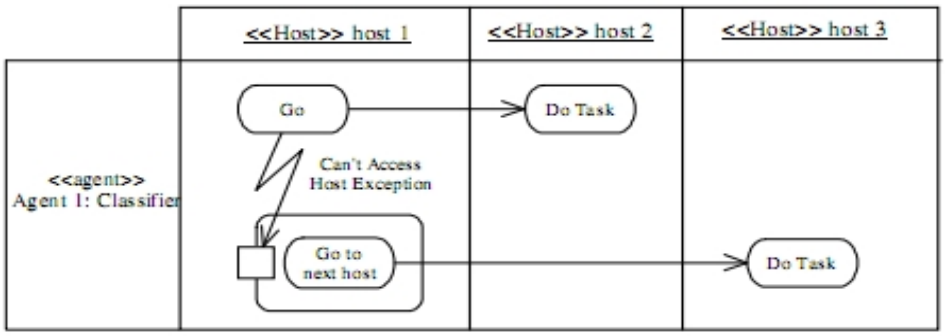



FIGURE 1.18 – Action Go avec exception [16]

#### 1.3.1.3 Clonage

Le clonage est un autre comportement important d'agents mobiles, qui comprend plusieurs sous-activités. Cette activité permet à un agent en premier de cloner, puis de l'envoyer à un autre hôte. Ce comportement peut être répété afin d'envoyer le nombre de clones nécessaires. On utilise la notation UML 2.0 " activités d'invocation " afin de modéliser cette opération.

La figure 1.19 représente cette notation

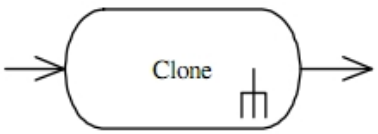



FIGURE 1.19 – Clonage [16]

#### 1.3.1.4 Communication

Les agents ont besoin de protocole de communication pour l'échange des messages. JADE *(Java Agent Development programming language)* [17], qui est conforme à FIPA *(Foundation for Intelligent Physical Agents)*, utilise une classe *ACLMessage*. Cette dernière représente le Langage de communication entre Agent (ACL) par messages. Cette

classe définit un ensemble de constantes, par exemple, *REQUEST*, *INFORM*, *QUERY*, qui doivent être utilisés pour se référer aux performatifs FIPA.

On utilise le stéréotype prédéfini, *«ACLMessage»* qui désigne la classe de message *"ACLMessage"*, pour spécifier explicitement le protocole de communication entre agent utilisé dans le modèle. L'exemple représenté par la figure **1.20** montre que le type de message de réception et d'envoi est "ACLmessage" et ses performatif est "REQUEST".

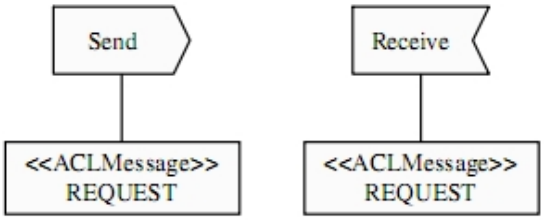
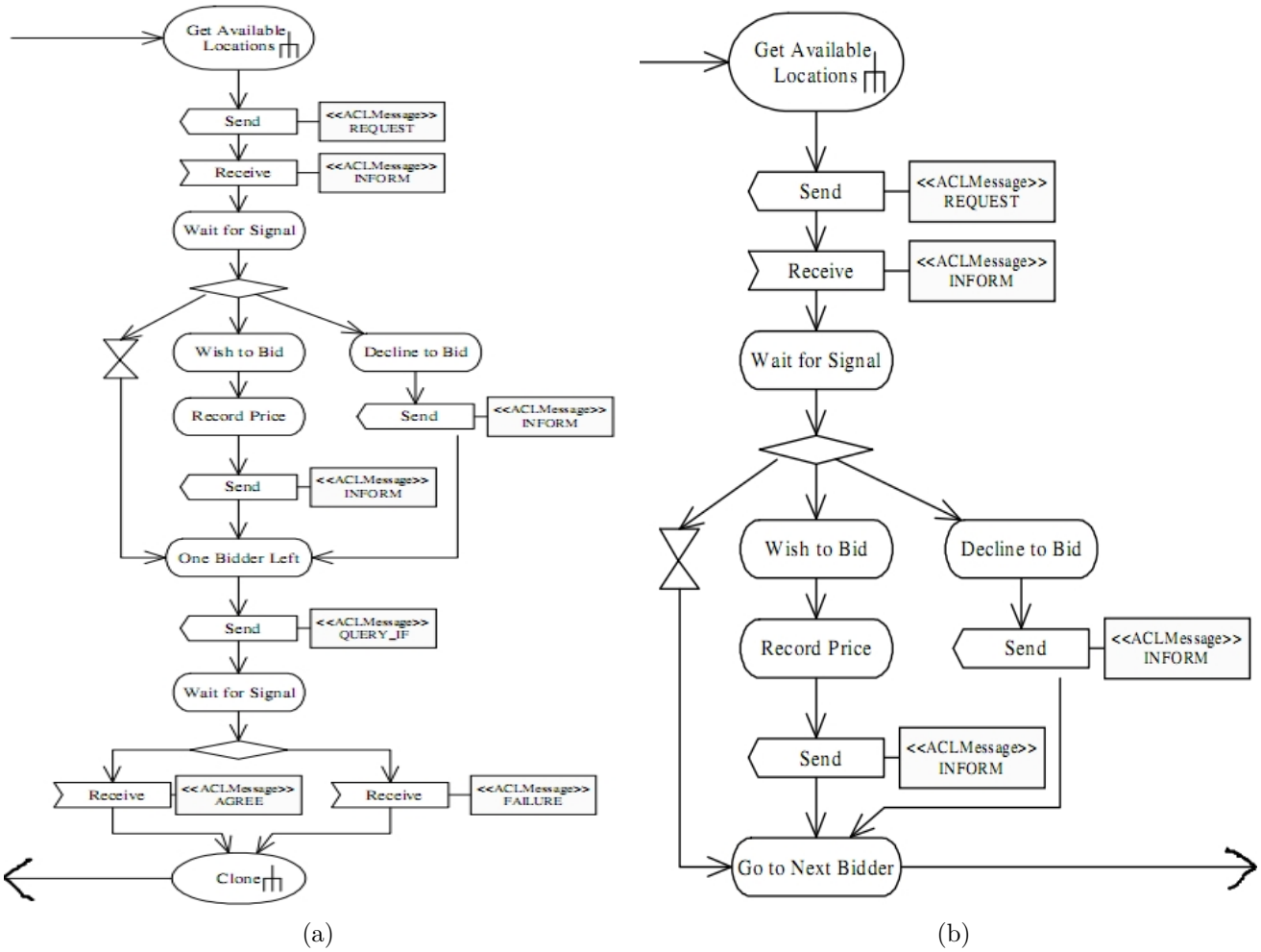



FIGURE 1.20 – Communication entre agent [16]

### 1.3.2 Exemple

Afin d'illustrer l'utilisation de cette approche, nous présenterons un système de vente aux enchères comme cas d'étude. Ce cas d'étude se compose de deux agents, un agent vendeur *(Auctioneer)*, qui est stationnaire et un agent mobile acheteur *(Bidder)*. L'agent *''Auctioneer''* réside dans l'hôte de vente et gère une interface d'un homme vendeur pour lui permettre de contrôler l'opération de vente entièrement. L'agent mobile ''*Bidder*'' va circuler autour des acheteurs pour accumuler ses commandes, en affichant l'offre de *''Auctioneer''* sur l'interface de chaque acheteur.

La figue 1.21 et 1.22**,** illustre le diagramme d'activités mobile de l'étude de cas.




(a) (b)

FIGURE 1.21 – Partie I et Partie II de l'exemple

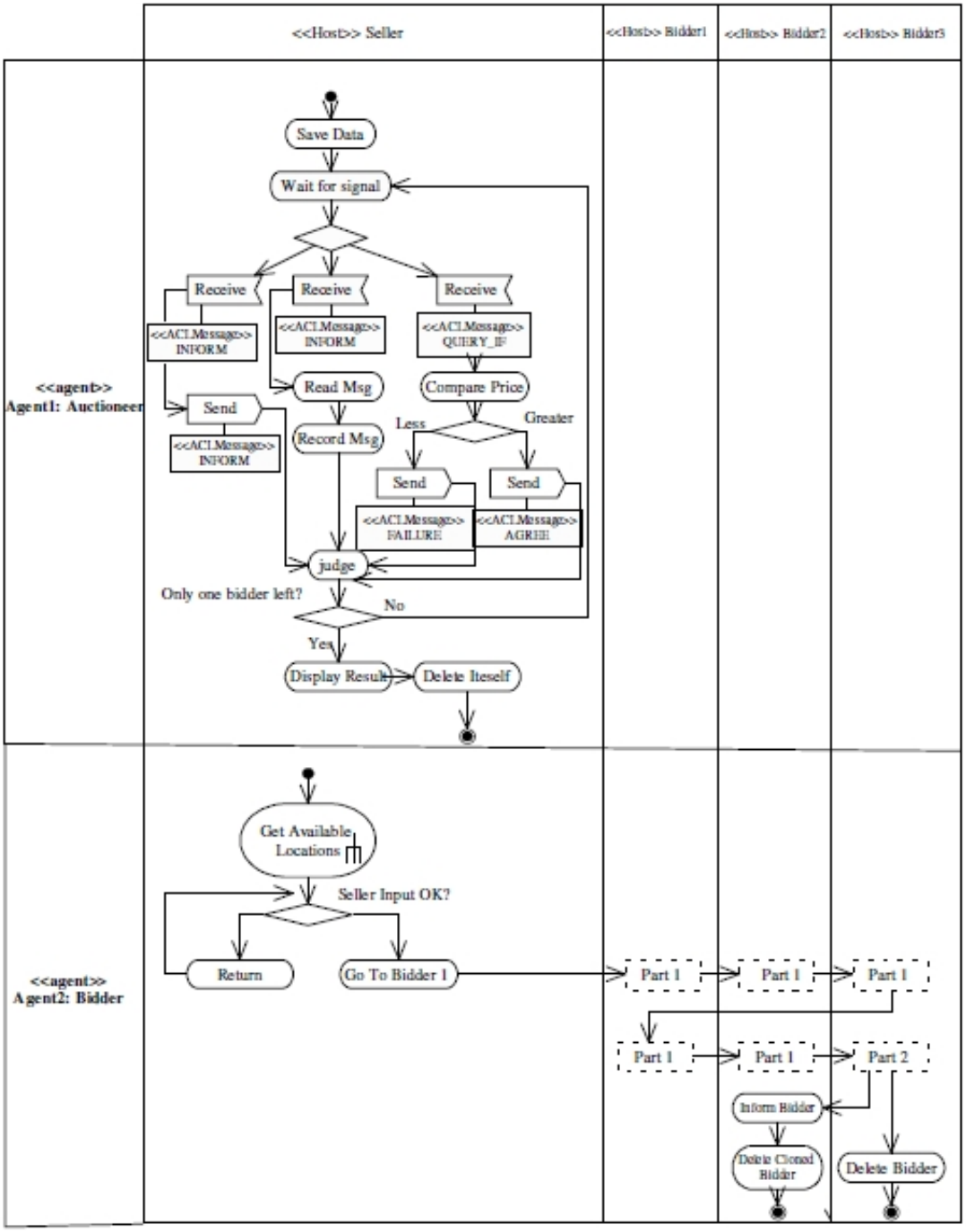



FIGURE 1.22 – Diagramme d'activité de l'étude de cas

## 1.4 Conclusion

Nous avons présenté dans ce chapitre des notions et des concepts de base sur la mobilité et plus particulièrement les agents mobiles. Ils sont, principalement, caractérisés par leur pouvoir de se déplacer entre les nœuds d'un réseau afin de compléter des tâches qui leur ont été confiées.

Les spécificités des agents mobile ont rendu l'utilisation des méthodes et des moyens de modélisation traditionnels tel que UML non satisfaisantes. Pour cette raison, nous avons présenté une extension de diagramme d'activité mobile qui permet de les modéliser.

Dans le chapitre qui va suivre nous présenterons un formalisme de réseau de Petri qui permet aussi de modéliser les agents mobiles.

# Chapitre 2

# Réseaux de Petri Mobile

Nous avons vu dans le chapitre précédent le langage semi formel UML mobile (*diagramme d'activité mobile*) qui permet de modéliser les agents mobiles. Malheureusement, il ne permet pas de les vérifier. Dans ce chapitre, nous allons voir une extension de formalisme de réseaux de Petri qui permet de modéliser les agents mobiles.

Historiquement, le concept de réseau de Petri a son origine de la thèse de Carl Adam Petri [18], présentée en 1962 à la Faculté de mathématique et de physique à l'Université technique de Darmstadt, en Allemagne de l'Ouest.

Il est caractérisé par son aspect graphique qui nous aide à comprendre facilement le système modélisé. Il permet de simuler les activités dynamiques, concurrentes et parallèles. Son aspect mathématique permet d'analyser le système modélisé grâce aux outils d'analyse automatique [19].

## 2.1 Concepts de Bases des Réseaux de Petri

### 2.1.1 Définition Graphique

Un réseau de Petri *(RDP)* est un graphe biparti orienté valué. Il a deux types de nœuds [20] :

1. *les places* : notées graphiquement par des cercles. Chaque place contient un nombre entier (positif ou nul) de marques (ou jetons). Ces derniers sont représentés par des points noirs.
2. *les transitions* : notées graphiquement par un rectangle ou une barre. Une transition qui n'a pas de place en entrée est appelée transition source et une transition qui n'a pas de place en sortie est appelée transition puits.

Les places et les transitions sont reliées par des arcs orientés où :

- ❑ *Un arc* relie, soit une place à une transition, soit une transition à une place mais jamais une place à une place ou une transition à une transition.
- ❑ Chaque arc est étiqueté par une valeur (ou un poids), qui est un nombre entier positif. L'arc ayant $k$ poids peut être interprété comme un ensemble de $k$ arcs

parallèls. Un arc qui n'a pas d'étiquette est un arc dont le poids est égal à 1. La figure 2.1 illustre la notation graphique d'un Réseau de Petri.

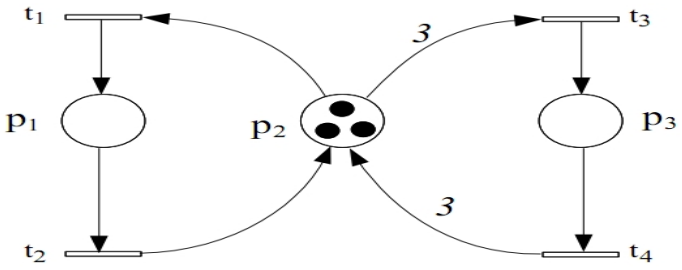



FIGURE 2.1 – Exemple d'un Réseau de Petri [21]

### 2.1.2 Définition Formelle

Formellement, un Réseau de Petri marqué est un 5-uplet, $Rdp = (P, T, F, W, M_0)$ où [20] :

- $P = \{P_1, P_2, \cdots, P_m\}$ est un ensemble fini non vide de places P.
- $T = \{t_1, t_2, \cdots, t_n\}$ est un ensemble fini non vide de transitions T.
- $F \subseteq (P \times T) \cup (T \times P)$ est un ensemble d'arcs où :
  - $(P \times T)$ est l'arc allant de $P$ à $T$.
  - $(T \times P)$ est l'arc allant de $T$ à $P$.
- $W : F \to \{1, 2, 3, \cdots\}$ est une fonction du poids où :
  - $W(P, T)$ : "*Pré(p, t)*" est le poids de l'arc allant de $P$ à $T$.
  - $W(T, P)$ : *"Post (p, t)"* est le poids de l'arc allant de $T$ à $P$.
- $M_0 : P \to \{0, 1, 2, 3, \cdots\}$ est le marquage initial.
- $P \cap T = \phi$ et $P \cup T \neq \phi$.

### 2.1.3 Marquage d'un Réseau de Petri

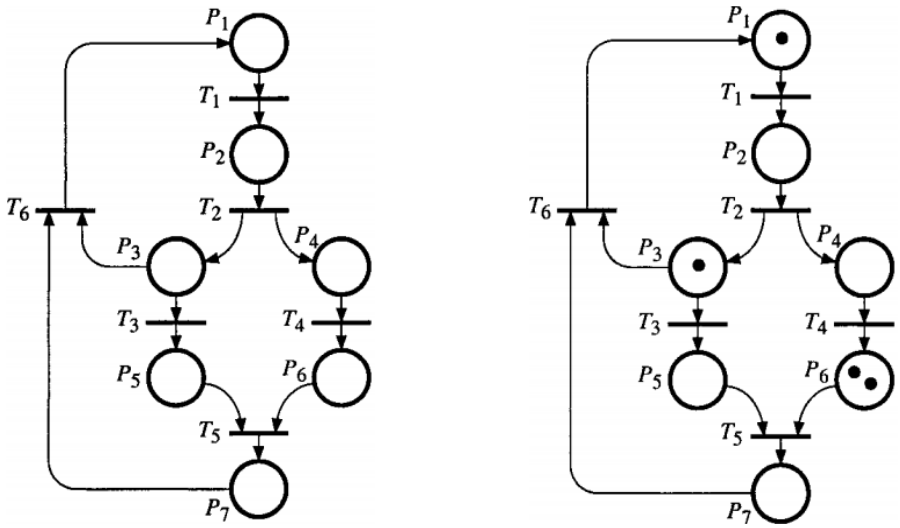
Un marquage est dénoté par un vecteur du nombre de jetons dans chaque place : la $i^{ième}$ composante correspond au nombre de jetons dans la $i^{ième}$ place [19].

Le marquage de réseau de Petri représenté par la figure 2.2b est donc

$$M = (1, 0, 1, 0, 0, 2, 0)$$

Un réseau de Petri $N = (P, T, F, W)$ est un réseau de Petri sans marquage initial et un réseau de Petri *Rdp* avec marquage initial $M_0$ est $Rdp = (N, M_0)$ [19].

Le réseau de Petri représenté par la figure 2.2a est un réseau de Petri non marqué alors que le réseau de Petri représenté par la figure 2.2b est un réseau de Petri marqué




(a) Réseau de Petri non marqué (b) Réseau de Petri marqué

FIGURE 2.2 – Exemple de Réseau de Petri [19]

### 2.1.4 Évolution d'un Réseau de Petri

L'évolution d'un Réseau de Petri correspond à l'évolution de son marquage au fil du temps (évolution de l'état du système) : il se traduit par un déplacement des jetons pour une transition $t$ de l'ensemble des places d'entrée vers l'ensemble des places de sortie de cette transition. Ce déplacement s'effectue par le franchissement de la transition $t$ selon des règles de franchissement [19] qu'on va voir dans la section 2.1.4.2.

#### 2.1.4.1 Transition validée

On dit qu'une transition est validée si toutes les places en entrée de celle-ci possèdent au moins une marque. Une transition source est par définition toujours validée [20].

#### 2.1.4.2 Règle de Franchissement

Si la transition est validée, on peut effectuer le franchissement de cette transition : on dit alors que la transition est franchissable [20].

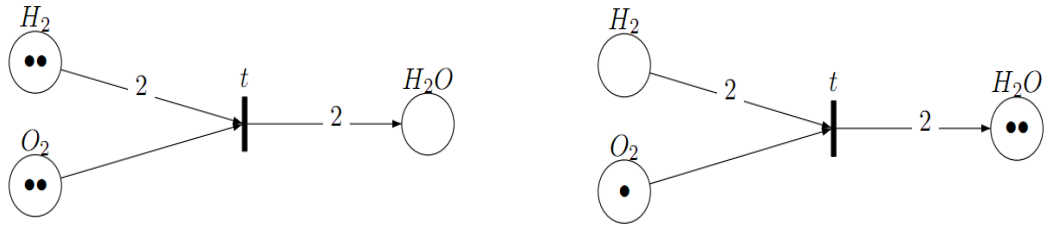
Le franchissement consiste à :

- ❑ retirer $W(p,t)$ jetons dans chacune des places en entrée $p$ de la transition $t$.
- ❑ ajouter $W(t,p)$ jetons à chacune des places en sortie p de la transition $t$.

La règle de franchissement est illustrée par la figure 2.3 en utilisant la réaction chimique connue [20] :

$$2H_2 + O_2 \rightarrow 2H_2O$$

La présence des deux jetons dans chaque place d'entrée (figure 2.3a), indique que 2 unités de $H_2$ et 2 unités de $O_2$ sont disponibles, donc la transition $t$ est franchissable. Après avoir franchi $t$, le marquage va changer et on obtient le réseau ayant le marquage comme celui de la figure 2.3b, maintenant $t$ n'est plus franchissable.




(a) Marquage avant de franchir la transition t.

(b) Marquage après avoir franchi t

FIGURE 2.3 – Exemple de règle de franchissement de transition [20]

## 2.2 Modélisation Avec les Réseaux de Petri

Les réseaux de Petri ont été conçus et utilisés principalement pour la modélisation. Plusieurs systèmes peuvent être modélisés par les réseaux de Petri, ces derniers peuvent être de natures très diverses : matériel informatique, logiciels informatiques, les systèmes physiques, les systèmes sociaux, etc [22].

Dans les sections suivantes nous donnerons quelques exemples de modélisation des problèmes informatiques et mathématiques avec les réseaux de Petri.

### 2.2.1 Parallélisme

Dans le réseau de Petri représenté par la figure 2.4 le franchissement de la transition *T1* met un jeton dans la place *P2* (ce qui marque le déclenchement du processus 1) et un jeton dans la place *P2* (ce qui marque le déclenchement du processus 2)

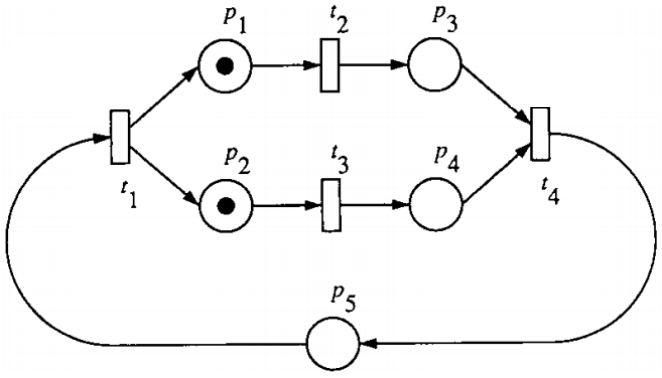



FIGURE 2.4 – Parallélisme dans les Réseaux de Petri [20]

### 2.2.2 Synchronisation

Les réseaux de Petri ont été utilisés pour modéliser une variété de mécanismes de synchronisation, y compris les problèmes de l'exclusion Mutuelle, producteur/consommateur, lecteurs/écrivains... etc [20].

#### 2.2.2.1 Exemple 1 : Problème du producteur/consommateurs

Le problème de *producteur/consommateurs*, implique un tampon partagé. Le processus producteur crée des objets qui sont mis dans le tampon, le consommateur attend jusqu'à ce qu'un objet est mis dans le tampon pour le consommer. Cela peut être modélisé comme le montre la figure 2.5. La place B représente le tampon ; chaque jeton représente un élément qui a été produit mais pas encore consommé [22].

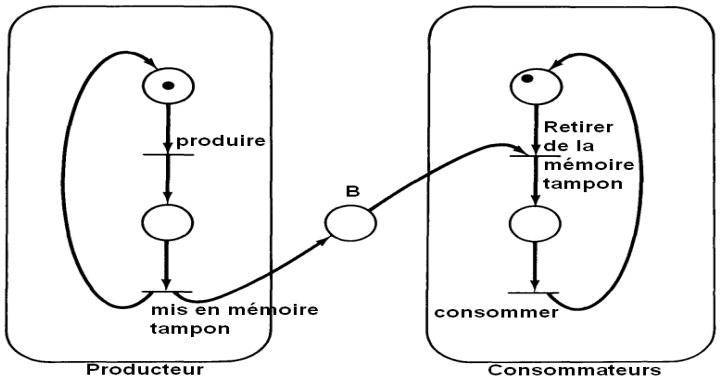



FIGURE 2.5 – Problème du producteur et consommateurs [22]

#### 2.2.2.2 Exemple 2 : Exclusion mutuelle

Ce problème peut être résolu par un réseau de Petri comme celui de la figure 2.6. La place $m$ représente la permission d'entrer dans la section critique. Pour qu'un processus entre dans la section critique, il doit avoir un jeton dans $p_1$ ou $p_2$ pour signaler qu'il souhaite entrer dans la section critique et il doit y avoir un jeton dans la place $m$ pour avoir la permission d'entrer dans la section critique.

Si les deux processus souhaitent entrer simultanément, les transitions $t_1$ et $t_2$ seront en conflit. Seulement l'une d'eux peut être franchie. Le franchissement de $t_1$ désactive $t_2$, ce qui oblige le processus 2 à attendre la sortie de processus 1 de sa section critique et à mettre un jeton à la place $m$.

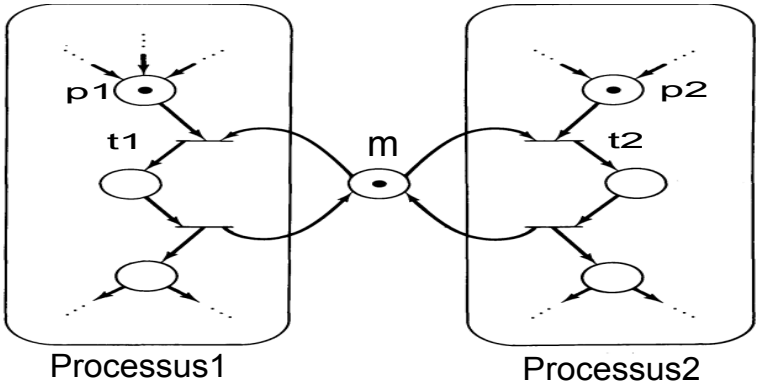



FIGURE 2.6 – Exclusion mutuelle [22]

### 2.2.3 Calcul De Flux De Données

Un calcul de flux de données est celui dans lequel les instructions sont activées pour l'exécution par l'arrivée de leurs opérandes. Il peuvent être exécutées simultanément.

Les réseaux de Petri peuvent être utilisés pour représenter non seulement le flux de contrôle, mais également le flux de données. Les jetons dénotent les valeurs des données en cours ainsi que la disponibilité des données [20].

Le réseau de Petri de la figure 2.7 représente un calcul de flux de données de la formule suivante [20] :

$$x = \frac{a+b}{a-b}$$

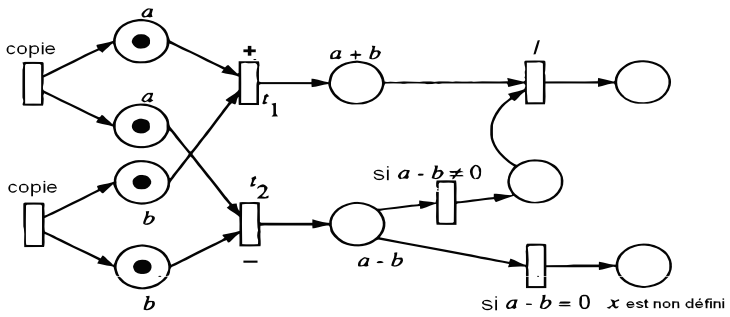



FIGURE 2.7 – Exemple d'un calcul de flux de données par un Rdp [20].

Après avoir modélisé le système, une question naturelle qui se pose : *"Qu'est-ce qu'on peut faire avec ce modèle ?"*. Un des principaux intérêts des réseaux de Petri est leur capacité d'analyser plusieurs propriétés et problèmes associés aux systèmes concurrents.

Dans la section suivante nous présenterons les principales propriétés des réseaux de Petri qui peuvent être analysées.

## 2.3 Principales Propriétés des Réseaux de Petri

### 2.3.1 Accessibilité

L'accessibilité est une propriété fondamentale pour étudier les propriétés dynamiques du système. Le franchissement d'une transition validée va changer la distribution des jetons sur le réseau selon les règles définies dans la section 2.1.4.2. Une séquence de franchissements $\sigma$ entraîne une séquence de marquages [20].

Un marquage $M_n$ est dit accessible à partir de $M_0$, s'il existe une séquence de franchissements $\sigma$ permettant de transformer $M_0$ à $M_n$. Une séquence de franchissements est notée par

$$\sigma = M_0 t_1 M_1 t_2 M_2 \cdots t_n M_n$$

ou simplement

$$\sigma = t_1 t_2 \cdots t_n$$

dans ce cas $M_n$ est accessible à partir de $M_0$ par $\sigma$ et on écrit :

$$M_0[\sigma > M_n$$

L'ensemble de séquences de franchissements à partir de $M_0$ dans un réseau $(N, M_0)$ est noté par $L(N, M_0)$ ou simplement $L(M_0)$.

L'ensemble de marquages possibles accessibles à partir de $M_0$ dans un réseau $(N, M_0)$ est noté par $R(N, M_0)$ ou simplement $R(M_0)$.

### 2.3.2 Bornitude et RDP Sauf

Un réseau de Petri $Rdp = (N, M_0)$ est *K-borné* ou simplement *borné* si le nombre de jetons dans chaque place ne dépasse pas un nombre fini $K$ pour tout marquage accessible de puis $M_0$ [20].

C'est- à-dire :

$$\forall M \in R(M_0), \forall p \in P : M(p) \leq k$$

Un réseau de Petri marqué $Rdp = (N, M_0)$ est sauf si et seulement s'il est *1-borné* [20].

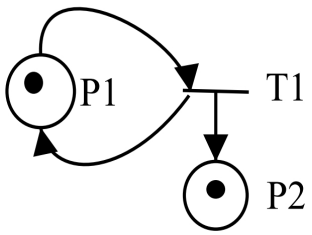



FIGURE 2.8 – Exemple d'un réseau de Petri non borné [23]

Dans l'exemple de réseau de Petri représenté par la figure 2.8 la transition *T1* admet la place *P1* comme l'unique place d'entrée. La place *P1* a un jeton : la transition *T1* est franchissable. Comme *P1* est aussi place de sortie de *T1*, le franchissement de *T1* ne change pas le marquage de *P1*. La transition *T1* est donc franchissable en permanence et peut donc être franchie un nombre de fois infini. Chaque franchissement de *T1* ajoute un jeton dans la place *P2*, le marquage de celle-ci peut donc tendre vers l'infini [23]. Par contre le réseau de Petri de la figure 2.4 est borné [20].

### 2.3.3 Vivacité

Le concept de vivacité est étroitement lié à l'absence complète de l'inter-blocage dans les systèmes opérationnels.

#### 2.3.3.1 Transition Vivante

Une transition $T_j$ est vivante pour un marquage initial $M_0$ si pour tout marquage accessible $M_i \in R(M_0)$, il existe une séquence de franchissements $\sigma$ à partir de $M_i$ contenant

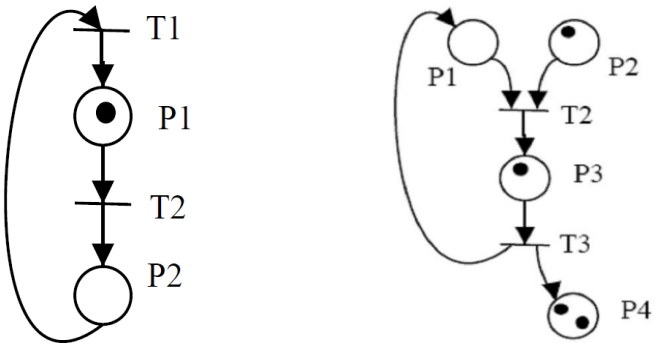
$T_j$. c.à.d quelque soit l'évolution du réseau à partir du marquage initial, le franchissement de cette transition est toujours possible [19].

Les transitions *T2* et *T3* du réseau de Petri marqué représenté par la figure 2.9b ne sont pas vivantes et les transitions *T1* et *T2* du réseau de Petri marqué représenté par la figure 2.9a sont vivantes

#### 2.3.3.2 RDP Vivant

Un réseau de Petri $Rdp = (N, M_0)$ est vivant si toutes ses transitions sont vivantes [19].

Le réseau de Petri marqué représenté par la figure 2.9a est vivant, alors que le réseau de Petri représenté par la figure 2.9b n'est pas vivant.




(a) Rdp vivant (b) Rdp non vivant

FIGURE 2.9 – Exemple de vivacité des Réseau de Petri [23]

### 2.3.4 Blocage

Un marquage $M$ d'un réseau $(N, M_0)$ est appelé marquage *"puits"* si aucune transition n'est franchissable depuis $M$.

Un réseau est dit sans blocage si tout marquage accessible depuis $M_0$ n'est pas un marquage "puits" [19].

Le réseau de Petri marqué représenté par la figure 2.9b a pour blocage le marquage :

$$M_3 = [1, 0, 0, 4]$$

### 2.3.5 Réinitialisable et État d'accueil

Un réseau de Petri s'appelle réinitialisable si pour chaque marquage $M$ dans $R(M_0)$, $M_0$ est accessible à partir de $M$.

Un marquage $M_0$ est État d'accueil si pour chaque marquage $M$ dans $R(M_0)$, $M$ est accessible à partir de $M_0$ [20].

La figure 2.10 représente un réseau de Petri réinitialisable.

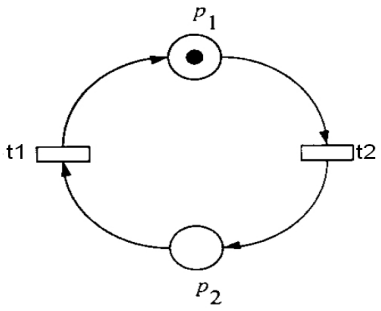



FIGURE 2.10 – Exemple d'un réseau de Petri réinitialisable [20]

### 2.3.6 Couverture

Un marquage $M$ dans un réseau de Petri $(N, M_0)$ est dit couvrable s'il existe un marquage $\grave{M}$ dans $R(M_0)$ tel que $\grave{M}(p) \geq M(p)$ pour chaque place du réseau [20].

### 2.3.7 Persistance

Un réseau de Petri $(N, M_0)$ est dit persistant si pour n'importe quelles deux transitions, le franchissement d'une transition ne doit pas inhiber l'autre transition. Si une transition dans un réseau de Petri persistant est une fois validée, elle reste validée jusqu'à son franchissement [20].

La figure 2.11 représente un réseau de Petri persistant.

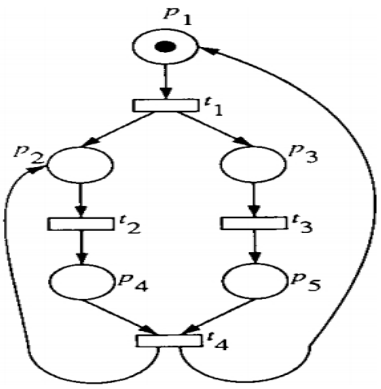



FIGURE 2.11 – Exemple d'un réseau de Petri persistant [20]

## 2.4 Les Réseaux de Petri de Haut Niveau

Pour l'utilisation des réseaux de Petri dans la modélisation des systèmes réels, plusieurs auteurs ont trouvé qu'il est convenable d'étendre le formalisme de réseau de Petri pour compacter la représentation de modèle ou pour étendre le pouvoir de modélisation du formalisme de réseau de Petri. Ce qui a donné naissance aux réseaux de Petri de haut niveau.

### 2.4.1 Réseau de Petri Coloré

Le développement des réseaux de Petri coloré (CP-nets [1] ou CPN) a été motivé par le désir de développer un langage de modélisation bien-fondé théoriquement et assez versatile en même temps. Ils sont utilisés pour les systèmes de taille et de complexité qu'on retrouve dans les projets industriels.

Contrairement aux réseaux de Petri ordinaires, chaque place a un type associé déterminant le type de données que la place peut contenir et chaque jeton a une valeur de données dont le type est le même que le type associé à la place.

Les actions d'un CP-net sont représentées par des transitions (qui sont sous forme de rectangle). Pour que la transition soit franchissable, on doit avoir suffisamment de jetons dans les places d'entrée. Ces jetons doivent avoir une valeur qui correspond aux expressions d'arc [24, 25].

### 2.4.2 Réseau de Petri Objet

Les réseaux de Petri Objet *(OPN)* étendent le formalisme des réseaux de Petri colorés avec une intégration complète des propriétés orientées objet y compris l'héritage, le polymorphisme et la liaison dynamique. L'orientation objet fournit des primitives de structuration puissante permettant la modélisation des systèmes complexes [26].

## 2.5 Mobilité et Réseaux de Petri

Les réseaux de Petri ordinaires sont trop statiques pour être directement utilisés comme langage de spécification des agents. En particulier, ils ne présentent aucun moyen direct pour exprimer les agents qui évoluent, qui changent de structure, qui communiquent avec d'autres agents et qui peuvent être, dynamiquement, liés à d'autres agents. Pour combler cette lacune, des réseaux de Petri de haut niveau ont été définis, tel que les réseaux de Petri imbriqués *(Nested Petri Net).*

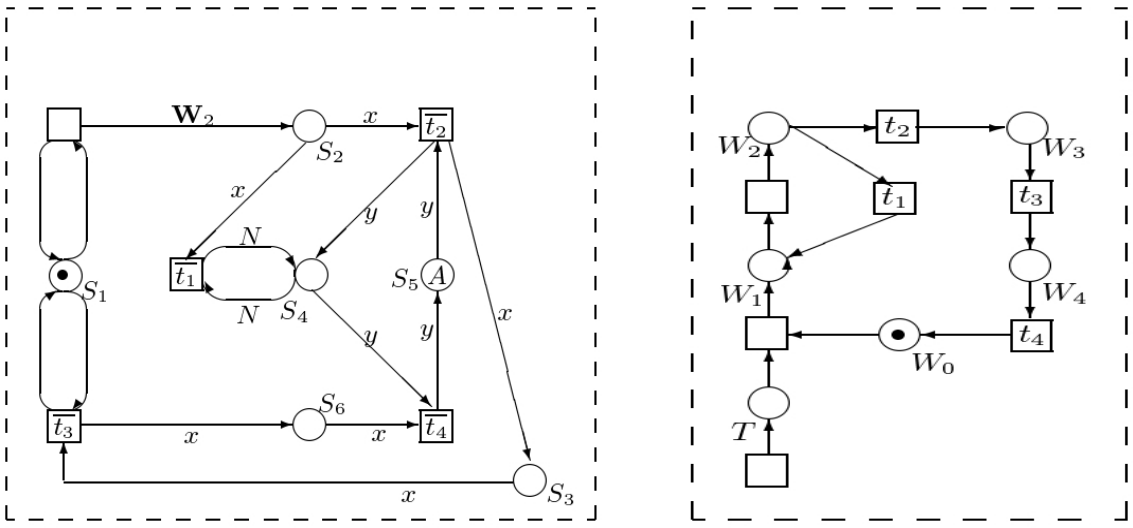
Le réseau de Petri imbriqué est un formalisme pour la modélisation des systèmes multi-agents hiérarchiques. Les jetons dans les réseaux de Petri imbriqués sont des éléments représentés par des réseaux eux-mêmes [27]. Dans ce qui suit, on ne considère que les réseaux de Petri imbriqué à deux niveaux dont les jetons de réseaux sont des éléments représentés par des réseaux de Petri ordinaires.

### 2.5.1 Exemple d'introduction

Pour donner une idée intuitive sur les réseaux de Petri imbriqué, nous commençons par un petit exemple pris de [28] représenté par la figure 2.12

Il modélise un ensemble de travailleurs qui reçoivent certaines tâches, de temps en temps. Le comportement d'un travailleur est décrit par un élément réseau *(EN)* comme

1. Coloured Petri Nets




(a) Système réseau *(SN)* (b) Élément réseau *(EN)*

FIGURE 2.12 – Exemple de Réseaux de Petri Imbriqués [28]

le montre la figure 2.12b. Un *EN* est un réseau de Petri élémentaire. Lorsque il y a une tâche, un travailleur emprunte un outil à partir de tampon d'outils. Le tampon d'outils est représenté par le système réseau *(SN)* comme dans la figure 2.12a. Il s'agit d'un réseau de Petri de haut niveau avec des jetons de trois types :

1. les points noirs,
2. les outils (les points non structurés avec certaines couleurs)
3. les travailleurs (représentés par des réseaux).

Le nombre de travailleurs impliqués dans ce système est illimité. L'ensemble des outils $A$ est fixe et initialement représenté dans la place $S_5$. Dans notre exemple, $A$ est fini (avec $N$ éléments).

Les arcs, dans le système réseau *SN,* sont marqués par des expressions (variables et constantes dans notre exemple), comme dans les réseaux de Petri de haut niveau. Si aucune expression n'est associée, l'arc est censé de transférer un point noir.

Dans notre exemple, l'expression *"x"* d'un arc est une variable pour un travailleur (ayant un réseau marqués *EN* comme une valeur), $y$ est une variable d'un outil (l'arc correspondant transfert des jetons colorés pour un outil), $W_2$ est une constante *EN* pour un élément réseau avec le marquage $\{W_2\}$ c.à.d il comporte un seul jeton dans la place $W_2$.

Certaines transitions sont marquées par les étiquettes $t_1, t_2, t_3, t_4$ dans EN et $\overline{t_1}, \overline{t_2}, \overline{t_3}, \overline{t_4}$ dans SN. Ils sont utilisés pour la synchronisation de franchissement des transitions dans EN et SN. Ainsi le franchissement de la transition marquée par $\overline{t_2}$ dans SN ne peut être effectué que simultanément avec le franchissement de la transition marquée par $t_2$ dans EN qui est impliqué dans le franchissement de $\overline{t_2}$ (qui est transféré par ce dernier).

La signification des places dans l'élément réseau EN est comme suit :

❑ ***T*** : Il y a une tâche pour le travailleur.

- ❑ $W_0$ : le travailleur ne fait rien.
- ❑ $W_1$ : le travailleur a reçu une tâche.
- ❑ $W_2$ : le travailleur fait une demande pour un outil.
- ❑ $W_3$ : le travailleur est occupé par une tâche.
- ❑ $W_4$ : le travailleur a terminé une tâche.

La signification des places dans le système réseau SN est comme suit :

- ❑ $S_1$ : un tampon d'outils est ouvert.
- ❑ $S_2$ : un travailleur fait une demande pour un outil.
- ❑ $S_3$ : des travailleurs avec des outils
- ❑ $S_4$ : le nombre d'outils empruntés
- ❑ $S_5$ : outils disponibles
- ❑ $S_6$ : des travailleurs retournent des outils

Pour illustrer le comportement des réseaux de Petri imbriqués, nous suivons plusieurs étapes possibles de *NPN*. Dans le marquage initial représenté par la figure 2.12a et la figure 2.12b, la transition sans étiquette dans *SN* peut être franchie, mettant un jeton réseau $W_2$ dans la place $S_2$. Cette étape crée une instance de l'*EN* dans la place $S_2$. Puis, la transition marquée par $\overline{t_2}$ dans *SN* peut être franchie simultanément avec la transition marquée par $t_2$ dans l'élément réseau qui est un jeton dans $S_2$. Ensuite, l'élément réseau *EN* avec le marquage $W_3$ sera situé dans la place $S_3$, l'ensemble $A$ dans $S_5$ sera diminué d'un jeton et la place $S_4$ reçoit un jeton. Alors la transition marquée par $\overline{T_4}$ dans *SN* peut être franchie simultanément avec la transition marquée par $T_4$ dans l'élément réseau se trouvant dans $S_3$.

La poursuite de ce processus peut conduire à un marquage comme il est représenté par la figure 2.13, où il y a un travailleur faisant une demande pour un outil, deux travailleurs avec des outils et un travailleur est venu pour rendre un outil.

$\grave{A}$ désigne l'ensemble $A$ diminué de trois jetons.

### 2.5.2 Définition Formelle

Un réseau de Petri imbriqué *(NPN)* est un formalisme pour la modélisation des systèmes multi-agents hiérarchiques. Les jetons dans un *NPN* sont des éléments dont ils [27, 29, 30] :

1. peuvent être représentés par des réseaux de Petri eux-mêmes.
2. ces derniers peuvent évoluer et disparaître durant la vie de système.
3. leur nombre est illimité.

Un réseau de Petri imbriqué *(NPN)* est défini formellement comme suit [27] :

Soit $Var = \{v, \cdots\}$ un ensemble des noms des variables et $Con = \{c, \ldots\}$ un ensemble des noms des constantes. On écrit *Atom* pour l'ensemble des atoms $Var \cup Con$. On définit un langage d'expression *Expr(Atom)* sur *Atom* comme suit.

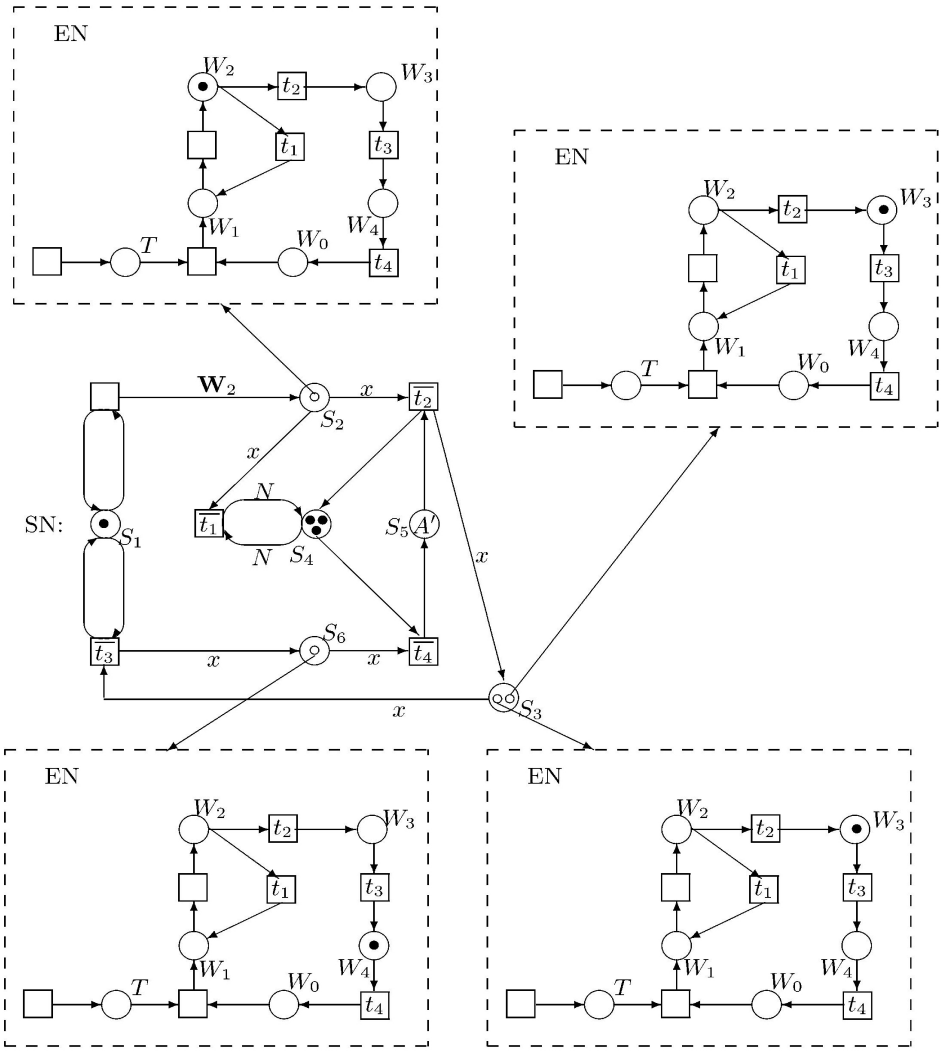



FIGURE 2.13 – Exemple d'un état accessible d'un *NPN [28]*

#### 2.5.2.1 Définition 1

1. un atom $atom \in Atom$ est une expression dans *Expr(Atom)* avec une dimension égale à 1.
2. Si $atom_1, atom_2, ..., atom_n \in Atom$, alors le tuple $(atom_1, atom_2, ..., atom_n)$ est une expression dans $Expr\,(Atom)$ avec une dimension égale à $n$.
3. si $e_1, e_2 \in Expr\,(Atom)$ sont des expressions avec la même dimension $n$, alors $(e_1 + e_2)$ est une expression dans *Expr(Atom)* avec la dimension $n$.

Nous tenons en considération les points suivants dans la définition d'un réseau de Petri imbriqué :

1. Soit $e \in Expr\,(Atom)$, par $Var\,(e)$ on dénote l'ensemble des occurrences des variables dans $e$.
2. les constantes dans les expressions de *Expr(Atom)* sont interprétées soit comme :
   (a) des Réseaux de Petri marqués ordinaires (Jeton réseau).
   (b) des jetons individuels sans structure intérieure (Jeton atom).

3. Par $A_{net} = \{a, ...\}$ on désigne l'ensemble des jetons réseau et par $A_{atom}$ on désigne l'ensemble des jetons atom. On suppose que $A_{atom} \neq \emptyset$ et on fixe un jeton distingué, qui correspond à 1 dans le langage d'expression *Expr(Atom)* et qui sera désigné comme un jeton point noir.
4. une expression $(a_1, a_2, \cdots, a_n)$ sera interprétée comme un tuple d'éléments $(a_1, a_2, \cdots, a_n)$ et l'opération *"+"* dans $e \in Expr\,(Atom)$ sera interprété comme une union multi-ensemble. Ainsi, les expressions à partir de $Expr\,(Atom)$ ont des tuples de jetons multi-ensembles comme leurs valeurs. Comme d'habitude, il suffit d'écrire $n$ pour la valeur de *n.1* (n copies de jeton point noir).
5. Soit également $Lab_v = \{l_1, l_2, ...\}$ et $lab_h = \{\lambda_1, \lambda_2, ...\}$ deux ensembles disjoints d'étiquettes. Pour chaque étiquette $l \in lab_v$ et chaque étiquette $\lambda \in lab_v$ nous définissons une étiquette adjacentes $\overline{l} \in lab_v$ et $\overline{\lambda} \in lab_v$ respectivement, telle que pour $l_1, l_2 \in lab_v, l_1 \neq l_2$ implique $\overline{l_1} \neq \overline{l_2}$ ; pour $\lambda_1, \lambda_2 \in lab_h, \lambda_1 \neq \lambda_2$ implique $\overline{\lambda}_1 \neq \overline{\lambda}_2$ et $\overline{\overline{l}} =_{deff} l; \overline{\overline{\lambda}} =_{deff} \lambda$.
6. Les étiquettes de $lab_v$ sont utilisées pour la synchronisation verticale et les étiquettes de $lab_h$ pour la synchronisation horizontale.
7. soit $Lab =_{deff} lab_v \cup lab_h$.

#### 2.5.2.2 Définition 2

Un réseau de Petri imbriqué (NP-net) est un tuple

$$NPN = \left(Atom, Lab, \left(EN_1, m_0^1\right), \cdots, \left(EN_k, m_0^k\right), SN, \Lambda\right)$$

Avec

1. $Atom = Var \cup Con$ : un ensemble d'atoms.
2. $Lab = lab_v \cup lab_h$ : un ensemble d'étiquettes comme défini dans la section 2.5.2.1 .
3. $(EN_1, m_0^1), \cdots, \left(EN_k, m_0^k\right)\,(k \geq 1)$ : un nombre fini $(k \geq 1)$ des réseaux de Petri ordinaires $EN_1, \cdots, EN_k$, appelés éléments réseau de *NPN* avec leurs marquages initiaux $m_0^1, \cdots, m_0^k$ respectivement.
4. $SN = (N, \mathcal{L}, \mathcal{U}, W, M_0)$ : un réseau de Petri de haut niveau, appelé système réseau de *NPN*, où :
   - ❑ $N = (P, T, F)$ : est un réseau.
   - ❑ $P$ : un ensemble des noms (places de réseau), chaque place est attribuée une arité (nombre naturel positif), $P \cap Atom = \emptyset$.
   - ❑ $\mathcal{L} = Expr\,(Atom)$ : un langage d'expressions comme défini dans la section 2.5.2.1.
   - ❑ $\mathcal{U} = (A, \mathcal{I})$ : un modèle de $\mathcal{L}$ constitué d'un univers $A$ et une fonction d'interprétation $\mathcal{I}$ où

$$A = A_{net} \cup A_{atom}$$

et

$$A_{net} = \{(EN, m) \mid \exists i = 1, \cdots, k : EN = EN_i, /m \text{ un marquage dans } EN_i\}$$

c.à.d. $A_{net}$ est l'ensemble des *EN* marqués. Nous appelons $A_{net}$ l'ensemble des jetons réseau de *SN et* $A_{atom}$ l'ensemble des jetons atom de *SN*. Comme il a déjà été mentionné, on suppose que $A_{atom}$ contient au moins un élément "un jeton point noir".

La fonction d'interprétation $\mathcal{L} : Con \rightarrow A$ donne quelque interprétation à des noms de constantes. L'interprétation pour un tuple et l'opérations "+" a été définie ci-dessus.

- $W$ : fonction de mappage d'un arc $(x, y) \in F$ à une expression $W(x, y) = \overline{\theta} = (\theta_1, \theta_2, ..., \theta_n)$ avec une dimension $n$ où $\theta_i \in \mathcal{L}\,(1 \leq i \leq n)$ et $n$ est une arité de position d'incident d'un arc $(x, y)$.

  En outre, l'expression d'arc d'entrée aux transitions doit satisfaire les restrictions suivantes (appelé les restrictions d'expression d'arc d'entrée ) :

  (a) Il n y a pas de constant réseau (avec des valeurs dans $A_{net}$) dans les expressions d'arc d'entrée.

  (b) Chaque variable n'a pas plus d'une occurrence dans chaque expression d'arc d'entrée.

  (c) pour chaque deux expressions $W(p_1, t)$ $W(p_2, t)$ attribuées à deux arcs d'entrée d'une même transition $t$ , $Var(W(p_1, t) \cap W(p_2, t)) = \emptyset$ .

  L'exemple de la figure 2.14 démontre ces trois restrictions.

  Si l'expression ne figure pas explicitement pour certains arcs on la suppose qu'elle soit $1 \in \mathbb{N}$.

- $M_0$ : un marquage distingué, appelé marquage initial du réseau, où un marquage est une fonction de mappage $M$, assignant à chaque place $p \in P$ un multi-ensemble *M (p)* sur l'ensemble $(A)^n$ des tuples, où $n$ est un arité de $p$.

5. $\Lambda$ : une fonction partielle d'étiquetage des transitions, permettant l'attribution des étiquettes de $Lab_v$ à des transitions dans un système réseau *SN et des* étiquettes de $Lab_v \cup Lab_h$ à des transitions dans les éléments réseau *EN*.

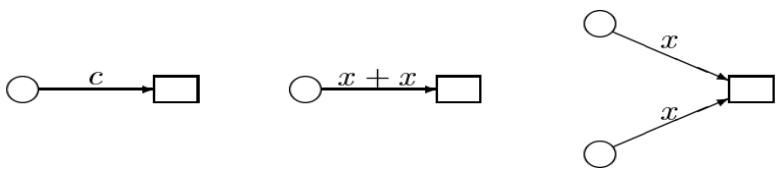



FIGURE 2.14 – Exemples d'expressions d'arc d'entrée interdites [27]

Donc, un réseau de Petri imbriqué se compose d'un ensemble de réseaux de Petri ordinaires qui définissent la structure des jetons réseau (*éléments réseau "EN"*) et un *SN*

(*Système réseau*), qui est un réseau de Petri de haut-niveau. Tout cela définit l'aspect structurel et formel d'un réseau de Petri imbriqué. Dans la section suivante nous verrons la définition de l'aspect comportemental du réseau de Petri imbriqué *(NPN)*.

### 2.5.3 Comportement d'un Réseaux de Petri Imbriqué

Le comportement d'un réseau de Petri imbriqué est défini par quatre types d'étapes [27, 29] :

#### 2.5.3.1 Étape de transport

Une étape de transport ne change pas le marquage interne des jetons réseau, mais peut transférer ou supprimer certains jetons. Ainsi de nouveaux jetons réseau peuvent évoluer en tant que résultat d'une étape de transport.

#### 2.5.3.2 Étape d'élément-autonome

Une étape d'élément-autonome est le franchissement d'une transition non marqué dans l'un des éléments réseau *(EN)* tandis que tous les éléments réseau restent dans la même place du système réseau *(SN)*.

#### 2.5.3.3 Étape de synchronisation horizontale

Cette étape signifie le franchissement simultané de deux jetons réseau, situés dans la même place d'un *SN*.

Un tel franchissement synchrone de $t_1$ et $t_2$ est appelé l'étape de synchronisation horizontale.

#### 2.5.3.4 Étape de synchronisation verticale

Elle est le franchissement simultané d'une transition $t$, étiquetée par une étiquette $l \in Lab \cup \bar{Lab}$ dans le *SN et des transitions* étiquetées par une étiquette adjacentes $\bar{l}$ dans un élément réseau impliqué dans le franchissement de $t$.

## 2.6 Conclusion

Nous avons vus dans ce chapitre le formalisme du réseau de Petri. Ils sont des outils graphiques et mathématiques puissants pour la modélisation, l'analyse et la vérification des systèmes.

Pour bénéficier de ces avantages dans la modélisation des systèmes dynamiques et complexes, les auteurs ont proposé des extensions au réseau de Petri ordinaire. Ainsi nous avons présenté dans ce chapitre les réseaux de Petri imbriqués pour la modélisation des agents mobiles. Ils sont des réseaux de Petri de haut niveau où les jetons peuvent être des réseaux de Petri eux-mêmes.

# Chapitre 3

# Approche de Transformation

Dans ce chapitre, nous présenterons, notre approche de transformation. Nous commencerons par l'étude de la transformation des modèles, puis nous étudierons brièvement un outil adapté pour cela. Ensuite nous présenterons notre approche pour transformer les diagrammes d'activités mobiles vers les réseaux de Petri. Enfin, nous conclurons par un rappel sur les principes de la méthode proposée.

## 3.1 Modèle et Méta-Modélisation

Qu'entendons-nous lorsque nous utilisons le mot modèle ? Plusieurs définitions ont été données. Parmi lesquelles :

- ❑ Un modèle est une abstraction d'un système (réel ou à base de langage) permettant de tirer des prédictions ou des conclusions [31].
- ❑ L'idée centrale de la modélisation est de produire une version réduite du système désiré afin de déterminer et d'évaluer ses propriétés saillantes [32].

Quels sont les propriétés des systèmes saillantes ? Quelles prédictions et inférences que les modèles permettent ? Bézivin et Gerbé offre une définition similaire, et proposer une réponse à ces questions.

- ❑ Un modèle est une simplification d'un système conçu avec un objectif visé à l'esprit. Le modèle devrait être en mesure de répondre à des questions en place du système actuel. Les réponses fourni par le modèle devrait être les mêmes que ceux proposés par le système lui-même, à la condition que les questions sont dans le domaine défini par l'objectif général du système [33].

dans le contexte du développement de logiciels dirigé par les modèles Warmer et ses collègues donnent la définition suivante

- ❑ Un modèle est une description (d'une partie) d'un système écrit dans un langage bien défini (méta-modèle) [34].

Dans son sens le plus large, un méta-modèle est un modèle d'un langage de modélisation. Le terme *"méta"* signifie transcendant ou au-dessus. Un méta-modèle décrit un langage de modélisation à un niveau d'abstraction supérieur que le langage de modélisa-

tion lui-même. Un méta-modèle est aussi un modèle, il a deux principales caractéristiques distinctives [35] :

Premièrement, il doit capturer les caractéristiques et les propriétés essentielles de langage modélisé. Ainsi, un méta-modèle doit être capable de décrire :

1. ***la syntaxe concrète :*** est la notation que le langage modélisé fournit et qui facilite la présentation et la construction des modèles. Il y a deux types principaux de syntaxe concrète ; la syntaxe textuelle et la syntaxe visuelle. La syntaxe textuelle permet de décrire les modèles sous une forme textuelle structurée et La syntaxe visuelle permet de décrire les modèles sous la forme de diagramme.
2. ***la syntaxe abstraite :*** permet de décrire le vocabulaire des concepts du langage modélisé et comment ils peuvent être combinés pour créer des modèles. Elle se compose d'une définition des concepts, des rapports qui existent entre les concepts et des règles qui définissent comment les concepts peuvent être combinés.
3. ***la sémantique :*** La définition de la sémantique du langage modélisé est très importante afin d'être clair sur ce que représente et signifie le langage modélisé. Dans le cas contraire, des fausses hypothèses peuvent être faites sur le langage modélisé qui mènent à une utilisation incorrecte.

Deuxièmement, un méta-modèle doit faire partie d'une architecture de méta-modèle. Une architecture de méta-modèle permet à un méta-modèle d'être vu comme un modèle, il est lui-même décrit par un autre méta-modèle. Cela permet à tous les méta-modèles d'être décrits par un méta-modèle unique. Ce méta-modèle unique, connu sous le nom d'une méta-méta-modèle, est la clé de méta-modélisation car il permet à tous les langages de modélisation d'être décrits d'une manière unifiée.

### 3.1.1 Architecture Méta-Modèle

L'architecture de méta-modèle traditionnelle proposée par l'OMG est basée sur 4 méta-niveaux distincts. Dans cette architecture, un modèle à un méta-niveau est utilisé pour spécifier des modèles dans le méta-niveau ci-dessous. À son tour un modèle à un méta-niveau peut être perçu comme une instance de certains modèles dans le méta-niveau au-dessus [36].

Les quatres méta-niveaux sont :

- ***(M0) :*** à ce niveau se trouve le système réel, système modélisé.
- ***(M1) :*** à ce niveau se trouve le modèle du système réel défini dans un certain langage. Il peut être les classes d'un système orienté objet, ou les définitions des tables d'une base de données relationnelle.
- ***(M2) :*** à ce niveau se trouve le méta-modèle définissant ce langage, par exemple, les éléments UML comme la classe, l'attribut et l'opération.
- ***(M3) :*** à ce niveau se trouve le méta-méta-modèle définissant les propriétés de tous les méta-modèles.

Le tableau 3.1 définit les concepts qui différencient ces méta-niveaux.

| | |
|---|---|
| Méta-méta-modèle | Langage de spécification des méta-modèles |
| Méta-modèle | Définition du langage utilisé pour exprimer le modèle |
| Modèle | Abstraction du système |
| Système | Information et flux de contrôle d'un domaine |

TABLE 3.1 – Niveaux d'abstraction de la méta-modélisation

## 3.2 Transformation de Modèle

La transformation de modèle est une activité centrale dans le développement de logiciels dirigé par les modèles. Elle est utilisée aussi pour l'optimisation des modèles et d'autres formes d'évolutions des modèls. En outre, la transformation du modèle est utilisée pour le mappage des modèles entre les différents domaines pour les analyser ou pour la génération automatique de code à partir d'eux-mêmes [37].

### 3.2.1 Définition

Une transformation est la génération automatique d'un modèle cible à partir d'un modèle source, selon une définition de transformation.

Une définition de transformation est un ensemble de règles de transformation qui décrivent comment un modèle source peut être transformé en un modèle cible [38].

### 3.2.2 Type de Transformation

On peut distinguer deux types de transformation de modèle sur la base de méta-modèle dans lequel le modèle source et le modèle cible d'une transformation sont exprimés [38] :

1. **Endogènes :** C'est une transformation entre modèles exprimées dans le même méta-modèle.
2. **Exogènes :** C'est une transformation entre modèles exprimées dans différents méta-modèles.

On peut aussi distinguer deux types de transformation de modèle sur la base de méta-niveau dans lequel le modèle source et cible résident :

1. **Horizontal :** Une transformation horizontale est une transformation où les modèles sources et cibles résident au même niveau d'abstraction.
2. **Verticale :** Une transformation verticale est une transformation où les modèles sources et cibles des modèles résident à des niveaux d'abstractions différents.

Une transformation de modèle peut également avoir plusieurs modèles sources et plusieurs modèles cibles. La transformation de modèles a la caractéristique d'être elle même un modèle puisque elle doit être conforme à un méta-modèle donné [38].

### 3.2.3 Caractéristiques des approches de transformation

Les approches de transformation de modèle sont généralement caractérisée par :

#### 3.2.3.1 Règle de transformation

Une règle de transformation se compose de deux parties : une partie gauche *"left-hand side" (LHS)* et une partie droite *"right-hand side" (RHS).*

La partie gauche *LHS* accède au modèle source, tandis que la partie droite se développe dans le modèle cible. Les deux *LHS* et *RHS* peuvent être représentés en utilisant n'importe quelle mixture de ce qui suit [39] :

- ***Variables*** **:** Les variables conservent des éléments des modèles sources et /ou des modèles cibles (ou certains éléments intermédiaires). Ils sont parfois appelés méta-variables pour les distinguer des variables qui peuvent faire partie du modèle transformé (par exemple, les variables Java dans les programmes Java transformés).
- ***Patterns*** **:** Les patterns sont des fragments de modèle avec zéro ou plusieurs variables. Nous pouvons avoir une chaîne de caractères, terme, et des patterns de graphe. La transformation de type modèle à un modèle, généralement utilise des termes ou des patterns de graphe.
- **Logique :** Elle exprime des calculs et des contraintes sur les éléments du modèle. Elle peut être non-exécutable ou exécutable. La logique non-exécutable est utilisée pour spécifier une relation entre les modèles. La logique exécutable peut prendre une forme déclarative ou impérative. Elle permet de récupérer des éléments du modèle source et la création implicite d'éléments cible à travers des contraintes.

#### 3.2.3.2 Ordonnancement de Règle

Les mécanismes d'ordonnancement déterminent l'ordre dans lequel les règles individuelles sont appliquées. Les mécanismes de planification peuvent varier dans quatre domaines principaux [39] :

- ***Forme :*** L'aspect d'ordonnancement peut être exprimé implicitement ou explicitement. L'ordonnancement implicite implique que l'utilisateur n'a aucun contrôle explicite sur l'algorithme d'ordonnancement défini par l'outil. La seule façon qu'un utilisateur peut influer sur l'algorithme d'ordonnancement défini par le système consiste à concevoir les patterns et les logiques des règles d'une manière à garantir un certain ordre d'exécution.

  L'ordonnancement explicite a consacré des constructions pour contrôler explicitement l'ordre d'exécution. L'ordonnancement explicite pourrait être interne ou externe. Dans l'ordonnancement externe, il y a une séparation claire entre les règles et la logique d'ordonnancement. En revanche, l'ordonnancement interne serait un mécanisme permettant à une règle de transformation d'invoquer directement d'autres règles.

- ***Sélection de Règle :*** Les règles peuvent être sélectionnées par une condition explicite. Certaines approches permettent le choix non-déterministe. Alternativement, un mécanisme de résolution des conflits fondés sur les priorités pourraient être fournis. La sélection des règles interactive est également possible.
- ***Itération des règles :*** Les mécanismes d'itération des règles incluent la récursivité, bouclage.
- ***en Phase :*** Le processus de transformation peut être organisé en plusieurs phases, où chaque phase a un objectif précis. Seulement certaines règles peuvent être invoquées dans une phase donnée.

D'autres caractéristiques ne sont pas abordées, telles que la stratégie d'application des règles, relations entre source et cible, l'organisation des règles,...etc. Pour plus de détails voir [39].

## 3.3 Mécanismes de Transformation

On distingue deux ensembles de mécanismes de transformation :

1. Les mécanismes de transformation qui reposent sur une approche déclarative. Elles se concentrent sur ce qui doit être transformé et en quoi doit être transformé.
2. Les mécanismes de transformation qui reposent sur une approche opérationnelle (impérative). Elle se concentrent sur la façon dont la transformation doit être effectuée elle-même.

Les approches déclaratives comprennent les mécanismes suivants :

1. Programmation fonctionnelle.
2. Programmation logique.
3. Transformation de graphe.
4. ...etc.

Pour notre méthode de transformation on s'intéresse par la transformation de graphe.

### 3.3.1 Transformation de graphe

Les graphes sont bien connus, bien compris et fréquemment utilisés comme moyens de représentation des objets complexes, des diagrammes et des réseaux, tel que les diagrammes entités-relations, les réseaux de Petri et ainsi de suite [40].

La transformation de Graphe a évolué à l'origine en réaction à des lacunes dans l'expressivité des approches classiques de réécriture, comme les grammaires de Chomsky. Elle permet de traiter le cas des structures non-linéaires [41].

Cette catégorie d'approches de transformation de modèle s'appuie sur les travaux théoriques effectués sur les transformations de graphes, en particulier, les approches fonctionnant sur les graphes typés, attribués et étiquetés.

Les règles de transformation de graphe sont composées d'un pattern graphique LHS et d'un pattern graphique RHS. Les patterns de graphe peuvent être rendus dans la syntaxe concrète de leurs sources ou cible respectives.

Les patterns LHS sont appariés dans le modèle en cours de transformation et remplacés par les patterns RHS en place. Les LHS contiennent souvent des conditions en plus des patterns LHS. Certain logique supplémentaire est nécessaire afin de calculer les valeurs d'attribut cible (tels que les noms des éléments) [39].

Parmi les outils permettant la transformation de graphe : *VIATRA*[1], $AToM^3$, *GreAT*[2], *UMLX*[3], et *AGG*[4], ...etc. Nous nous intéressons dans notre méthode par l'outil $AToM^3$.

## 3.4 $AToM^3$

$AToM^3$ *(A Tool for Multi-formalism and Meta-Modelling)* est un outil pour la modélisation multi-paradigme développé dans le laboratoire MSDL *(Modelling, Simulation and Design Lab)* de l'institut d'informatique à l'université de McGill Montréal, Canada. Il est développé avec le langage *Python*[5] en collaboration avec le professeur Juan de Lara de l'université Autónoma de Madrid *(UAM)*, Espagne [42].

$AToM^3$ est développé pour satisfaire deux fonctionnalités principales qui sont

1. La méta-modélisation.
2. La transformation des modèles.

Les formalismes et les modèles dans $AToM^3$ sont décrits graphiquement. À partir d'une méta-spécification (exemple : dans le formalisme Entité-relation) d'un formalisme, $AToM^3$ génère un outil pour manipuler visuellement (créer et modifier) les modèles décrits dans le formalisme spécifié. Les transformations des modèles sont réalisées par la réécriture des graphes, qui peuvent être exprimées d'une manière déclarative comme un modèle de grammaire de graphes [43].

Les méta-modèles permettent de construire des modèles valides dans un certain formalisme et les méta-méta-modèles sont utilisés pour décrire les formalismes eux-mêmes. $AToM^3$ traite les modèles de la même manière dans n'importe quel méta-niveau. L'idée principale d'$AToM^3$ est : *"tout est un modèle"* [44].

Certains méta-modèles sont disponibles avec $AToM^3$ :

- ❑ Entité-relation (Entity-Relationship).
- ❑ Réseaux de Petri (Petri Nets).
- ❑ Diagrammes de Classes (Class Diagrams).
- ❑ ...etc.

1. VIsual Automated model TRAnsformations, http ://eclipse.org/gmt/VIATRA2/
2. Graph Rewriting and Transformation
3. Executable UML
4. Attributed Graph Grammar System
5. *http ://www.python.org/*

Dans notre approche nous utilisons le formalisme " *diagramme de classe* ".

La figure 3.1 illustre l'interface d'$ATOM^3$ avec le formalisme de diagramme de classe chargé.

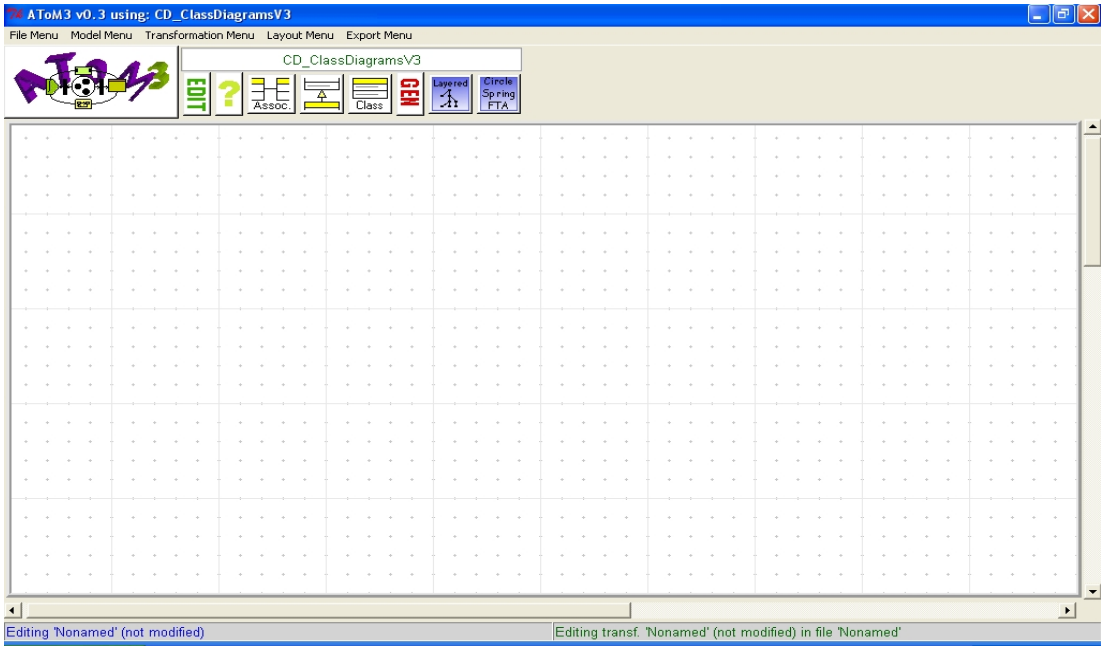



FIGURE 3.1 – Interface d'$ATOM^3$

### 3.4.1 Formalisme Diagrammes de Classes dans $ATOM^3$

Dans $ATOM^3$ les méta-modèles peuvent être construites à partir des Classes et des relations. La description des classes et des relations d'associations se compose de :

- ❑ Nom
- ❑ Attributs
- ❑ Contraintes
- ❑ Action
- ❑ Cardinalités
- ❑ Apparence.

L'éditeur illustré par la figure 3.2 permet de manipuler ces propriétés.

#### 3.4.1.1 Contraintes

Les contraintes peuvent être spécifiées comme des expressions OCL (*Object Constraint Language*) ou Python [43]. Elles peuvent êtres locales associées à une entité, ou globales.

Elles ont les propriétés suivantes :

- ❑ Un nom de contrainte
- ❑ Un événement déclencheur : il peut être soit
  - ➥ sémantique tel que la sauvegarde d'un modèle,...etc.
  - ➥ graphique ou structurel, tel que le déplacement ou la sélection d'une entité.

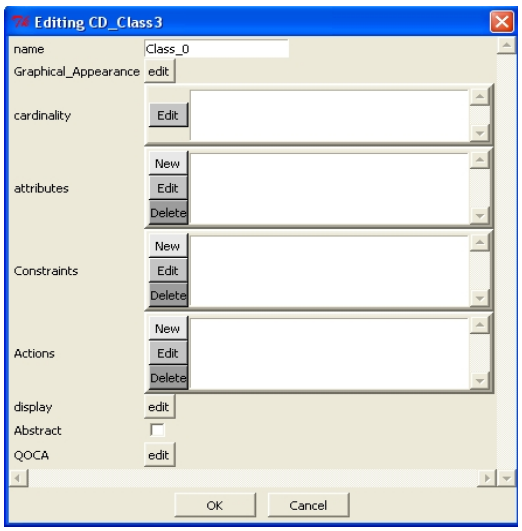



FIGURE 3.2 – Éditeur des propriétés

- ❑ L'évaluation soit :
  - ➥ avant l'événement (pré-condition) ou
  - ➥ après (post-condition)
- ❑ Le code (soit en OCL ou Python).

La figure 3.3 illustre l'éditeur des contraintes dans $ATOM^3$.

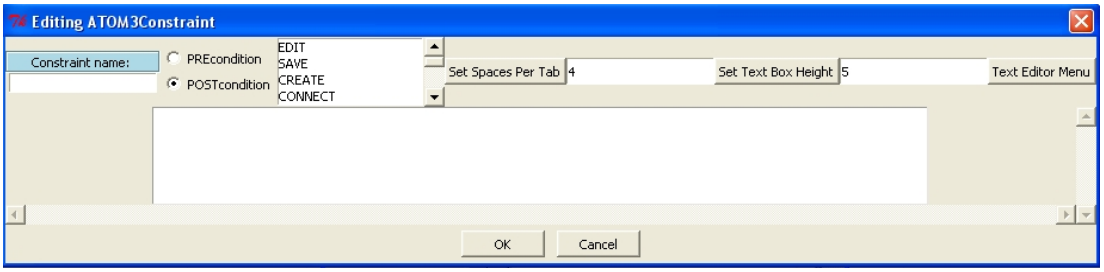



FIGURE 3.3 – Éditeur de contraintes

#### 3.4.1.2 Action

Une action est similaire à une contrainte sauf qu'elle a d'autres effets et elle est un code en Pyton seulement [43].

Elles ont les propriétés suivantes :

- ❑ Un nom d'action.
- ❑ Un événement déclencheur : Il peut être soit
  - ➥ Sémantique tel que la sauvegarde d'un modèle, etc.
  - ➥ Graphique ou structurel, tel que le déplacement ou la sélection d'une entité.
- ❑ L'exécution soit :
  - ➥ Avant l'événement (pré-condition) ou
  - ➥ Après (post-condition)
- ❑ Le code (soit en OCL ou Python).

L'éditeur des actions est similaire à l'éditeur des contraintes illustré par la figure 3.3.

#### 3.4.1.3 Attributs

Les entités (classes et relation d'association) qui doivent apparaître sur les modèles sont spécifiées ensemble avec leurs attributs et leurs apparences graphiques. Par exemple, pour définir le formalisme de réseau de Petri, il est nécessaire de définir à la fois les places et les transitions. En outre, pour les places nous avons besoin d'ajouter l'attribut nom et le nombre de jetons. Pour les transitions, nous avons besoin de spécifier le nom [43].

Deux types d'attributs existent dans $ATOM^3$ :

1. ***Attributs réguliers*** **:** Ils sont utilisés pour identifier les caractéristiques d'une entité.
2. ***Attributs générateurs*** **:** Ils permettent de générer d'autres propriétés.

Il y a deux types de base dans $ATOM^3$ :

- ❑ Type régulier tel que String, Integer, Float, Boolean. . . etc.
- ❑ Type générateur qui permet de générer des attributs, des contraintes, des attributs graphiques.

La figure 3.4 illustre l'éditeur des attributs.

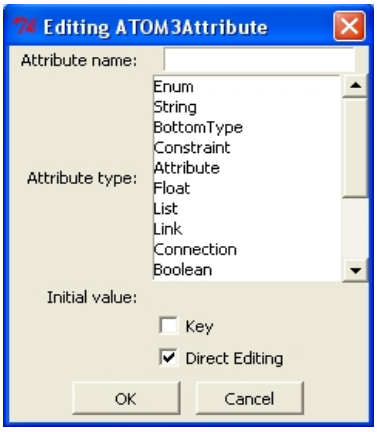



FIGURE 3.4 – Éditeur des attributs.

### 3.4.2 Transformation de Graphes

Dans $ATOM^3$ la grammaire est un modèle caractérisé par

- ❑ Une action initiale.
- ❑ Une action finale.
- ❑ L'ensemble des règles.

L'éditeur de grammaire (figure 3.5) permet l'édition, la génération et l'exécution de la grammaire.

Chaque règle est constituée de :

- ❑ Un nom spécifique pour la règle.
- ❑ Une priorité indiquant l'ordre dans lequel la règle est appliquée.
- ❑ Une partie gauche *(Left Hand Side : LHS)* qui est un graphe.

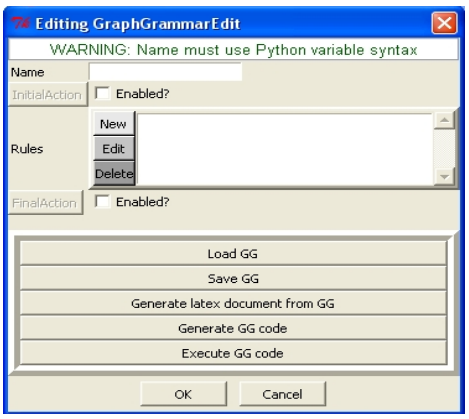



FIGURE 3.5 – Éditeur de grammaire

- ❑ Une partie droite *(Right Hand Side : RHS)* qui peut être un graphe.
- ❑ Une condition *(Un code en Pyton)* qui doit être vérifiée avant que la règle soit appliquée.
- ❑ Une action *(Un code en Pyton)* qui doit être exécutée une fois que la règle soit appliquée.

L'éditeur de règle (figure 3.6) permet l'édition des différentes parties de la règle ainsi que l'action initiale et finale de chaque règle.

L'éditeur de condition et l'éditeur d'action d'une règle sont similaires à l'éditeur des contraintes présenté par la figure 3.3.

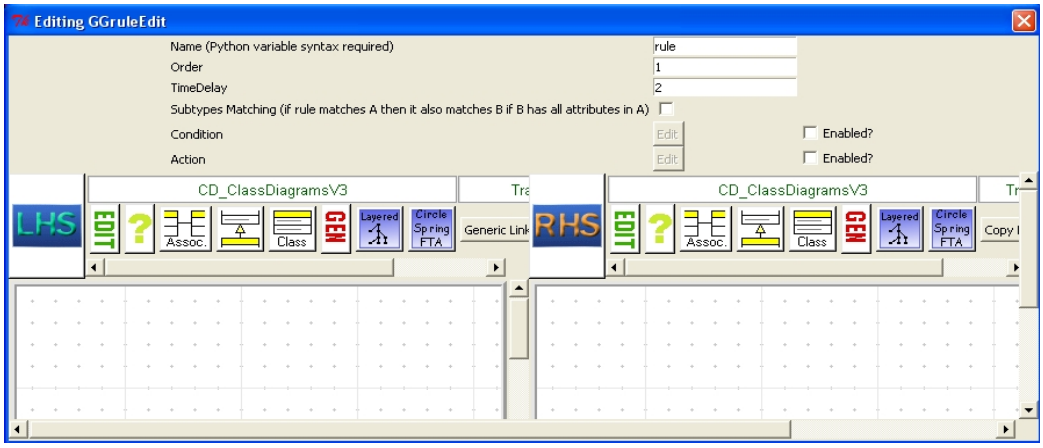



FIGURE 3.6 – Éditeur de règle

## 3.5 Présentation de l'Approche

Pour transformer les diagrammes d'activités mobiles vers les réseaux de Petri imbriqués, on propose une méthode qui s'appuie sur les trois étapes suivantes :

1. Construction de méta-modèle des diagrammes d'activités mobiles.
2. Construction de méta-modèle des réseaux de Petri imbriqués.
3. Définition des règles de la transformation.

### 3.5.1 Méta-Modèle des Diagrammes d'activités

La spécification UML [1] definit des différents niveaux d'activités qui sont :

1. Activités fondamentales (Fundamental Activities).
2. Activités de base (Basic Activities).
3. Activités intermédiaires (Intermediate Activities).
4. Activités complètes (Complete Activities).
5. Les activités structurées (Structured Activities).
6. Activités complète structurées (Complete Structure dActivities).
7. Activités extra structurées (Extra Structured Activities).

Chaque niveau ajoute ses propres constructions pour un domaine particulier, par exemple, les activités structurées adressent la modélisation avec les langages de programmation traditionnelle. Autres niveaux d'activités comme le niveau *"fondamental"* et le niveau *"base"* sont idéaux pour la modélisation de haut niveau et la modélisation des processus métiers.

Les niveaux les plus adaptés à convertir en modèles de réseaux de Petri sont [1, 45] :

- ❑ Activités fondamentales (FundamentalActivities) : Le niveau fondamental définit les activités comme des contenant des nœuds, qui comprend les actions [1].
- ❑ Activités de base (BasicActivities) : Ce niveau comprend les contrôles de séquence et les flux de données entre les actions, mais les contrôles de bifurcation et de joint explicite, ainsi que les les contrôles de décision et de fusion, ne sont pas supporté [1].
- ❑ Activités intermédiaires (IntermediateActivities) : Le niveau intermédiaire prend en charge la modélisation des diagrammes d'activités qui incluent les contrôles de concurrence, les flux de données et les décisions. Il supporte la modélisation similaire aux réseaux de Petri traditionnels avec files d'attente [1].

Ainsi, pour concevoir notre méta-modèle, on a combiné les trois niveaux et les nouveaux concepts définis par les diagrammes d'activités mobiles dans [16]. Nous utilisons $ATOM^3$ pour construire le méta-modèle de diagramme d'activités mobile. Pour cela, on suit les étapes suivantes :

1. Chargement de méta-modèle *"Diagramme de Classe"*. Ce méta-modèle est disponible dans le répertoire des formalismes principaux d'$ATOM^3$.
2. Concevoir le méta-modèle de diagramme d'activités mobile comme il est expliqué précédemment. La figure 3.7 illustre le méta-modèle construit. Il est composé de 09 classes et 08 associations. Ce méta-modèle va être utilisé pour générer un outil de modélisation visuel. Ainsi, nous avons défini pour chaque classe son apparence graphique selon sa spécification standard dans [1] et selon la définition des diagrammes d'activités mobiles dans [16]. Nous donnerons la signification des classes les plus importantes dans le méta-modèle.

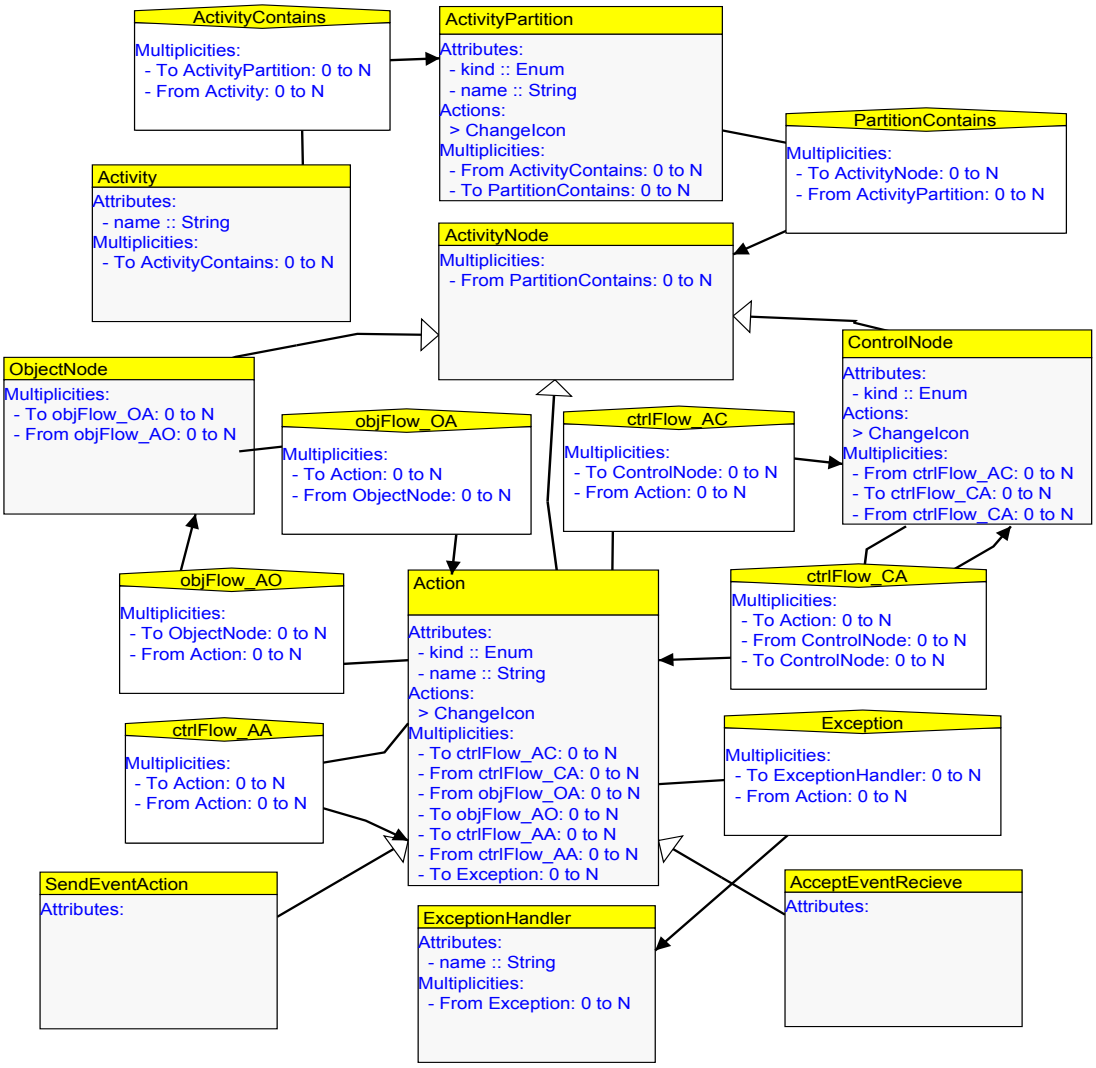



FIGURE 3.7 – Méta-modèle de diagramme d'activités

- ***La classe ActivityPartition*** **:** est la classe qui représente l'élément de modélisation *"Activity Partition"* dans la notation UML, elle a un attribut *"kind"* qui permet de spécifier si la partition représente un agent, ou un hôte. Elle a aussi une action qui permet de changer son apparence graphique selon la valeur de l'attribut *"kind"*. Si la valeur de l'attribut *"kind"* est égale à *"Host"* alors l'apparence de la partition est changée en partition verticale avec le stéréotype *"«Host»"* et si sa valeur est égale à *"agent"* alors l'apparence de la partition est changée en partition horizontale avec le stéréotype *"«Agent»"*.
- ***La classe ControlNode*** **:** est une classe qui représente les nœuds de contrôles dans la notation UML. Elle peut avoir l'apparence graphique des nœuds de contrôles suivants : *Initial, Final, Décision, Fusion, Bifurcation, jointure, flux final.*
- ***La classe Action*** **:** est une classe qui représente soit

  (a) Une action simple.

  (b) Une action avec un broche d'entré ou broches de sortie.

(c) L'élément de modèle AcceptEventTimer.

(d) L'activité spéciale de clonage *"clone action"*.

- ❑ ***La classe SendEventAction*** **:** est une classe qui représente l'action d'envoie de signal, stéréotypé par le stéréotype *"«ACLMessage»"*.
- ❑ ***La classe AcceptEventRecieve*** **:** est la même que la classe SendEventAction sauf quelle représente l'action de réception d'un événement.

3. À partir de ce méta-modèle, $ATOM^3$ génère un outil de modélisation du diagramme d'activités mobile. Cet outil offre un ensemble de boutons de manipulation de ce diagramme, comme le montre la figure 3.8.

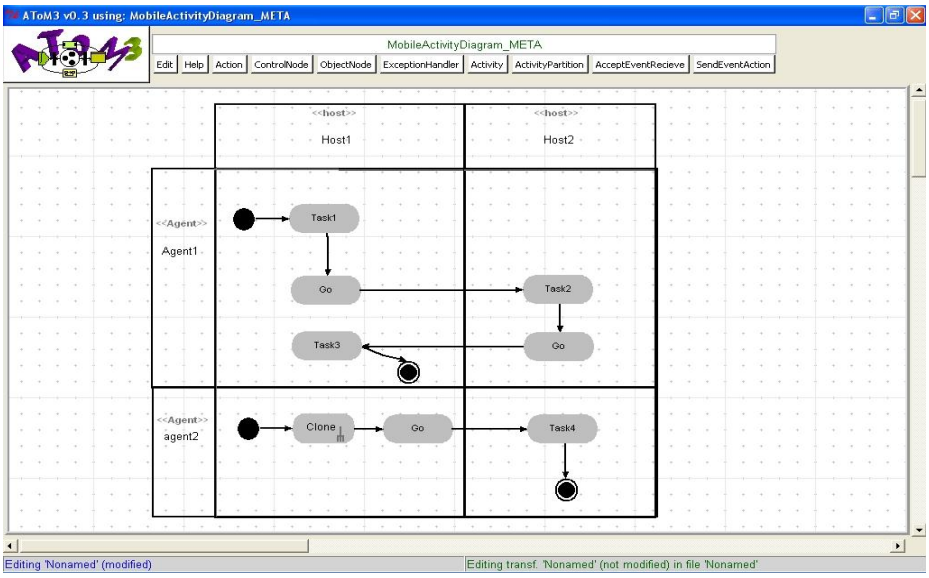



FIGURE 3.8 – Outil de modélisation généré par $ATOM^3$

L'exemple de la figure 3.8 représente un modèle d'un système composé de deux agents "*agent1*" et "*agent2*" qui se déplacent entre deux hôtes "*host1*" et "*host2*".

### 3.5.2 Méta-modèle de réseau de Petri imbriqué

On suit les mêmes étapes de création de méta-modèle de diagramme d'activités mobile pour obtenir le méta-modèle représenté par la figure 3.9

Notre méta-modèle est composé de 07 classes et 08 associations. Il est principalement constitué de deux ensembles de classes. Des classes définissent les éléments réseaux (*EN*) et d'autres définissent le système réseau (*SN*).

1. **La classe *"EN"*** représente un éléments réseau. Ils sont des réseaux de Petri ordinaires. Les classes *"EPlace" et "ETransition" représentent* les places et les transitions de ce réseau de Petri ordinaire la relation de contenance *"ENContain"* permet de les distinguer par rapport au reste de modèle.
2. **La classe "*SN*"** représente un système réseau. Il est un réseau de Petri de haut niveau. Les classes *"SPlace" et "STransition"* représentent les places et les transitions

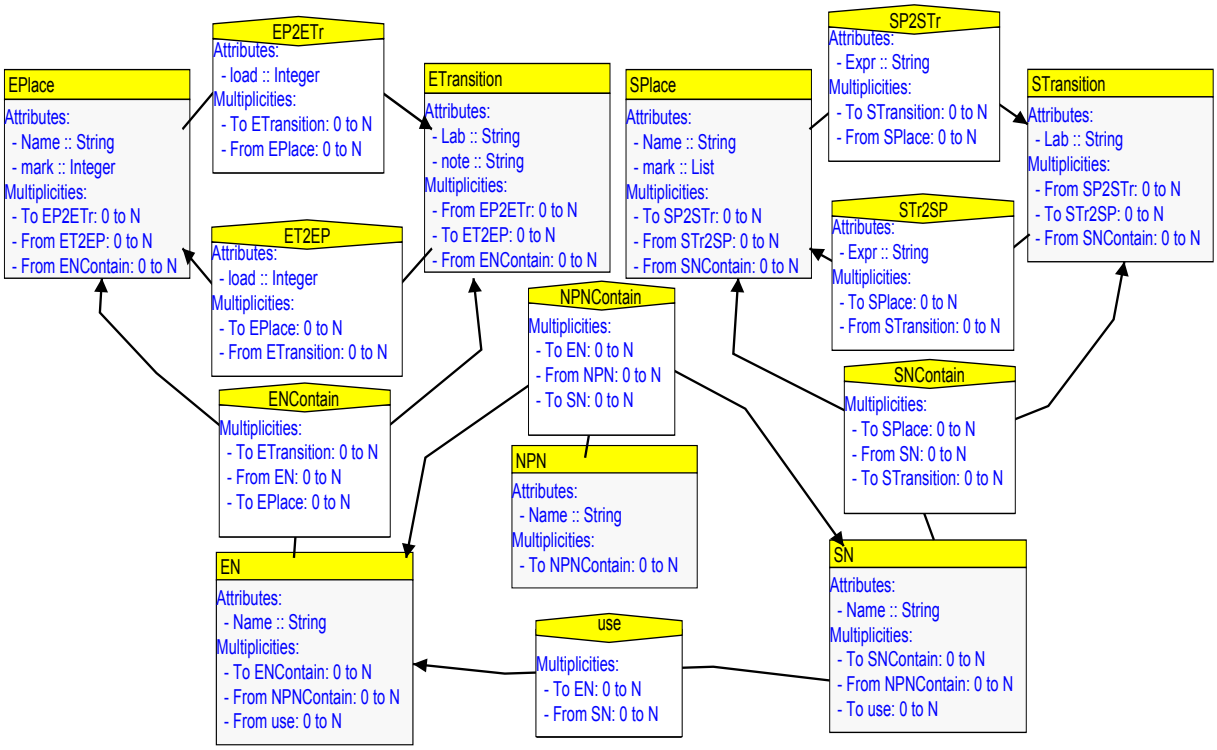



FIGURE 3.9 – Méta-modèle de NPN

du système réseau. Il sont distingués par la classe *"SN"* et la relation de contenance *"SNContain"*. Les place peuvent avoir des jetons de type point noir ou des variables dont le type doit être un des éléments réseaux déjà défini.

Les deux ensemble eux-mêmes ont une relation de contenance avec la classe *"NPN"*. Cette dernière représente le réseau entière dont le nom est contenue dans l'attribut *"Name"*.

L'outil de modélisation généré à partir de méta-modèle de réseau de Petri imbriqué est représenté par la figure 3.10. Dans cette figure on trouve, également, un exemple d'utilisation de l'outil généré.

### 3.5.3 Définition des Règles de Transformation

Une grammaire de graphes est une grammaire constituée d'un ensemble de règles, permettant de transformer des formalismes de même nature ou de nature différente. Chaque règle est composée de deux parties, la partie gauche *(LHS)* et la partie droite *(RHS)*. Chaque partie peut être un sous graphe des formalismes considérés dans la transformation.

Dans notre travail, les formalismes considérés dans la transformation sont le formalisme diagramme d'activités mobile comme graphe source à transformer et le formalisme de réseau de Petri imbriqué comme graphe destinataire de transformation.

Cette grammaire est définie en utilisant l'outil $AToM^3$ selon les étapes suivantes :

1. Charger les deux méta-modèles définis précédemment qui sont le méta-modèle de diagramme d'activités mobile et le méta-modèle de réseau de Petri imbriqué.

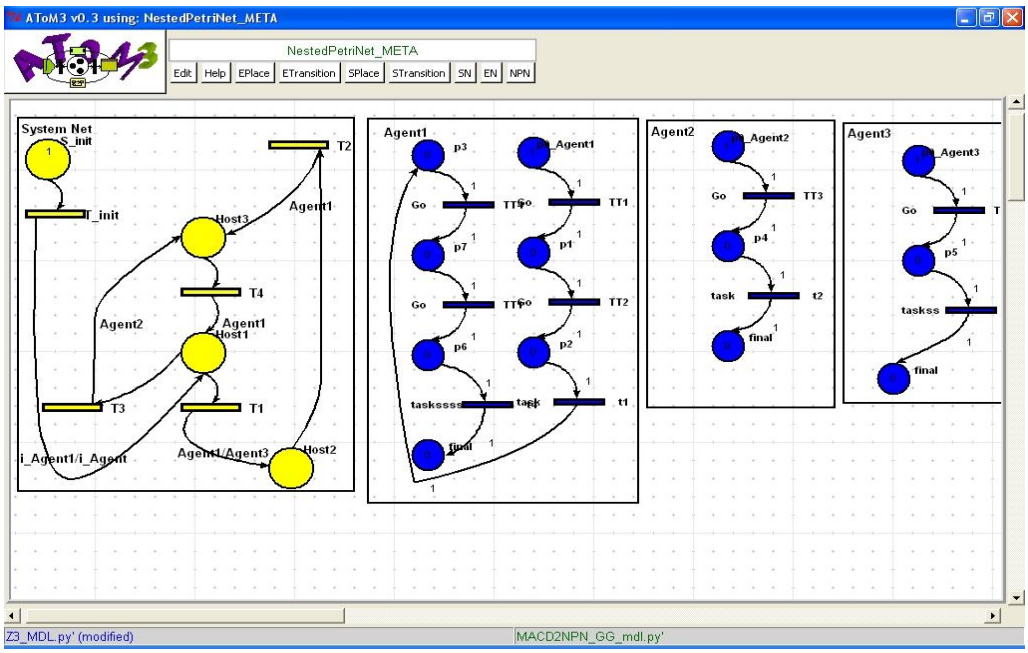



FIGURE 3.10 – Outil de modélisation de Réseau de Petri généré par $ATOM^3$

2. Définir les règles de la grammaire
3. Générer le fichier exécutable de la grammaire.

#### 3.5.3.1 Grammaire de Graphes

Notre grammaire est composée de trois parties :

- ***Action initiale :*** Cette partie de notre grammaire sert à initialiser l'ensemble des variables globales utilisées. Ils sont utilisés pour satisfaire divers besoins. Par exemple, la variable *"visited"* est utilisé pour marquer qu'un nœud dans le graphe est déjà traité. La figure 3.11 représente le code de l'action initiale dans l'éditeur des contraintes.

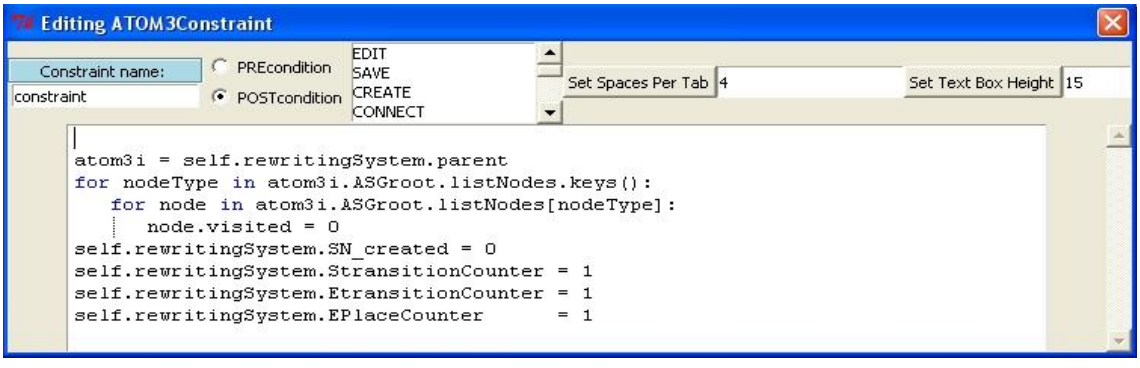



FIGURE 3.11 – Action Initiale

- ***Action finale :*** À l'inverse de l'action initiale, l'action finale permet de libérer les variable globales. La figure 3.12 représente le code de l'action finale dans l'éditeur des contraintes.

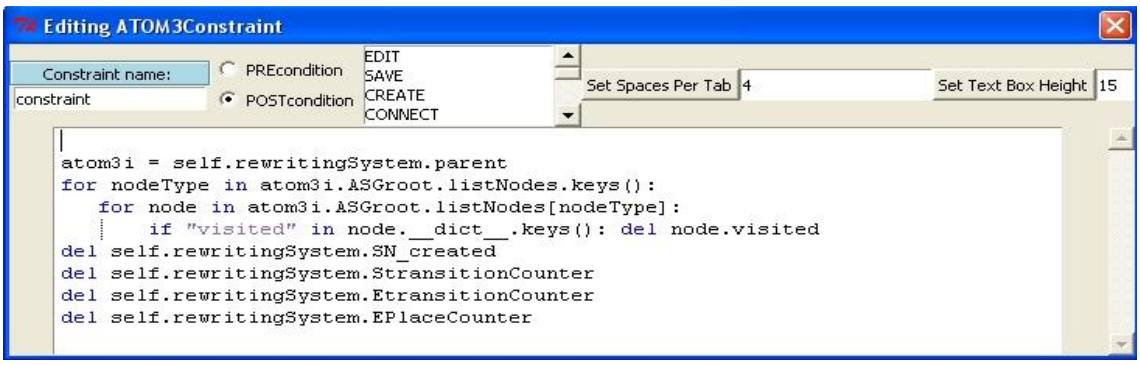



FIGURE 3.12 – Action Finale

- ***Ensemble de règles :*** Notre grammaire graphes est composée de soixante-et-un règle. Chaque règle est caractérisée par un nom et une priorité d'exécution. Elles sont classés en 04 catégories :
  1. Règles pour créer le système réseau.
  2. Règles pour créer les éléments réseau.
  3. Règles pour synchroniser entre les éléments réseau et le système réseau.
  4. Règles pour le nettoyage du résultat de la transformation.

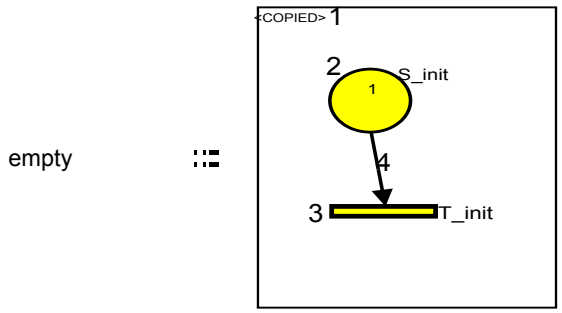
L'idée que nous avons suivi pour la transformation des éléments de modèle dans notre grammaire est que :

- La création de système réseau revient à identifier les places, les transitions et les valeurs des expressions des arcs où :
  Chaque partition de type *"host"* représente une place dans le système réseau. Une action mobile entre deux partition représente un arc reliant la place source et la place destinataire. L'expression de l'arc reliant la transition à la place est, principalement, construite des variables et des constantes des éléments réseau faisant l'action mobile.
- La création des éléments réseau revient à transformer chaque partition de type *"agent"* vers un élément réseau.
- Chaque action mobile, dans le diagramme d'activités mobile représente une synchronisation verticale dans le réseau de Petri imbriqué, car elle permet de déplacer un agent d'un hôte représenté par une place dans le système réseau vers d'autres hôtes représentés par d'autres places dans le système réseau.
- Les éléments de modèle ordinaires sont transformés simplement on s'inspirants de [45].

Nous citerons ici les règles les plus importantes :

***1) SystemNetCreate (priorité 1) :*** Cette règle (figure 3.13 ) a la priorité égale à *"1"*, c à d, elle est la première règle appliquée dans la grammaire. Elle permet de créer le système réseau *(SN)* de réseau de Petri imbriqué, avec un nom par défaut "*system net*".

Cette règle permet aussi de créer la place et la transition initiale du système réseau, dont le nom est *"S_init"* et l'étiquette est *"T_init"*.




(a) LHS (b) RHS

FIGURE 3.13 – Règle SystemNetCreate

***(a) condition :*** Pour chaque réseau de Petri imbriqué, il y a un seul système réseau. Cette contrainte est interprétée par une condition qui vérifie si la règle n'est pas traitée auparavant. Le code de cette condition est comme le montre la figure 3.14.

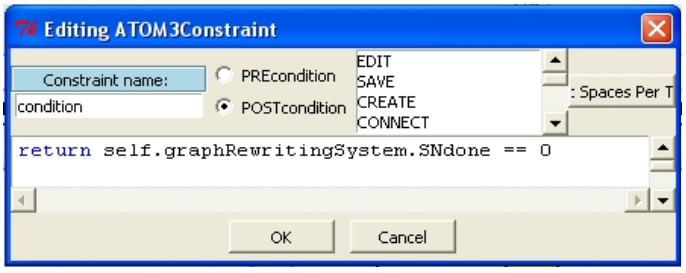



FIGURE 3.14 – Condition d'application de la règle *SystemNetCreate*

***(b) Action :*** Après l'application de la règle, une action est exécutée permettant de marquer que la règle est exécuté. La figure 3.15 illustre cette action.

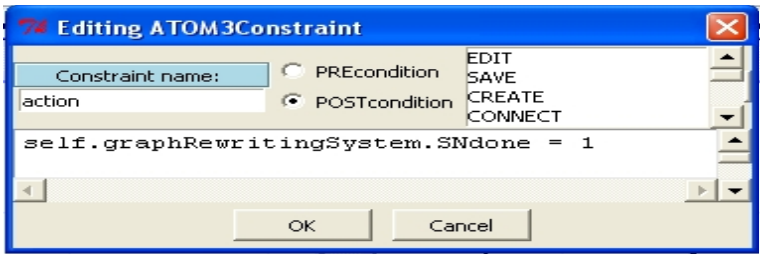
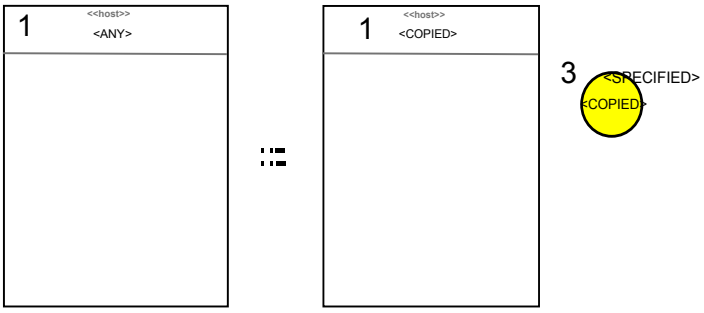



FIGURE 3.15 – Action de la règle *SystemNetCreate*

***2) Host2SPlace (priorité 2 ) :*** L'application de cette règle (figure 3.16) permet d'identifier et de créer les places de système réseau *(SN)*. Chaque partition de type *"host"* Correspond à une place dans le système réseau. Le même nom de la partition de type *"host"* est attribué à la place identifiée.

Chaque place représente un emplacement possible dans lequel les agents pourraient évoluer. Le nombre de jetons sur toues les places transformées est égal à *"0"*.




(a) LHS (b) RHS

FIGURE 3.16 – Règle Host2SPlace

***(a) condition :*** Pour l'application de cette règle, il faut vérifier que la partition à transformer n'est pas traitée auparavant. Pour cela, nous avons défini une condition pour cette règle dont le code est représenté par la figure 3.17

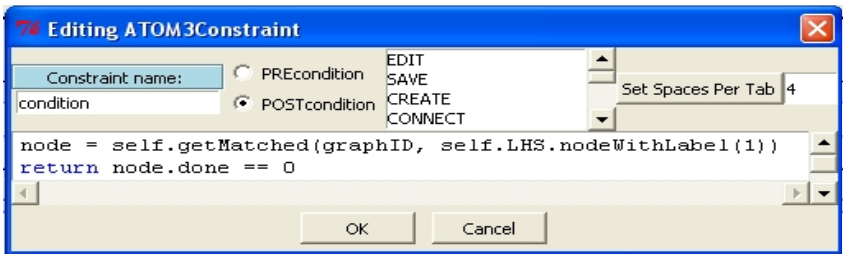



FIGURE 3.17 – Condition d'application de la règle *Host2SPlace*

***(b) Action :*** Une fois la règle est appliquée, son action permet de marquer que l'objet de la partie gauche est traité. La figure 3.18 illustre cette action.

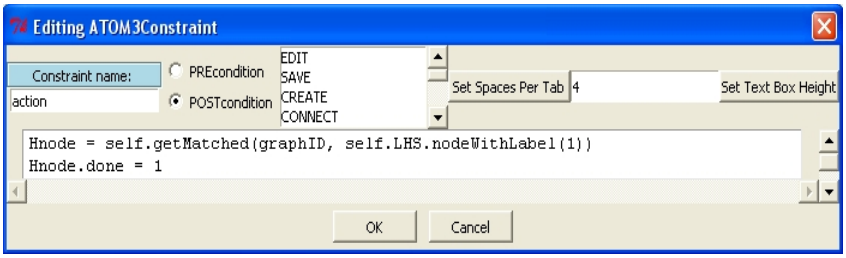
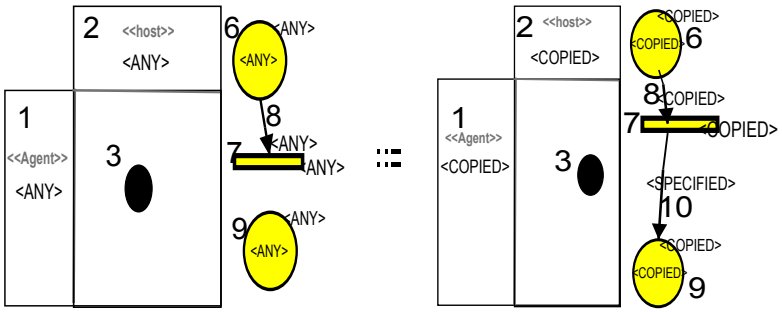



FIGURE 3.18 – Action de la règle *Host2SPlace*

***(3) SPlace2Init_2 (priorité 4 ) :*** La transition initiale représente le point du départ de système. À cet état de système, aucun élément réseau *(agent)* n'existe .Le franchissement de cette transition crée tous les éléments réseau qui pourraient exister à la première exécution du système et les met à leurs places. Donc chaque place contenant

un agent à l'état initial doit être reliée avec la transition initiale. Par exemple, si on a un système constitué de 03 places dans le système réseau. À l'état initiale ce système contient 02 agent, *"agent1"* dans la place *"place1"* et *"agent2"* dans la place *"place3"* alors les place *"place1"* et *"place3"* doivent être reliées avec la transition initiale.




(a) LHS (b) RHS

FIGURE 3.19 – Règle SPlace2Init_2

L'application de cette règle (figure 3.19) permet d'identifier les places de système réseau qui doivent être reliées avec la transition initiale du système réseau. Chaque partition de type "*host*" contenant un nœud de contrôle initial correspond à une place dans le système réseau qui satisfait les conditions pour être relier avec la transition initiale.

Elle crée un arc entre la place identifiée et la transition initiale. L'expression de l'arc créé sera la concaténation des lettres "i_" et le nom de l'élément reseau.

***(a) condition :*** On utilise l'éditeur de condition, comme on a fait avec les règles précédentes pour spécifier la condition. Le code de cette condition est exprimé en python comme le présente la figure 3.20.

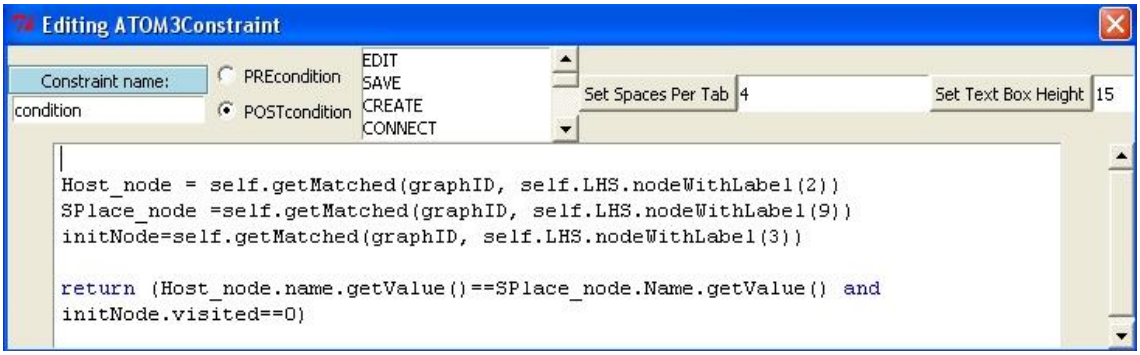
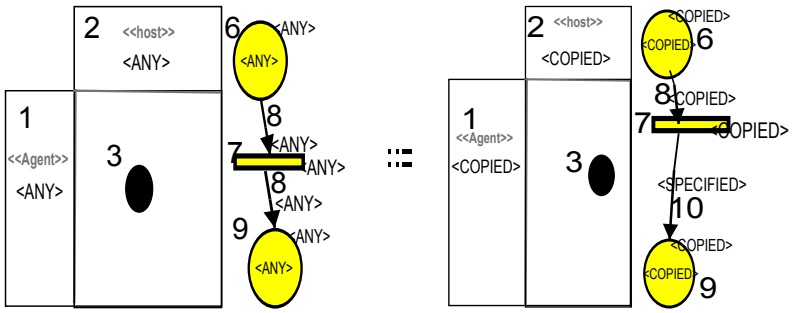



FIGURE 3.20 – Condition d'application de la règle *SPlace2Init_2*

***(b) Action :*** L'action permet de marquer que l'objet de la partie gauche est traité. Le code de cette action et le même que celui de l'action de la règle N°2 (figure 3.18).

***(4) SPlace2Init_1 (priorité 3 ) :*** Elle est représentée par la figure 3.21. Elle est la même que la règle précédente (*"SPlace2Init_1"*) sauf que les arcs entre les places du système réseau et la transition initiale sont déjà crées.

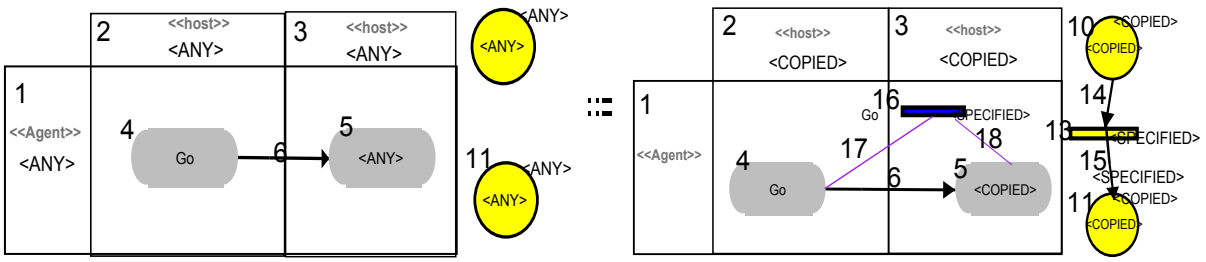



(a) LHS (b) RHS

FIGURE 3.21 – *Règle* SPlace2Init_*1*

***(5) Go2VerticalSync_2 (priorité 6 ) :*** Chaque action mobile (action *"Go"*) représente Un déplacement d'un agent entre un hôte source vers un hôte destinataire. L'agent faisant le déplacement représente un élément réseau et l'action *"Go"* représente une transition dans cet élément réseau. Les hôtes représentent des places dans le système réseau. Ainsi, l'application de cette règle (figure 3.22) permet de :

1. Rechercher l'action mobile *"Go"* et la transformer à une transition dans l'élément réseau impliqué par l'action mobile "Go". Le nom de cette transition est *"Go"*.
2. Créer une transition dans le système réseau entre les places correspondantes aux hôtes sources et destinataire de l'action mobile. L'expression de L'arc créé dans le système réseau portera le nom de l'agent qui avait fait l'action mobile *"Go"*.
3. Donner des étiquettes adjacentes aux transitions créés pour qu'elles soient synchronisées verticalement. Par exemple si la transition est *"T1"* la transition adjacente sera *"TT1"*.




(a) LHS (b) RHS

FIGURE 3.22 – Règle Go2VerticalSync_2

***(a) condition :*** Le code de la condition d'application pour cette règle est comme le présente la figure 3.23.

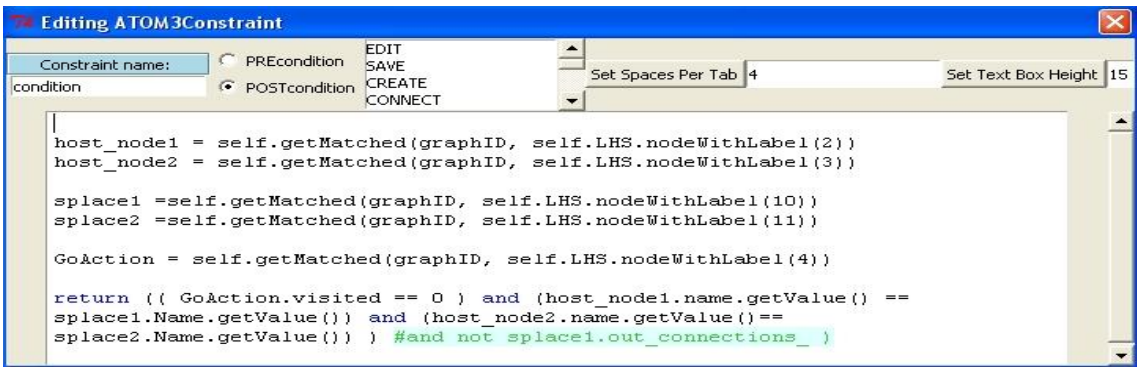



FIGURE 3.23 – Condition d'application de la règle *Go2VerticalSync_2*

***(b) Action :*** L'action de cette règle est représenté par la figure 3.24.

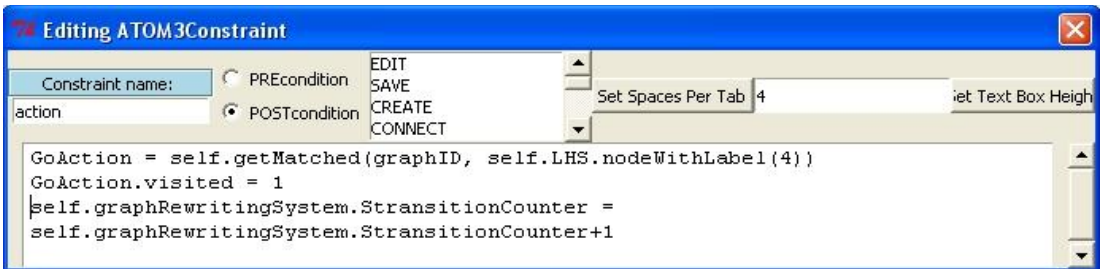
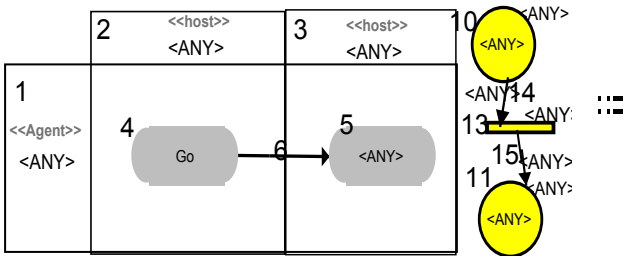
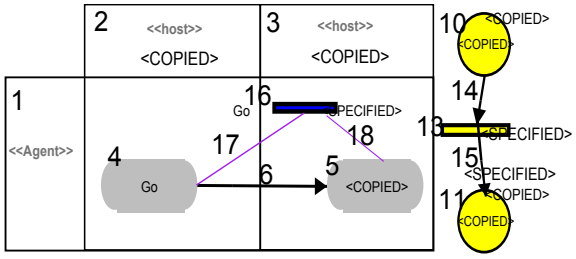



FIGURE 3.24 – Action de la règle *Go2VerticalSync_2*

***(6) Go2VerticalSync_1 (priorité 5 ) :*** Elle est représentée par la figure 3.25. Elle est la même que la règle précédente (*"Go2VerticalSync_2"*) sauf que la transition de synchronisation dans le système réseau est déjà crée.




(a) LHS



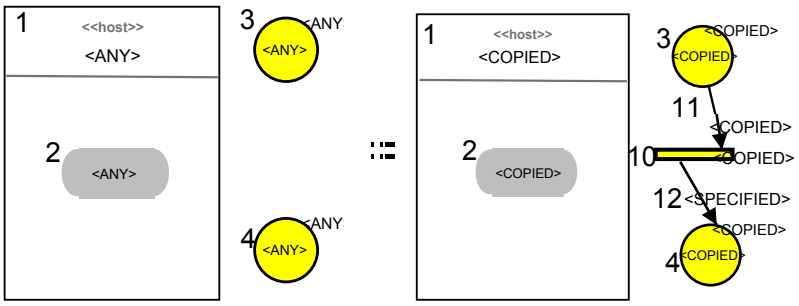

(b) RHS

FIGURE 3.25 – Règle Go2VerticalSync_1

L'action et la condition de cette règle sont aussi les même que celles de la règle

(*"Go2VerticalSync_2"*).

***(7) Clone2TransferStep_2 (priorité 8 ) :*** Cette règle nous permet de créer une étape de Transfer dans le système réseau pour créer l'agent à cloner et le transférer ver l'hôte destinataire. Le nom de l'hôte destinataire et l'agent à cloner sont extraits du nom de l'activité de clonage. Cette règle est représentée par la figure 3.26



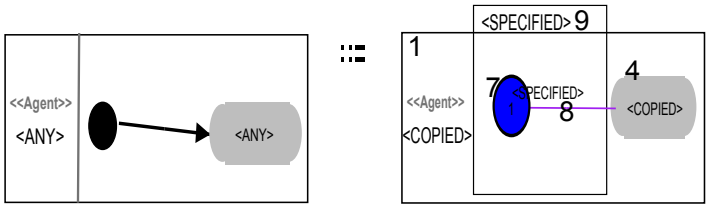

(a) LHS (b) RHS

FIGURE 3.26 – Règle Clone2TransferStep_2

***(a) condition :*** Pour appliquer cette règle il faut vérifier que les places dans le système réseaux sont celles impliquées dans la création de l'étape de Transfer. Le code de cette condition est le suivant :

***(b) Action :*** L'action de cette règle est la même que l'action de la règle *Go2VerticalSync_2*.

***(8) ElementNetCreate (priorité 9) :*** Après la construction du système réseau et l'identification des transitions de synchronisation, il ne reste que de construire les éléments réseau. Ainsi, l'application de cette règle (figure 3.27) permet de les identifier.




(a) LHS (b) RHS

FIGURE 3.27 – Règle ElementNetCreate

Cette règle nous permet de transformer une partition de type *"agent"* sur le diagramme d'activités mobile à un élément réseau. Le nom de l'élément réseau créé est le même que celui de l'agent.

Elle nous permet, également, de transformer le nœud d'activité initial contenu dans cette partition à une place initiale pour l'élément créé. Le nom de la place initiale est

obtenu par la concaténation des lettres "p0_" et le nom de l'agent. Le nombre des jetons est égal à *"1"*. Les éléments réseau créés sont utilisés par le système réseau comme jetons. Cette règle n'a ni condition, ni action.

Ce qui reste de la construction des éléments réseau est une transformation simple d'élément de modèle du diagramme d'activités d'UML 2.0 vers les réseaux de Petri ordinaires. On s y inspirais de [45] dans la construction de nos règles. Nous présentons dans ce qui suit un sous ensemble de ces règles. Les règles que nous n'avons pas vu ici serons présentées dans l'annexe A.

La transformation naturelle des éléments de modèle du diagramme d'activités consiste à transformer chaque action ou élément similaire à l'action en transition et à transformer le reste des éléments en places. Ce qui relie ces éléments est transformé en arc entre les places et les transitions de réseau de Petri ordinaire (*elément réseau* ).

Cependant, des exceptions peuvent être engendrées si par exemple un nœud de contrôle est relié avec un autre nœud de control, alors que le réseau de Petri ne permet pas de relier une place avec une place. Ce genre d'exception est résolu en ajoutant des places ou des transitions intermédiaires. L'ensemble des règles ont des priorités d'exécution qui leur permettent de suivre les étapes suivante :

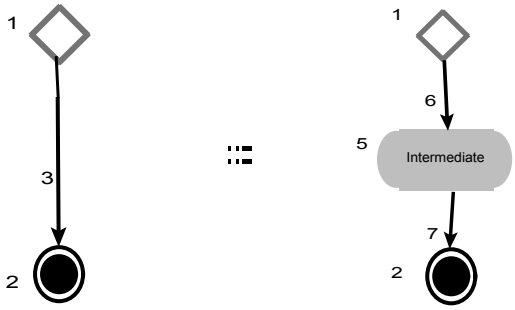
1. Traiter les exceptions par ajout des actions et des places intermédiaires.
2. Transformer les éléments de modèle.
3. Nettoyage de résultat.

***(10) Decision2Decision (priorité 9) :*** Cette règle (figure 3.28) est une des règles qui traitent les exceptions. Elle permet d'éliminer l'exception d'avoir un nœud de contrôle de type "*décision*" ou "*fusion*" relié avec un autre nœud de contrôle aussi de type "*fusion*" ou "*décision*". L'exception est éliminée par ajout d'une action dont le nom est "*intermediate*". Elle n'a ni condition ni action.


(a) LHS (b) RHS

FIGURE 3.28 – Règle Decision2Decision

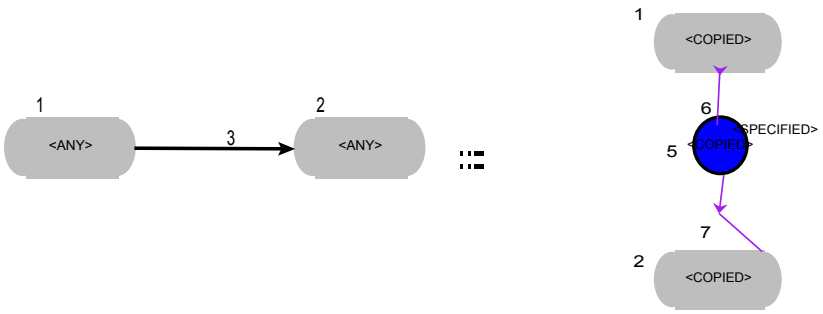
***(11) Decision2Final (priorité 9) :*** Cette règle (figure 3.29) est la même que la règle précédente sauf que les nœuds de contrôles reliés sont les nœuds de contrôles "*fusion*", "*décision*" avec un nœud de contrôle de type "*final*".




(a) LHS (b) RHS

FIGURE 3.29 – Règle Decision2Final

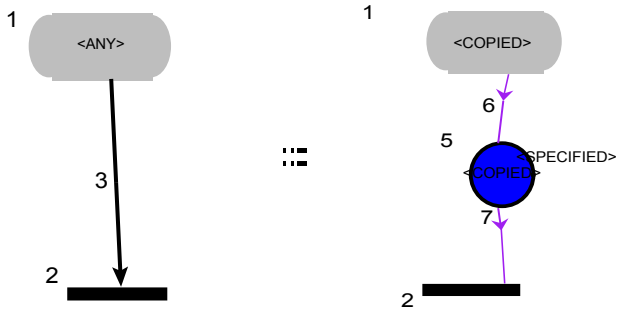
***(12) Action2Action (priorité 9) :*** Cette règle (figure 3.30) nous permet de transformer le flux de contrôle entre les actions à une place dans l'élément réseau correspondant. Le nom de cette place est le nom par défaut généré par la règle.




(a) LHS (b) RHS

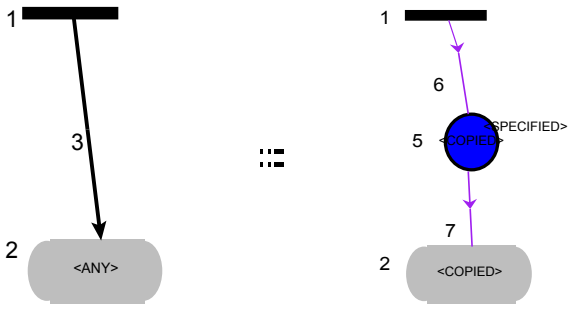
FIGURE 3.30 – Règle Action2Action

***(13) Action2Join, Join2Action (priorité 31) :*** Ces deux règles (figure 3.31, 3.32) nous permettent de transformer le flux de contrôle entre une action et un nœud de contrôle de synchronisation à une place dans l'élément réseau correspondant. Le nom de cette place est le nom par défaut généré par la règle.




(a) LHS (b) RHS

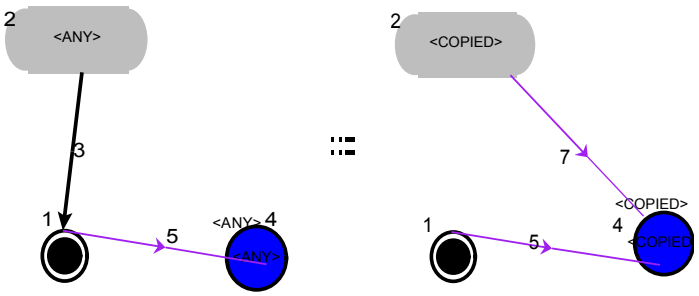
FIGURE 3.31 – Règle Action2Join




(a) LHS (b) RHS

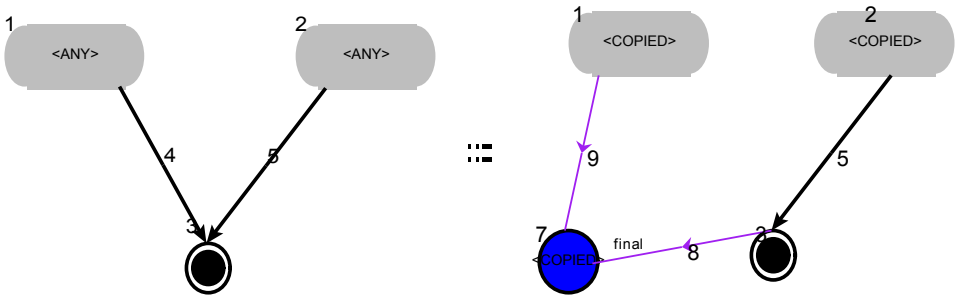
FIGURE 3.32 – Règle Join2Action

***(14) Action2Final :*** Les règles représentées par les figures 3.33, 3.34, 3.35, 3.36, permettent de transformer le flux de contrôle entre les actions et un nœud de contrôle de type "*final*" à une place dans l'élément réseau correspondant. Le nom de cette place est le nom par défaut généré par la règle. Elles ont les priorités suivante 42, 43, 44, 45.



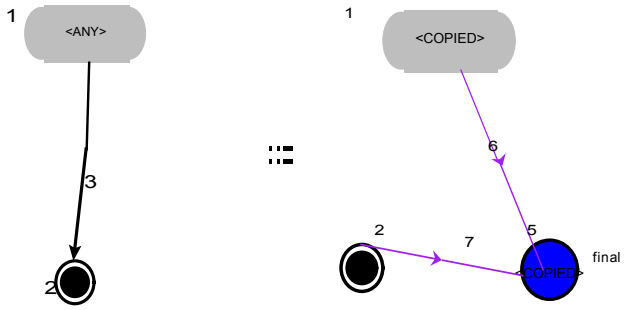

(a) LHS (b) RHS

FIGURE 3.33 – Règle Action2Final_1




(a) LHS (b) RHS

FIGURE 3.34 – Règle Action2Final_2



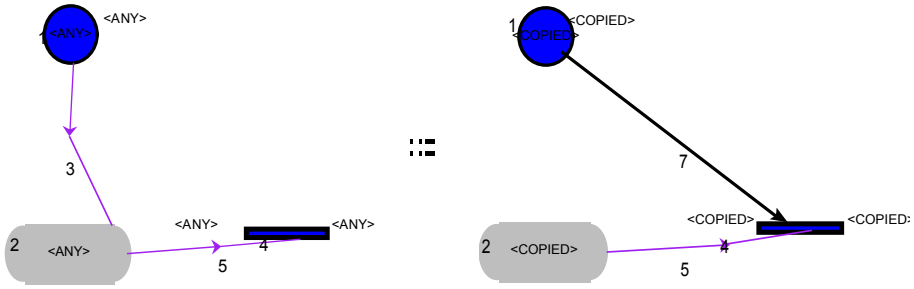

(a) LHS (b) RHS

FIGURE 3.35 – Règle Action2Final_3


(a) LHS (b) RHS

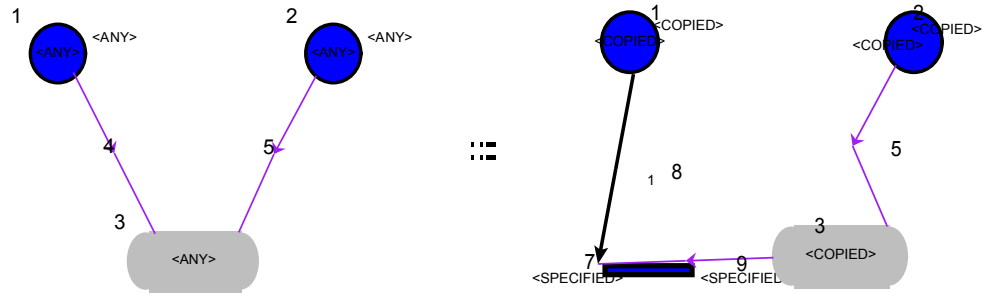
FIGURE 3.36 – Règle Action2Final_4

***(15) Eplace2Action :*** Les règles représentées par les figures 3.37, 3.38, 3.39, 3.40, et 3.41 permettent de transformer les actions reliées avec des places d'un élément réseaux à une transition. Le nom de cette transition est le nom de cette action. Elles ont les priorités suivantes : 50, 51, 52, 53



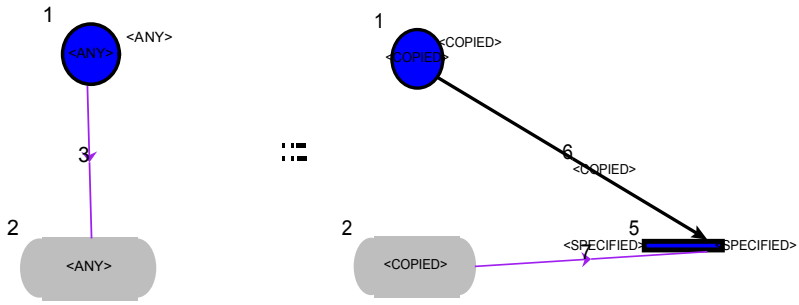

(a) LHS (b) RHS

FIGURE 3.37 – Règle Eplace2Action_1



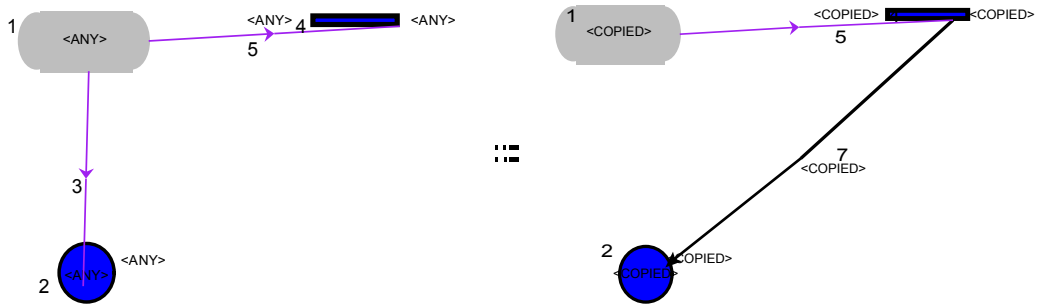

(a) LHS (b) RHS

FIGURE 3.38 – Règle Eplace2Action_2




(a) LHS (b) RHS

FIGURE 3.39 – Règle Eplace2Action_3




(a) LHS (b) RHS

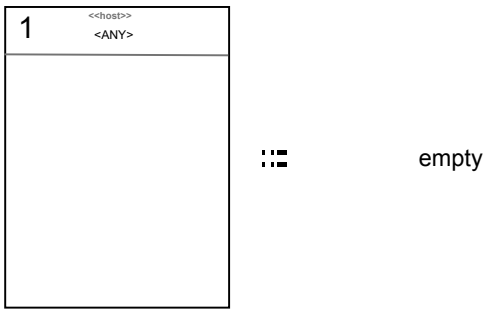
FIGURE 3.40 – Règle Eplace2Action_4


(a) LHS (b) RHS

FIGURE 3.41 – Règle Eplace2Action_5

***(7) CleanHostPartition (priorité 100 ) :*** Cette règle est appliquée pour chercher des partitions de type *"host"*, pour les supprimer. La partie droite de la règle est vide.

C'est la règle qui permet de nettoyer le modèle obtenu par la transformation des éléments de modèle source (partition de type *"host"*). Cette règle n'a ni condition, ni action.

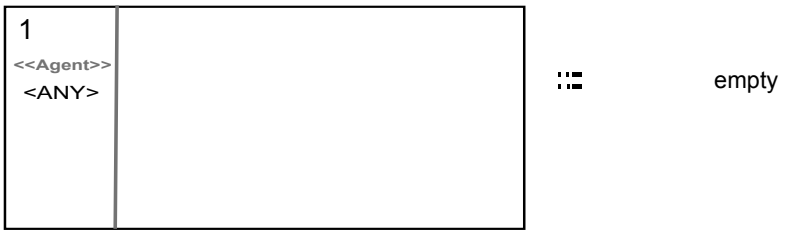



(a) LHS (b) RHS

FIGURE 3.42 – *Règle CleanHostPartition*

***(8) CleanAgentPartition (priorité 100 ) :*** Cette règle est appliquée pour chercher des partitions de type *"agent"*, pour les supprimer. La partie droite de la règle est vide.

C'est la règle qui permet de nettoyer le modèle obtenu par la transformation des éléments de modèle source (partition de type *"agent"*). Elle n'a ni condition ni action.




(a) LHS (b) RHS

FIGURE 3.43 – *Règle CleanAgentPartition*

A la fin de la présentation de cette approche, nous donneront, dans la section suivante, des exemples de transformation des modèles des diagrammes d'activités mobiles en utilisant cette approche.

## 3.6 Etude de cas

Nous essayerons à travers plusieurs exemples de montrer en détails l'application de notre approche de transformation proposée. Nous commençons par des exemples simples que nous avons créés pour les utiliser comme moyen d'explication puis nous utilisons l'exemple présenté dans la section 1.3.2 du chapitre 1.

### 3.6.1 Exemple 1

Cet exemple représente un système très simple composé d'un seul agent stationnaire *"agent1"* et un seul hôte *"host1"*. La figure 3.44 illustre cet exemple.

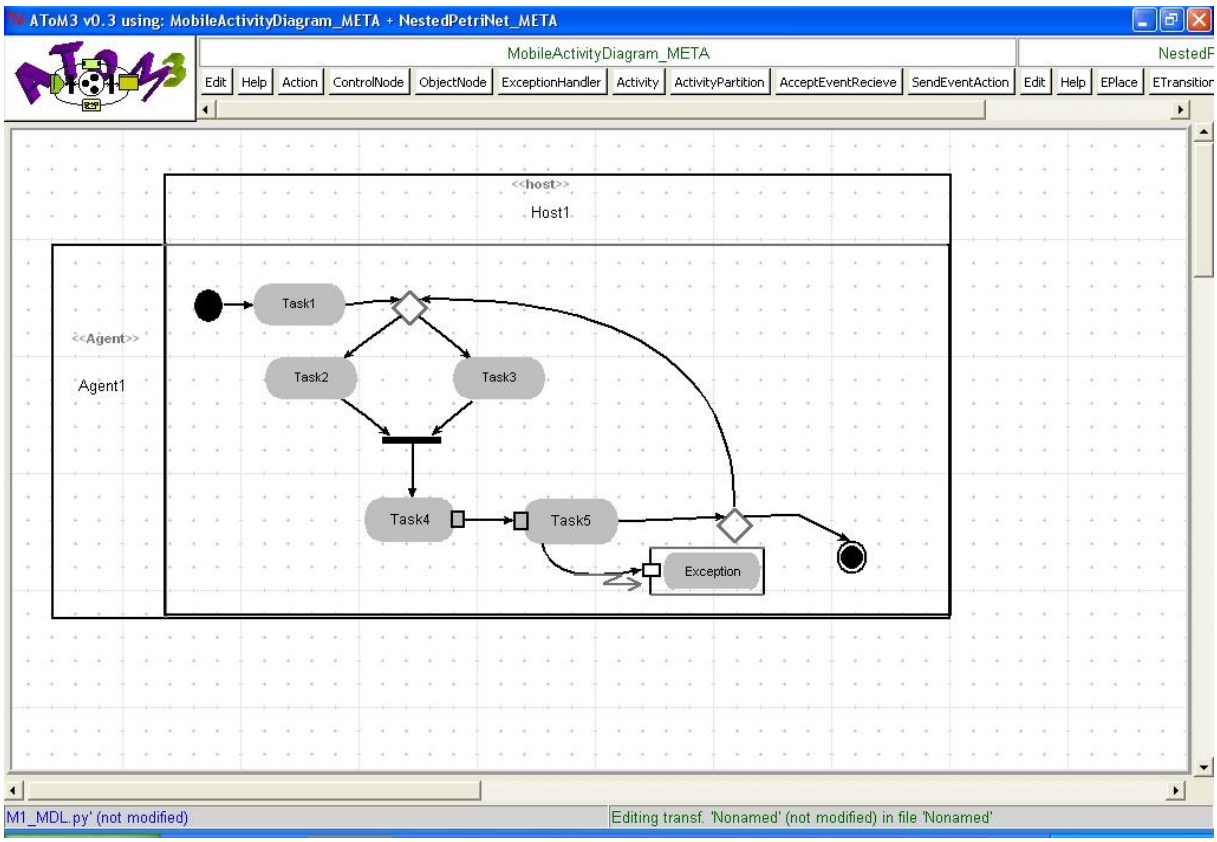



FIGURE 3.44 – Exemple 1 : Diagramme d'activités mobile

Le but de cet exemple est de démontrer la transformation des éléments de modèle de diagramme d'activités et de voir le cas d'un système à un seul agent stationnaire.

Pour plus de clarté, on n'a utilisé que peu d'éléments de modèle de diagramme d'activités mobile. La transformation des éléments de modèle qui ne sont pas présentés est similaire à la transformation des éléments présentés car ils sont de même type ou de nature similaire. Par exemple, l'élément de modèle *"flux final"* est transformer de la même façon que l'élément de modèle *"nœud d'activité final"* . La figure 3.45 illustre le résultat de transformation.

Il est clair que le réseau de Petri imbriqué résultant se compose d'un système réseau et d'un seul élément réseau représentant l'agent stationnaire.

Le système réseau créé a le nom par défaut *"System Net"*. Il est composé de deux places et une transition

- Une place dont le nom est *"S_Init"* qui représente l'état initial. Le nombre de jetons dans cette place est égal à 1.
- Une place dont le nom est *"Host1"*. Ce nom vient du fait qu'elle représente l'hôte modélisé dans le diagramme d'activités mobile transformé. Le nombre de jetons dans cette place est égal à 0.

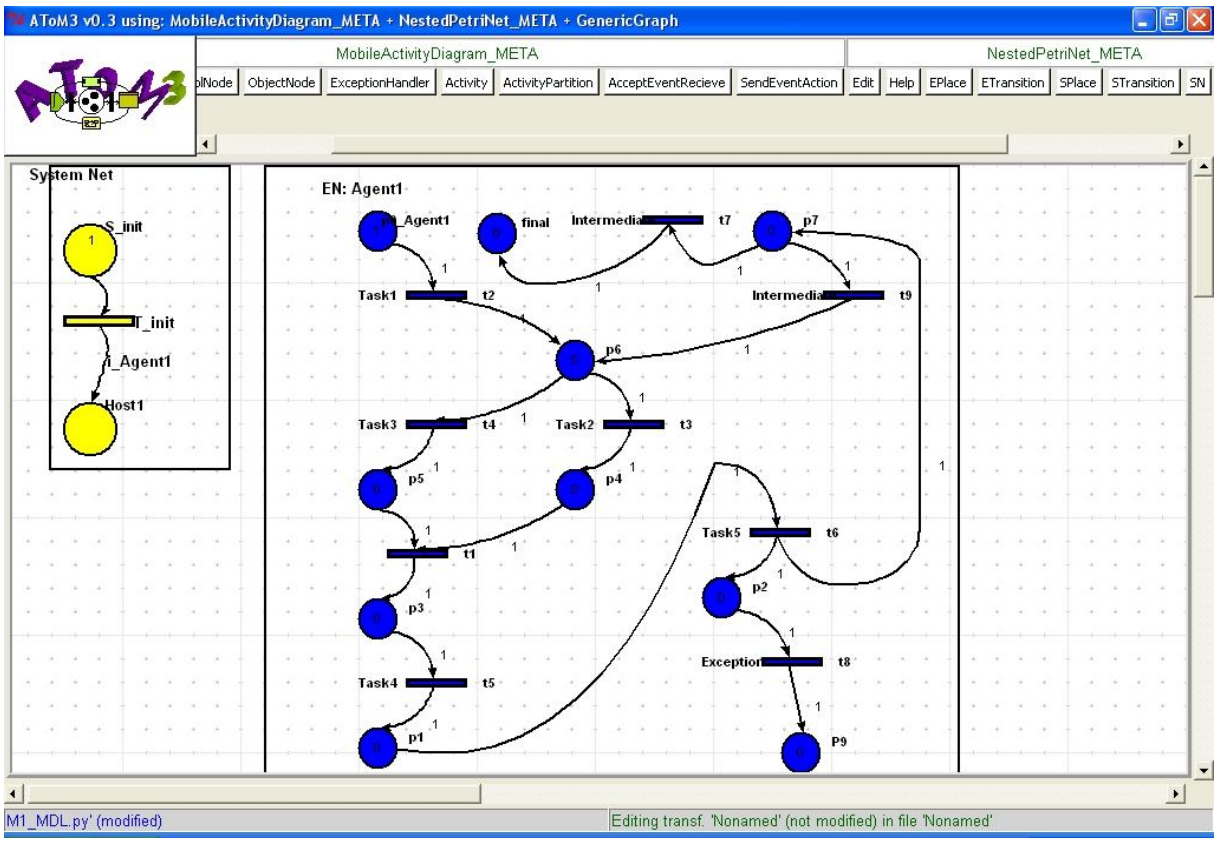



FIGURE 3.45 – Exemple 1 : Résultat de transformation

- ❑ Une transition initiale dont le nom est *"T_Init"*. La transition initiale est reliée avec la place *"Host1"*. L'expression de cet arc a la valeur *"i_Agent"*. Cela signifie qu'un élément réseau de type *"Agent1"* sera créé dans la place *"Host1"*.

L'élément réseau a la structure d'un réseau de Petri ordinaire. La seule place marquée est la place initiale dont le nombre de jetons est égal à 1. Les transitions ont des étiquettes par défaut tels que *"t1, t2, t3,..."* et des noms (note) dont la valeur vient des noms des actions transformés.

Ce réseau de Petri imbriqué ne contient pas d'étiquettes adjacentes par ce qu'il n y a pas d'action mobile dans le système modélisé.

Dans les exemples suivants, nous expliquons les propriétés mobiles d'un agent. Pour cela, les modèles ne vont contenir que le minimum des éléments de modèle pour simplifier les exemples.

### 3.6.2 Exemple 2

Cet exemple représente 03 agents *"agent1"*, *"agent2"* et *"agent3"* qui exécutent des actions mobiles dans un système composé de 03 hôtes *"Host1"*, *"Host2"*, *"Host3"*. À l'état initial, l'agent *"agent1"* et *"agent2"* résident dans le même hôte nommé *"Host1"*, alors que l'agent *"agent3"* réside dans l'hôte nommé *"Host2"*. La figure 3.46 illustre cet exemple et la figure 3.47 illustre le résultat de la transformation.

Le résultat de la transformation est un réseau de Petri imbriqué composé d'un système

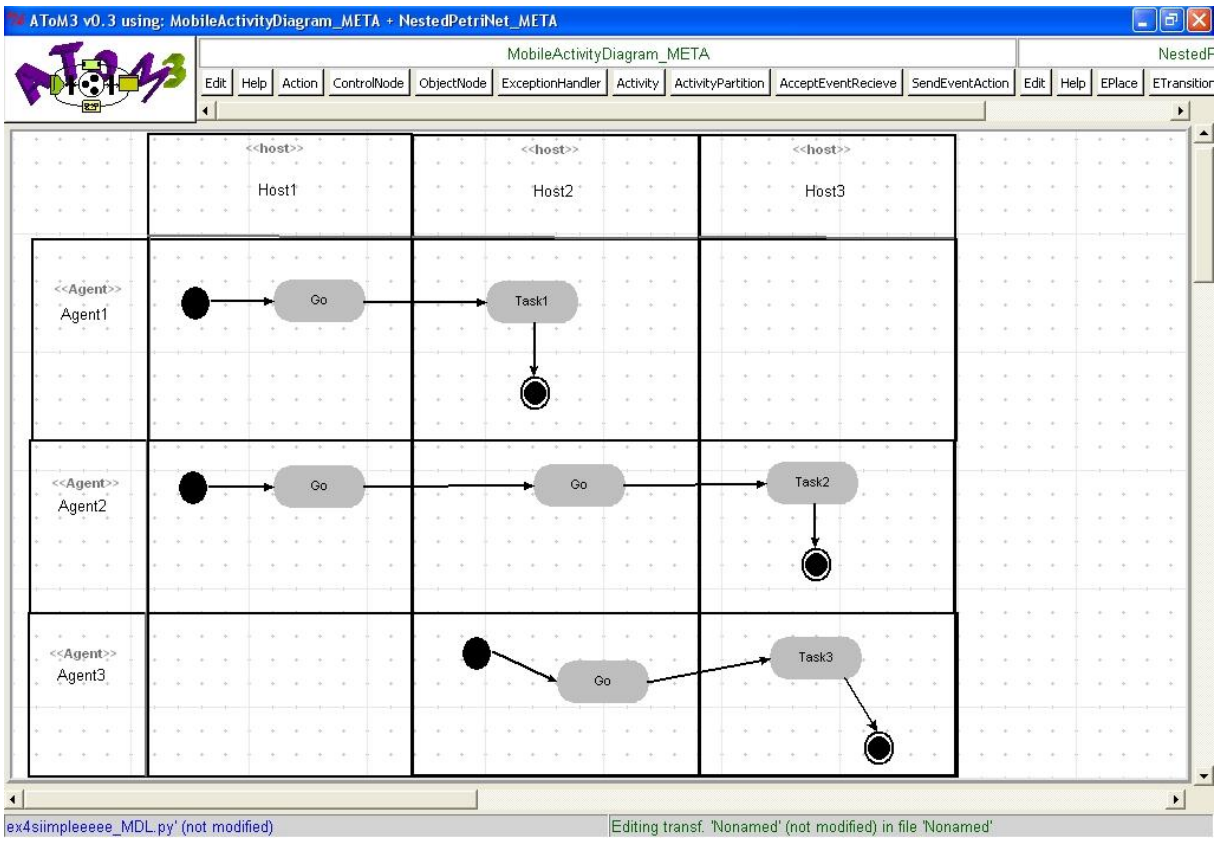



FIGURE 3.46 – Exemple 2 : Diagramme d'activités mobile

réseau et de 03 éléments réseau nommés selon les noms des agents modélisés par le modèle de diagramme d'activités mobile. Le système réseau est lui-même composé de 03 places représentant les hôtes modélisés et une place qui représente l'état initial.

L'analyse du diagramme d'activités mobile de l'exemple et son équivalent obtenu par la transformation de graphe, permet de dir que :

- À l'état initial, les agents "*agent1*" et "*agent2*" résident dans le même hôte nommé "*Host1*". Cela est représenté par l'arc qui lie la place nommé "*Host1*" avec la transition initiale. La valeur de l'expression de cet arc est composé des noms des agents "*agent1*" et "*agent2*".
- À l'état initial, aussi, l'agent "*agent3*" réside dans l'hôte nommé "*Host2*". Cela est représenté par l'arc qui lie la place nommé "*Host2*" avec la transition initiale. La valeur de l'expression de cet arc est composé des noms des agents "*agent1*" et "*agent2*".
- L'action mobile exécutée par les agents *"agent1"* et *"agent2"* pour se déplacer de l'hôte "*host1*" vers l'hôte "*host2*" est représenté par :

  1. Une transition qui relie la place nommé *"Host1"* avec la place nommé "*Host2*". Elle a l'étiquette "*T1*".
  2. L'expression d'arc qui relie la transition "*T1*" avec la place "*Host2*" est composée des noms des agents exécutant l'action mobile.
  3. La transition exécutant l'action mobile a une étiquette "*TT1*" adjacente à

l'étiquette "*T1*" pour qu'elle soit synchronisée verticalement avec la transition *T1* de système réseau.

On peut dire la même chose sur le reste des actions mobiles.

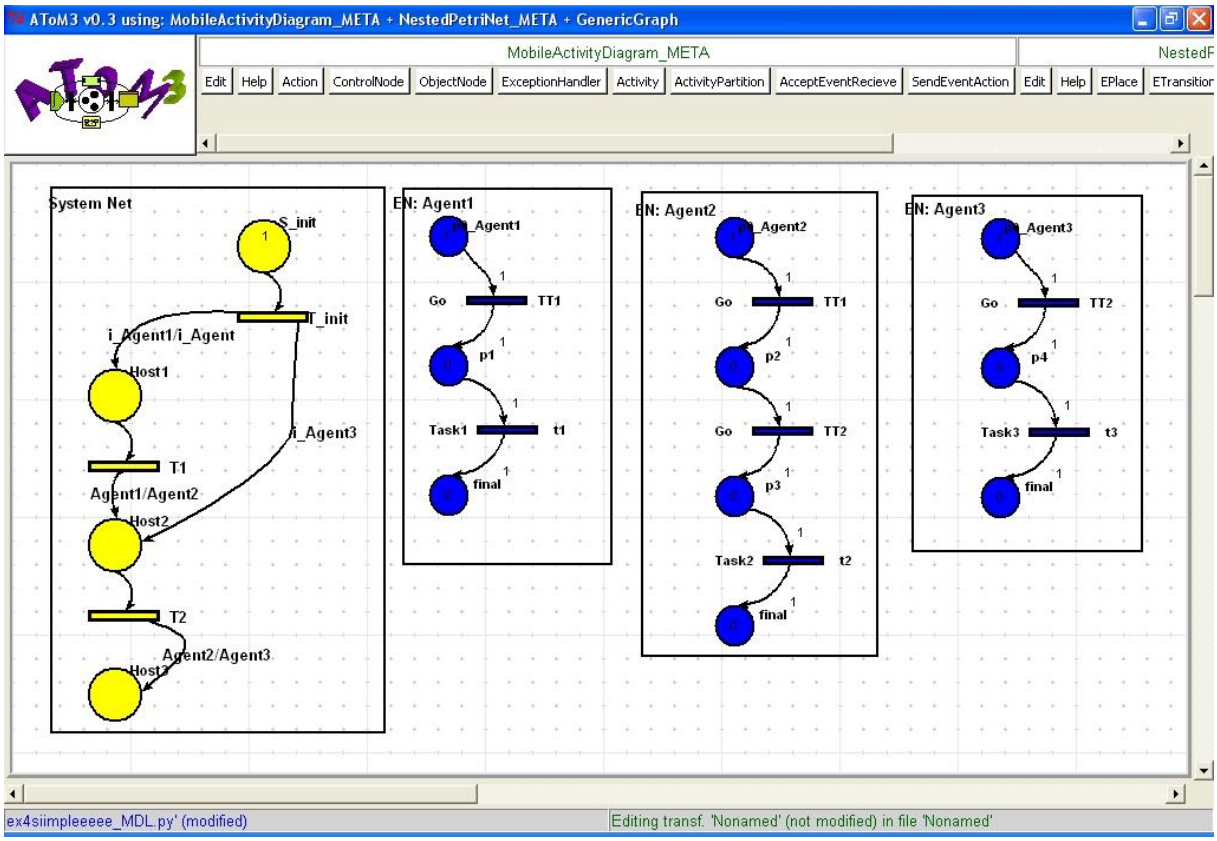



FIGURE 3.47 – Exemple 2 : Résultat de transformation

### 3.6.3 Exemple 3

L'exemple représente un système composé de deux agents *"agent1"*, *"agent2"* et trois hôte *"Host1"*, *"Host2"* et *"Host3"*. La figure 3.48 illustre cet exemple.

À travers cet exemple, nous illustrons la transformation de l'activité de clonage. Pour modéliser cette activité, on a utilisé une activité spéciale. Elle a comme paramètre le nom d'agent à cloner et l'hôte destinataire. Par exemple, pour que l'agent "*agent2*" clone l'agent *"agent1"* vers l'hôte "*Host3*", on spécifie dans le nom de l'activité de clonage ces paramètre comme suit : *"Clone :agent1 :Host3"*.

L'application de notre approche de transformation nous donne le résultat illustré par la figure 3.49.

L'activité de clonage est mappée à une étape de transfert dans le réseau de Petri imbriqué. Ainsi, la transformation de l'exemple de cette activité illustré par la figure 3.48 permet de lier la place "*Host1*" ou le clonage se déroule et la place destinataire "*Host3*" à travers la transition "*T2*".

L'expression de l'arc qui relie la transition "*T2*" et la place "*Host3*" est composée des noms des éléments réseau impliqué dans une synchronisation verticale ou une étape

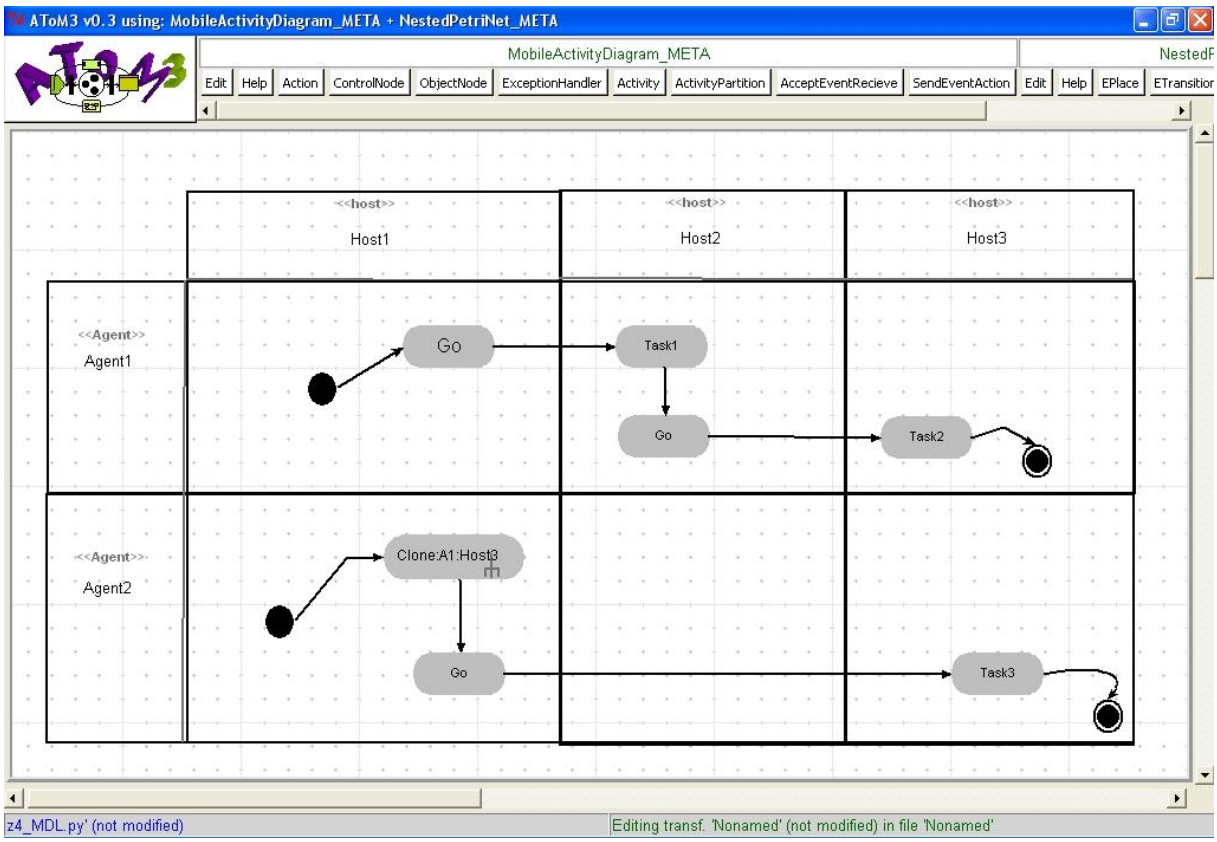



FIGURE 3.48 – Exemple 3 : Diagramme d'activités mobile

de transfert. Ainsi, le nom de l'agent *«agent1 »* précédé par la lettre "c" est placé dans l'expression. La lettre c permet de distinguer les noms engendrés par l'activité de clonage.

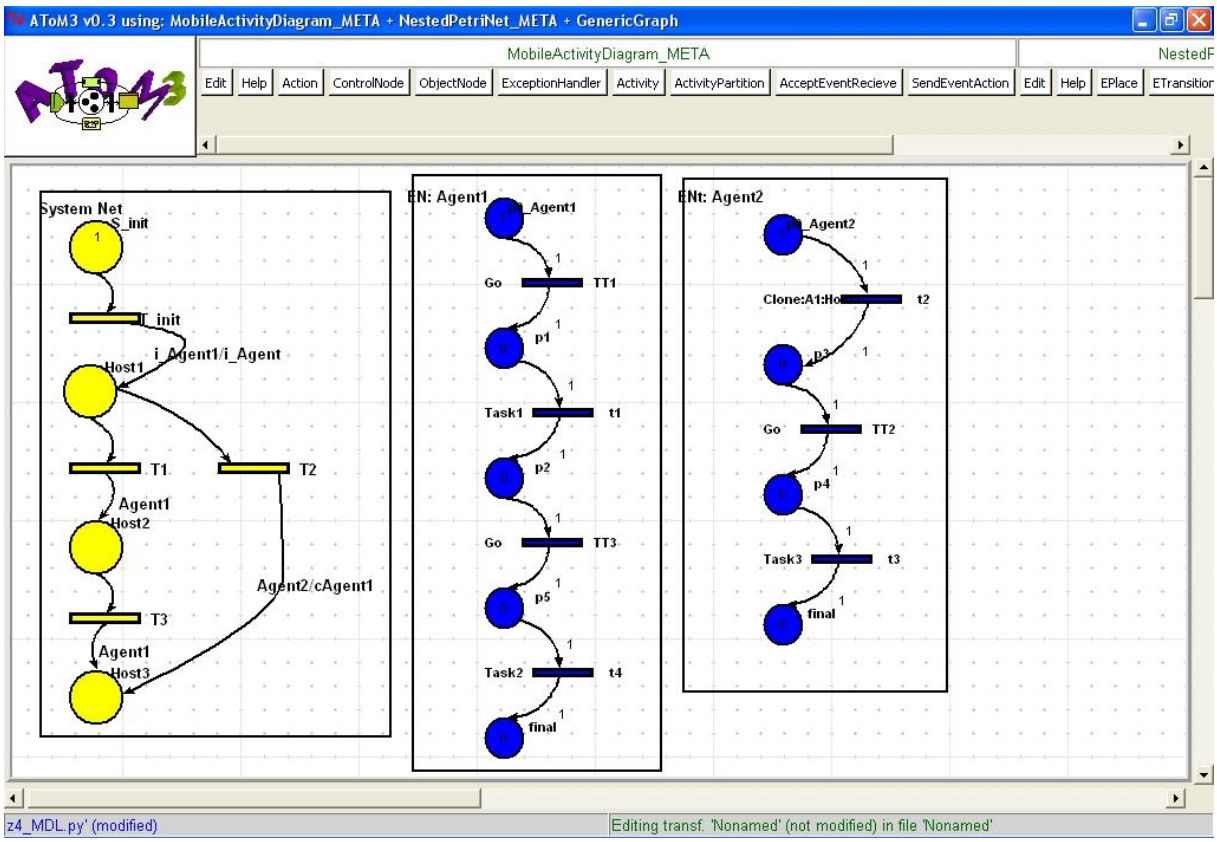



FIGURE 3.49 – Exemple 3 : Résultat de transformation

### 3.6.4 Exemple 4 :

Cet exemple illustré par la figure 3.50 est le même que l'exemple présenté dans la section 3.6.2 avec un nombre élevé d'actions mobiles. Le résultat de L'application de notre approche de transformation est illustré par les figures 3.51 et 3.52.

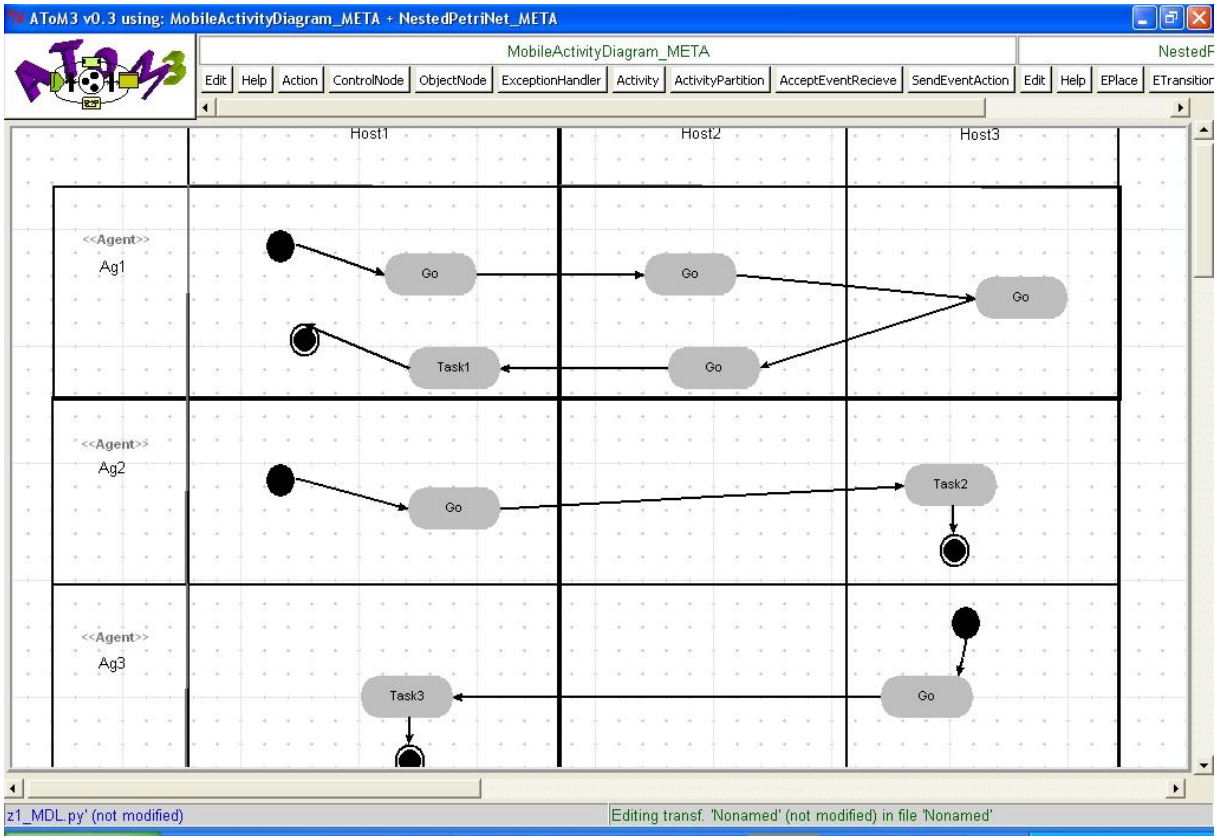



FIGURE 3.50 – Exemple 4 : Diagramme d'activités mobile

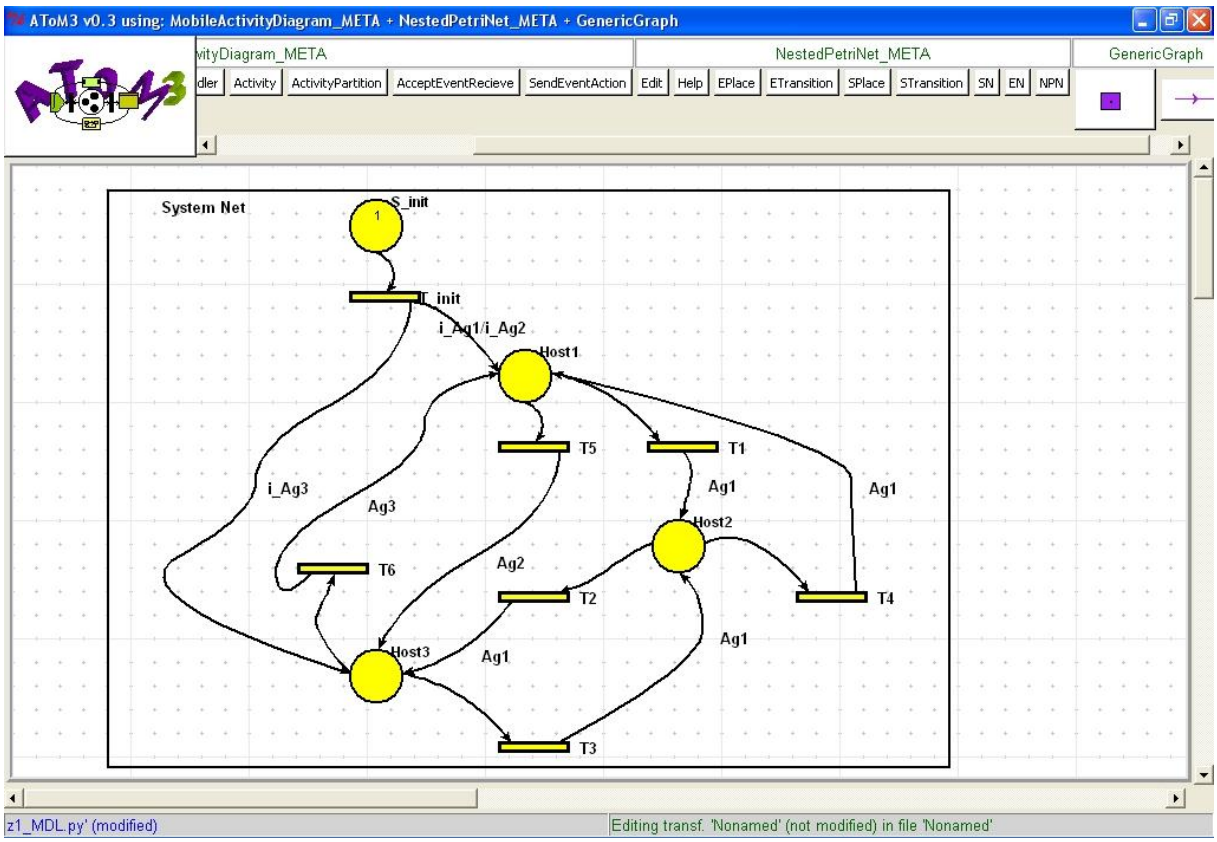



FIGURE 3.51 – Exemple 4 : Système réseau résultat

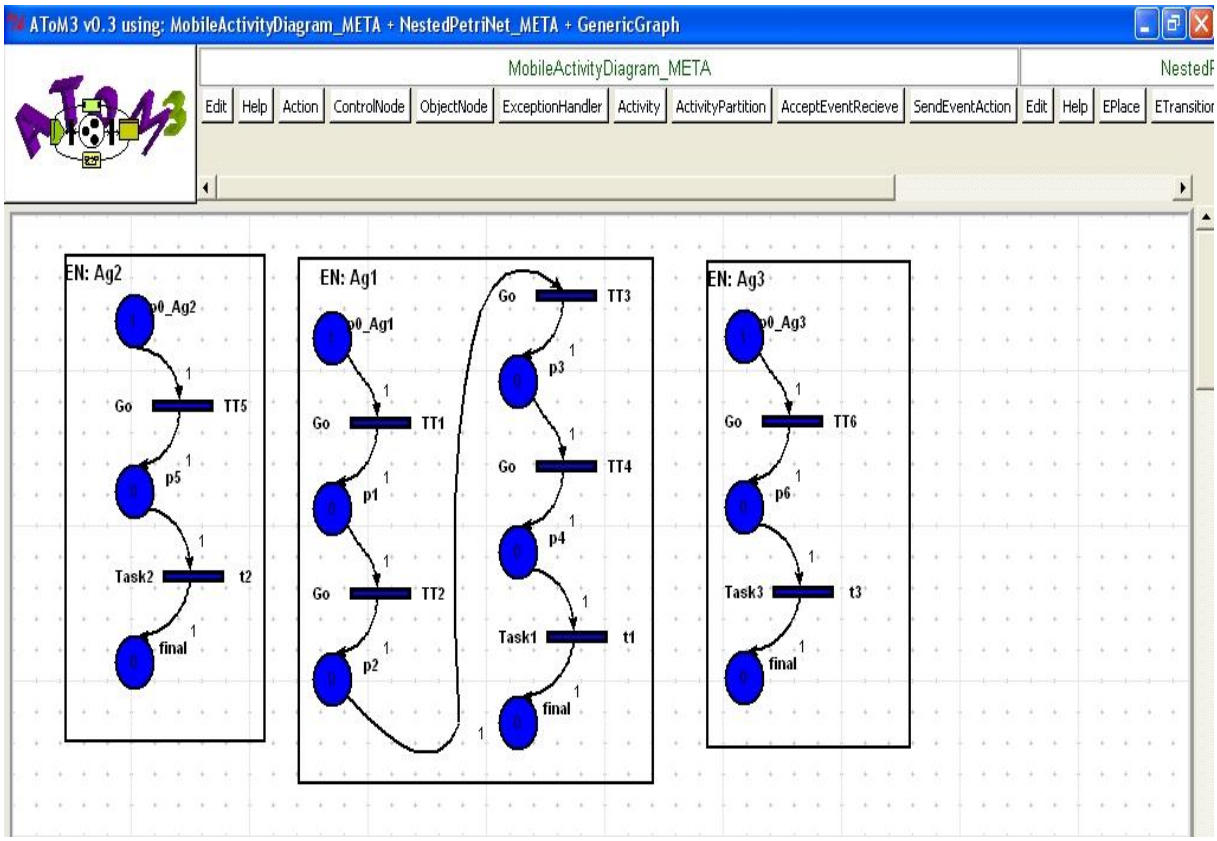



FIGURE 3.52 – Exemple 4 : L'élément réseau résultat

### 3.6.5 Exemple 5 :

Cet exemple a été présenté dans la section 1.3.2 du chapitre 1 comme un cas d'étude pour démontrer l'utilisation des diagrammes d'activités mobiles. Il représente un système de vente aux enchères composé d'un agent stationnaire vendeur (*Auctioneer*) et d'un agent mobile acheteur (*Bidder*). On a effectué de légère modification sur l'exemple originale pour des raisons de simplicité et de clarté. La figure 3.53 illustre cet exemple.

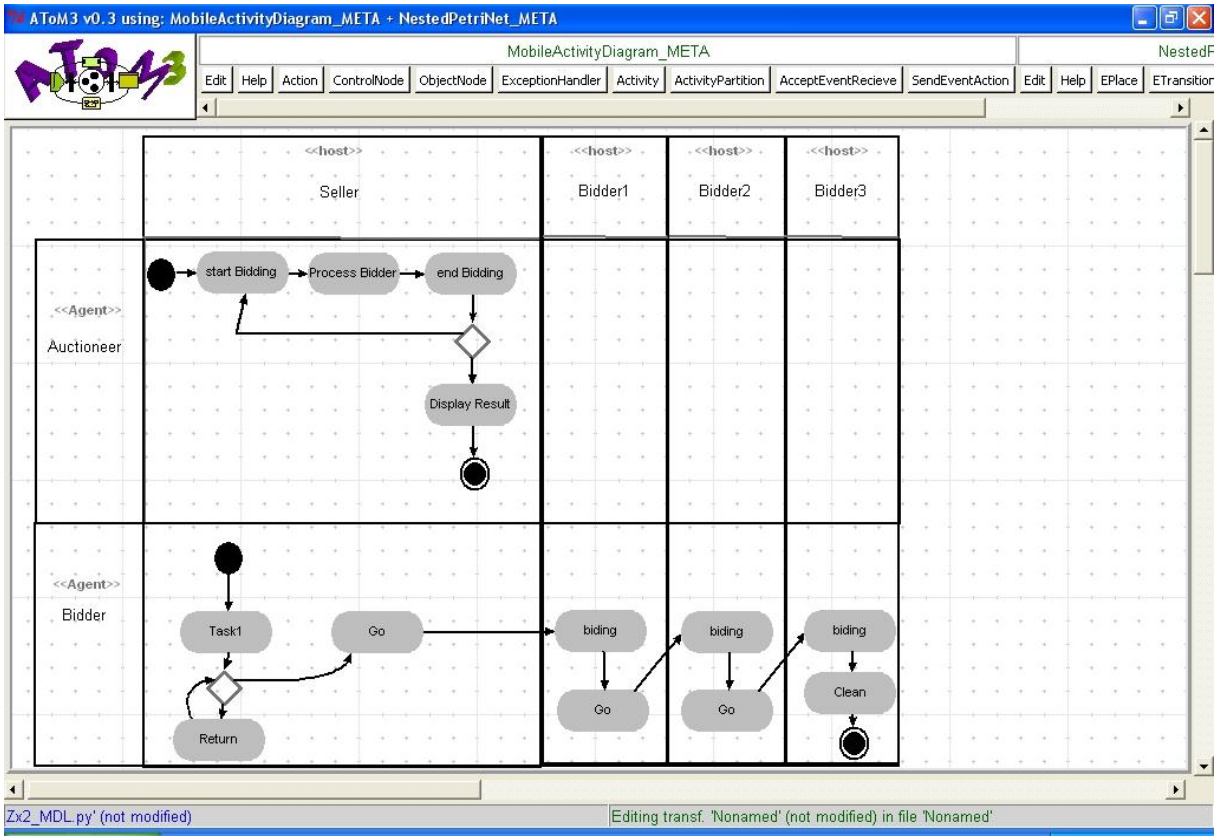



FIGURE 3.53 – Exemple 5 : Diagramme d'activités mobile

Le résultat de l'application de notre approche de transformation est illustré par les figures 3.54 et 3.55.

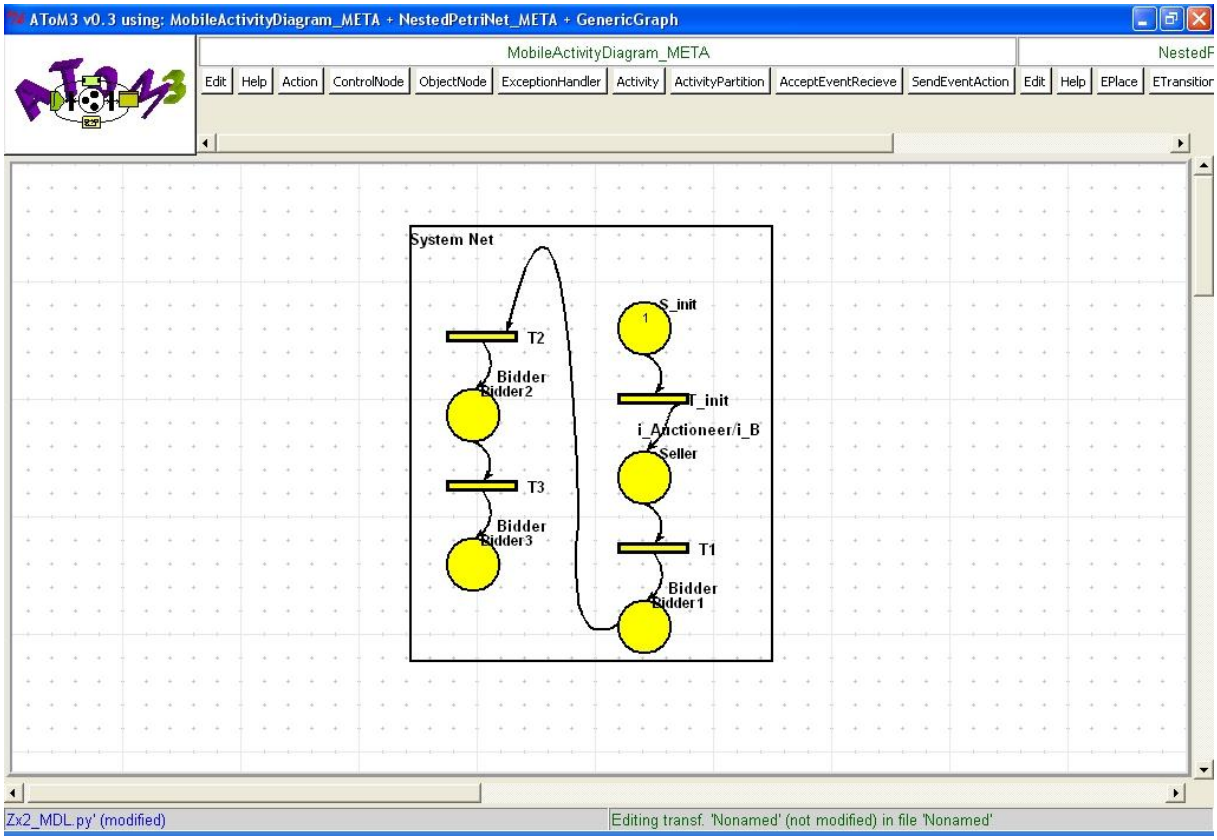



FIGURE 3.54 – Exemple 5 : Système réseau résultat

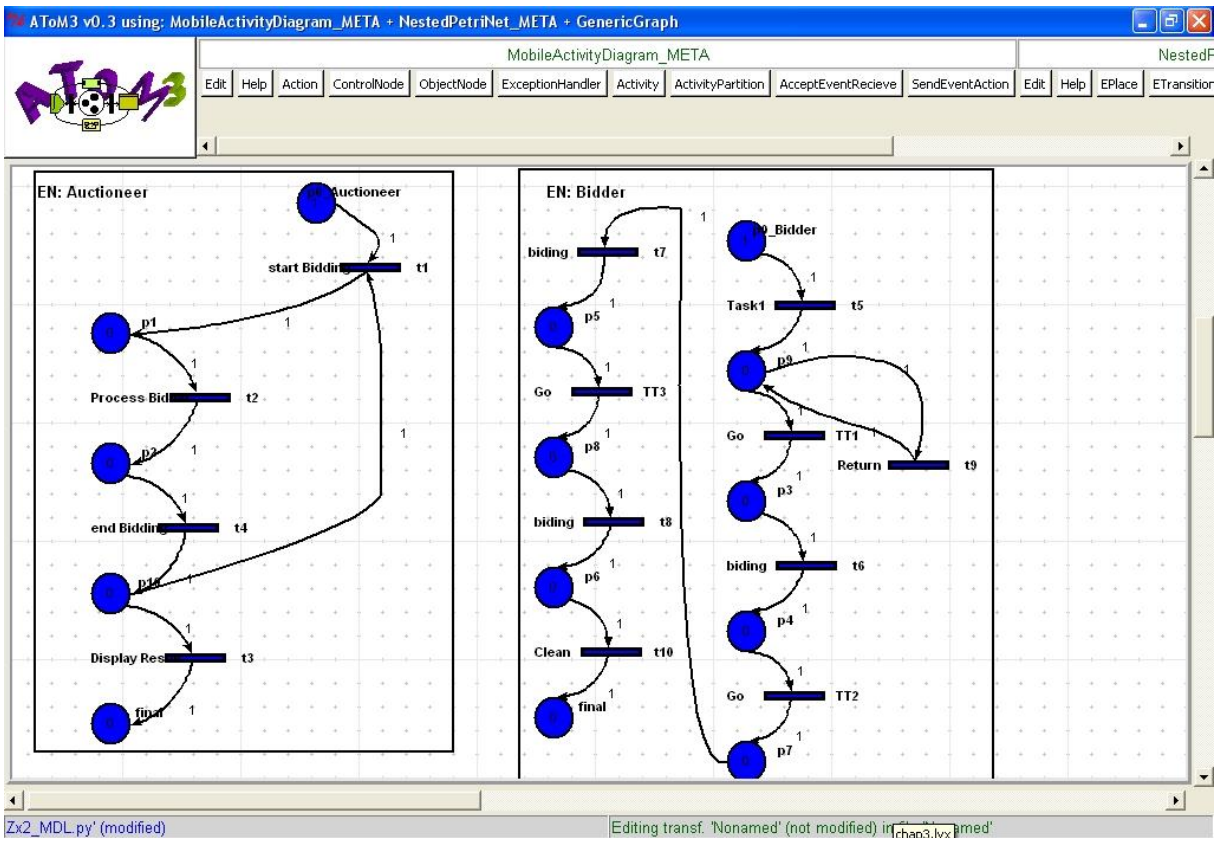



FIGURE 3.55 – Exemple 5 : L'élément réseau résultat

## 3.7 Conclusion

Nous avons présenté dans ce chapitre notre approche de transformation de diagramme d'activités mobile vers un réseau de Petri de haut niveau.

La méthode de transformation démarre par la définition d'un méta-modèle pour les réseaux de Petri imbriqués et d'un méta-modèle pour les diagrammes d'activités mobiles. Ces deux méta-modèles sont utilisés par $AToM^3$ pour générer deux outils de modélisations, puis par définir les règles de transformation d'un modèle de type diagramme d'activités vers un modèle de type réseau de Petri *(NPN)*. Ces derniers sont illustrées et expliquées.

# Conclusion Générale

A travers ce mémoire, notre objectif était de proposer une approche de transformation des diagrammes d'activités mobiles vers les réseaux de Petri imbriqués

Nous avons introduit dans le premier chapitre les concepts de base de la mobilité et en particulier, les agents mobiles. Puis nous avons décrit, brièvement, le langage UML. ce chapitre a été terminé par une présentation des diagrammes d'activités mobile qui sont une extension des diagrammes d'activités ordinaires.

Le deuxième chapitre a été le sujet de détail de réseau de Petri. Les concepts de base et les principales propriétés des réseaux de Petri ordinaires ont été évoqués. Puis les réseaux de Petri imbriqués ont été présentés. Ces derniers sont des réseaux de Petri de haut niveau permettant la modélisation des agents mobiles.

Dans le troisième chapitre, nous avons présenté notre approche pour la transformation des diagrammes d'activités mobiles vers les réseaux de Petri imbriqués. Nous avons vu que cette approche se divise en trois étapes : la première consiste à proposer un méta-modèle des diagrammes d'activités mobiles. Ce dernier est utilisé par l'outil $AToM^3$ pour générer un outil de modélisation des diagrammes d'activités mobiles. La deuxième étape consiste aussi à proposer un méta-modèle pour les réseaux de Petri imbriqués. A partir de ce dernièr, $AToM^3$ génère un outil de modélisation des réseaux de Petri imbriqués. La troisième étape consiste à définir une grammaire de graphes qui permet de transformer un graphe source décrit avec le formalisme de diagramme d'activités mobile vers un graphe de réseau de Petri

Malgré que le résultat de la transformation de diagramme d'activités mobile soit un réseau de Petri imbriqué, il n'est pas possible de réaliser une vérification automatique parce que il ne peut pas être utilisé directement pour l'analyse et la vérification avec les outils existants.

Nous proposons, comme perspective à ce travail, la définition d'une grammaire de graphes pour transformer les réseaux de Petri imbriqués à des réseaux de Petri ordinaires. Ces derniers peuvent être analysé et vérifié directement. Ainsi, nous proposons un seul outil qui intègrera la modélisation, l'analyse et la vérification des diagrammes d'activités mobiles d'une manière transparente.

# Bibliographie


[1] O. M. OMG, "Unified modeling language : Superstructure," tech. rep., OMG, march 2005.

[2] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Transactions on Software Engineering*, vol. 24, pp. 342–361, Feb. 03 1998.

[3] C. C. D. CROS, *Agents Mobiles Coopérants pour les Environnements Dynamiques*. PhD thesis, Institut National Polytechnique de Toulouse, 2 decembre 2005.

[4] J. Ferber, *Les Systèmes Multiagents : Vers une Intelligence Collective*. InterEditions, 1995.

[5] N. R. Jennings, K. P. Sycara, and M. Wooldridge, "A roadmap of agent research and development," *Autonomous Agents and Multi-Agent Systems*, vol. 1, no. 1, pp. 7–38, 1998. `http://www.ecs.soton.ac.uk/~nrj/download-files/roadmap.pdf`.

[6] M. Obitko, *Translations between Ontologies in Multi-Agent Systems*. PhD thesis, Gerstner Laboratory, Department of Cybernetics, Czech Technical University, April 5, 2007.

[7] J. L. Austin, *How to Do Things With Words*. Oxford : Oxford University Press, 1962.

[8] K. Hamilton and R. Miles, *Learning UML 2.0*. O'Reilly, April 2006.

[9] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. Addison Wesley Longman, Inc, 1998.

[10] T. Pender, *UML Bible*. John Wiley & Sons, 2003.

[11] H.-E. Eriksson, M. Penker, B. Lyons, and D. Fado, *UML2 Toolkit*. John Wiley & Sons, 2004.

[12] H. Baumeister, N. Koch, P. Kosiuczenko, and M. Wirsing, "Extending activity diagrams to model mobile systems," Aug. 21 2002.

[13] C. Klein, A. Rausch, M. Sihling, and Z. Wen, "Extension of the unified modeling language for mobile agents," pp. 116–128, 2001.

[14] K. Saleh and C. El-Morr, "M-UML : an extension to UML for the modeling of mobile agent-based software systems," *Information & Software Technology*, vol. 46, no. 4, pp. 219–227, 2004.

[15] H. Mouratidis, J. Odell, and G. Manson, "Extending the unified modeling language to model mobile agents,"

[16] M. Kang, L. Wang, and K. Taguchi, "Modelling mobile agent applications in UML2.0 activity diagrams," Apr. 21 2004.

[17] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE : a FIPA2000 compliant agent development environment," in *Proceedings of the Fifth International Conference on Autonomous Agents* (J. P. Müller, E. Andre, S. Sen, and C. Frasson, eds.), (Montreal, Canada), pp. 216–217, ACM Press, May 2001.

[18] C. A. Petri, *Kommunikation mit Automaten*. PhD thesis, University of Bonn, Bonn, Germany, 1962. (In German).

[19] R. David and H. Alla, *Discrete, Continuous, and Hybrid Petri Nets*. Springer, 1 ed., 23 November 2004.

[20] T. Murata, "Petri nets : Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, pp. 541–580, April 1989.

[21] R. Valette, "Les réseaux de pétri." LAAS-CNRS Toulouse, `http://www.laas.fr/~robert/enseignement.d`, Septembre 2002.

[22] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs : Prentice Hall, 1 ed., 1981.

[23] G.Scorletti and G.Binet, *Réseaux de Petri*. Université de Caen Basse Normandie France, 20 juin 2006. `http://www.metz.supelec.fr/~vialle/course/CNAM-ACCOV-NFP103/extern-doc/RdP/Cours_Petri_etudiant_GS_2006.pdf`.

[24] K. Jensen, "An introduction to the practical use of coloured petri nets.," *Lecture Notes in Computer Science : Lectures on Petri Nets II : Applications*, vol. 1492, 1998.

[25] K. Jensen, "A brief introduction to coloured Petri nets," in *Tools and Algorithms for the Construction and Analysis of Systems* (E. Brinksma, ed.), vol. 1217 of *Lecture Notes in Computer Science*, pp. 203–208, Springer-Verlag, 1997.

[26] J. Lamp, "Using petri nets to model weltanschauung alternatives in soft systems methodology," Oct. 27 1998.

[27] I. A. Lomazova, "Nested petri nets – a formalism for specification and verification of multi-agent distributed systems," *FUNDINF : Fundamenta Informatica*, vol. 43, pp. 195–214, 2000.

[28] I. A. Lomazova and P. Schnoebelen, "Some decidability results for nested petri nets," in *Ershov Memorial Conference* (D. Bjørner, M. Broy, and A. V. Zamulin, eds.), vol. 1755 of *Lecture Notes in Computer Science*, pp. 208–220, Springer, 1999.

[29] I. A. Lomazova, "Nested petri nets : Multi-level and recursive systems," *Fundam. Inform*, vol. 47, no. 3-4, pp. 283–293, 2001.

[30] K. V. Hee, I. A. Lomazova, O. Oanea, E. Serebrenik, N. Sidorova, and M. Voorhoeve, "Nested nets for adaptive systems," Apr. 03 2008.

[31] T. Kühne, "Matters of (meta-)modeling," *Software and System Modeling*, vol. 5, no. 4, pp. 369–385, 2006.

[32] B. Selic, "Specification and modeling : An industrial perspective," in *ICSE*, pp. 676–677, IEEE Computer Society, 2001.

[33] J. Bézivin and O. Gerbé, "Towards a precise definition of the OMG/MDA framework," in *ASE*, pp. 273–280, IEEE Computer Society, 2001.

[34] A. G. Kleppe, J. Warmer, and W. Bast, *MDA Explained : The Model Driven Architecture : Practice and Promise*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2003.

[35] T. Clark, A. Evans, P. Sammut, and J. Willans, "Applied metamodelling : A foundation for language driven development," 2004.

[36] J. Álvarez, A. Evans, and P. Sammut, "Mml and the metamodel architecture," in *WTUML : Workshop on Transformation in UML 2001* (J. Whittle, ed.), apr 2001.

[37] H. Ehrig, K. Ehrig, C. Ermel, F. Hermann, and G. Taentzer, "Information preserving bidirectional model transformations," in *FASE* (M. B. Dwyer and A. Lopes, eds.), vol. 4422 of *Lecture Notes in Computer Science*, pp. 72–86, Springer, 2007.

[38] T. Mens, K. Czarnecki, and P. V. Gorp, "04101 discussion – A taxonomy of model transformations," in *Language Engineering for Model-Driven Software Development* (J. Bezivin and R. Heckel, eds.), no. 04101 in Dagstuhl Seminar Proceedings, (Dagstuhl, Germany), Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[39] K. Czarnecki and S. Helsen, "Classification of model transformation approaches," in *OOPSLA03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.

[40] M. Andries, G. Engels, A. Habel, B. Hoffmann, H.-J. Kreowski, S. Kuske, D. Plump, A. Schürr, and G. Taentzer, "Graph transformation for specification and programming," *Science of Computer Programming*, vol. 34, pp. 1–54, Apr. 1999.

[41] R. Heckel, "Graph transformation in a nutshell," *Electronic Notes in Theoretical Computer Science*, vol. 148, no. 1, pp. 187–198, 2006.

[42] "`http://atom3.cs.mcgill.ca/`," april 2010.

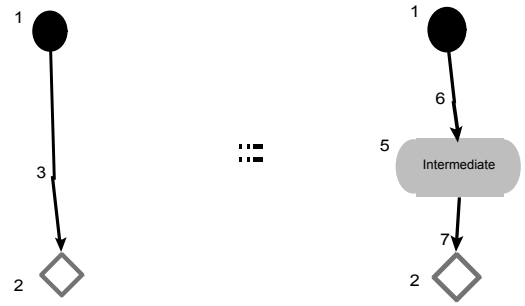
[43] J. de Lara and H. Vangheluwe, "AToM$^3$ : A tool for multi-formalism and metamodelling," in *FASE* (R.-D. Kutsche and H. Weber, eds.), vol. 2306 of *Lecture Notes in Computer Science*, pp. 174–188, Springer, 2002.

[44] J. de Lara, H. Vangheluwe, and M. Alfonseca, "Meta-modelling and graph grammars for multi-paradigm modelling in AToM$^3$," *Software and System Modeling*, vol. 3, no. 3, pp. 194–209, 2004.

[45] T. S. Staines, "Intuitive mapping of UML 2 activity diagrams into fundamental modeling concept petri net diagrams and colored petri nets," pp. 191–200, 2008.

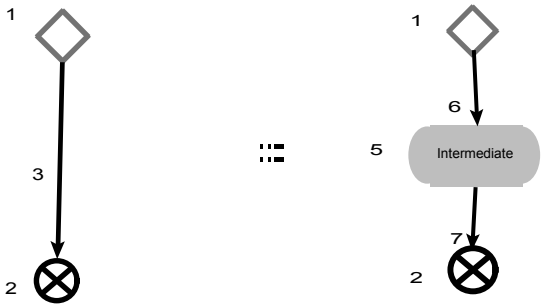
# Annexe A

Dans cette annexe, nous représentons un ensemble de règles de la grammaire permettant la transformation des éléments de modèle de diagramme d'activités mobile. L'ordre d'apparence des règles est selon leurs priorités décroissantes.




(a) LHS (b) RHS

FIGURE 1 – Initial2Decision

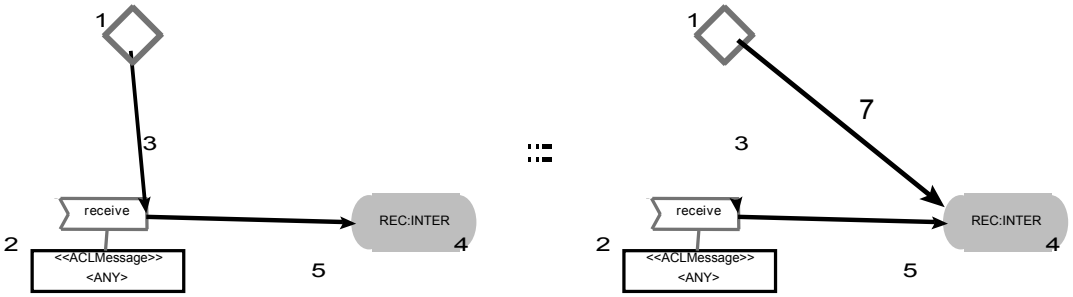



(a) LHS (b) RHS

FIGURE 2 – Decision2FlowFinal

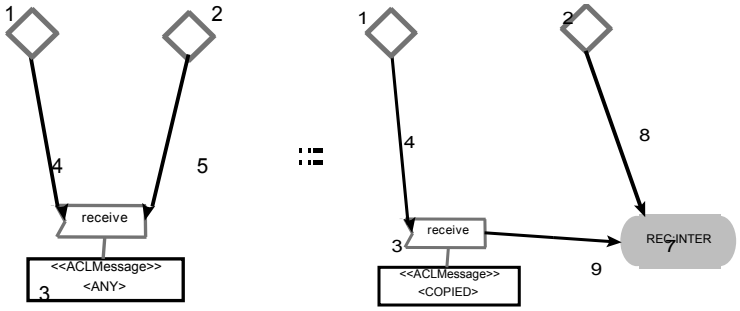

(a) LHS (b) RHS

FIGURE 3 – Action2Exception



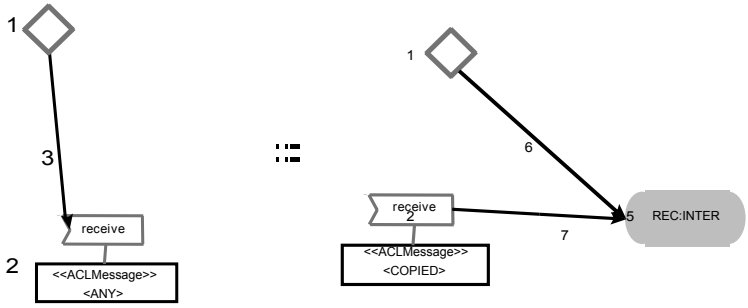

(a) LHS (b) RHS

FIGURE 4 – Ctrl2Recieve_1



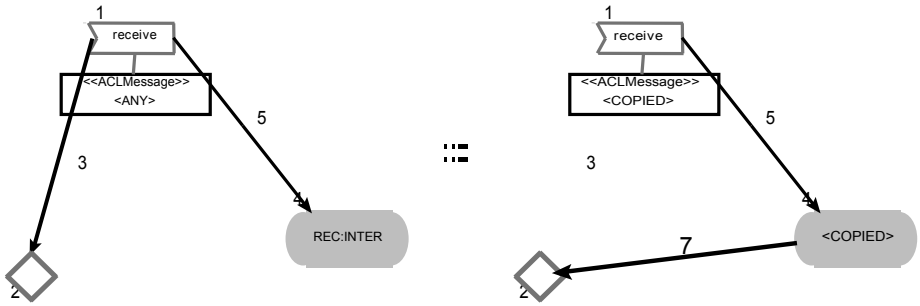

(a) LHS (b) RHS

FIGURE 5 – Ctrl2Recieve_2



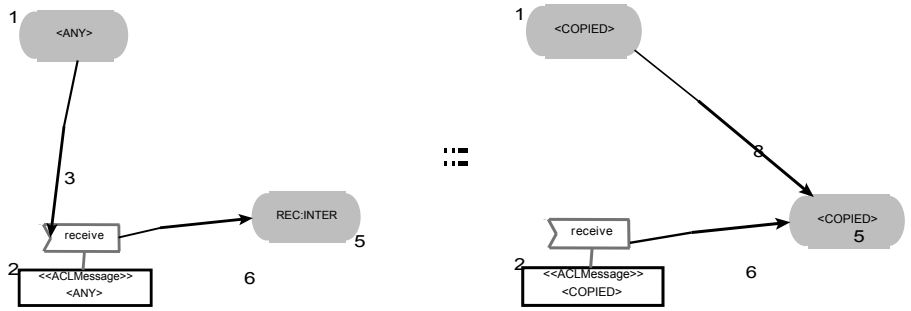

(a) LHS (b) RHS

FIGURE 6 – Ctrl2Recieve_3



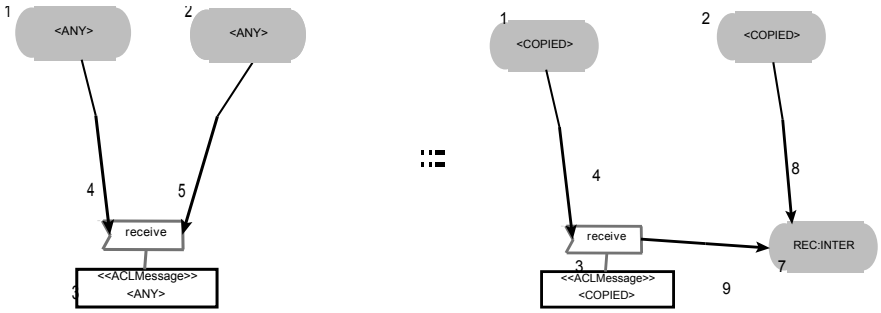

(a) LHS (b) RHS

FIGURE 7 – Ctrl2Recieve_4



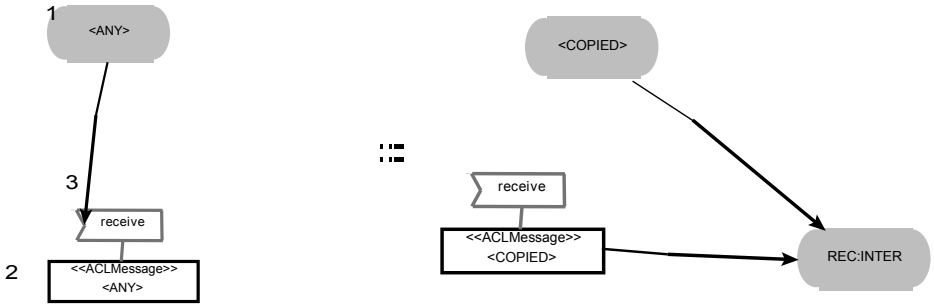

(a) LHS (b) RHS

FIGURE 8 – Action2Recieve_1



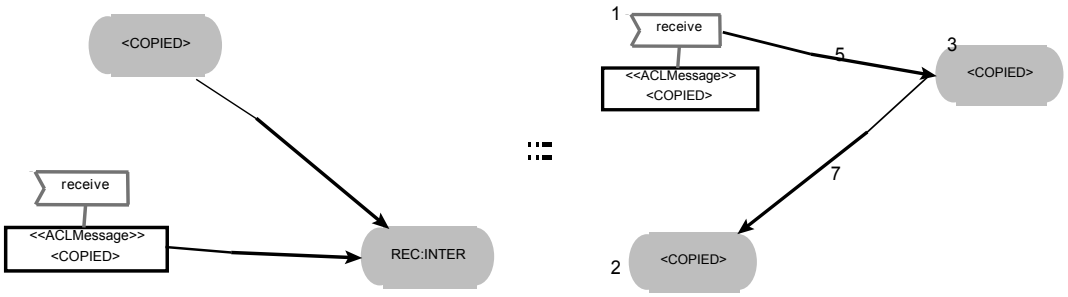

(a) LHS (b) RHS

FIGURE 9 – Action2Recieve_2



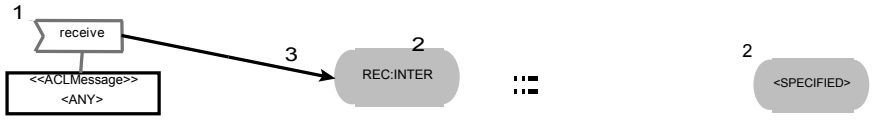

(a) LHS (b) RHS

FIGURE 10 – Action2Recieve_3



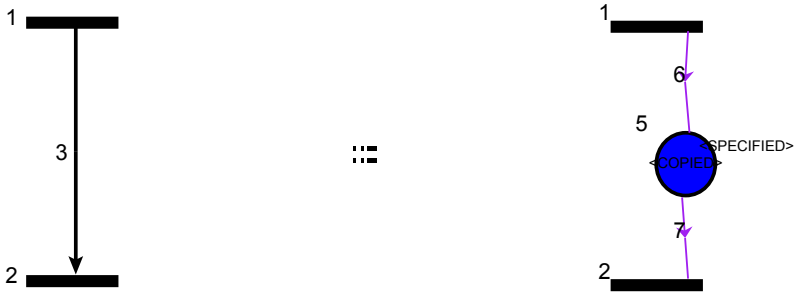

(a) LHS (b) RHS

FIGURE 11 – Action2Recieve_4



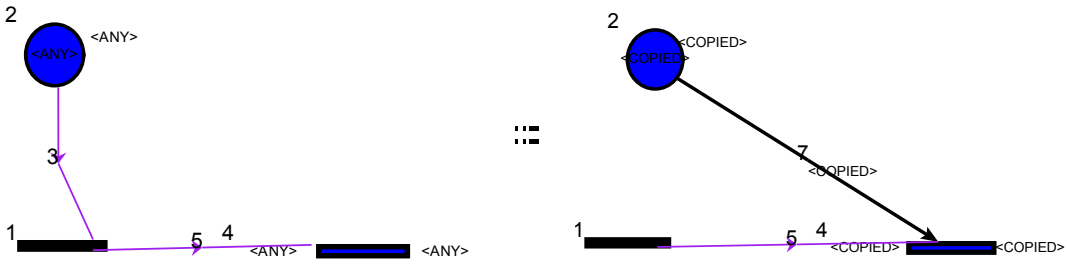

(a) LHS (b) RHS

FIGURE 12 – Action2Recieve_5



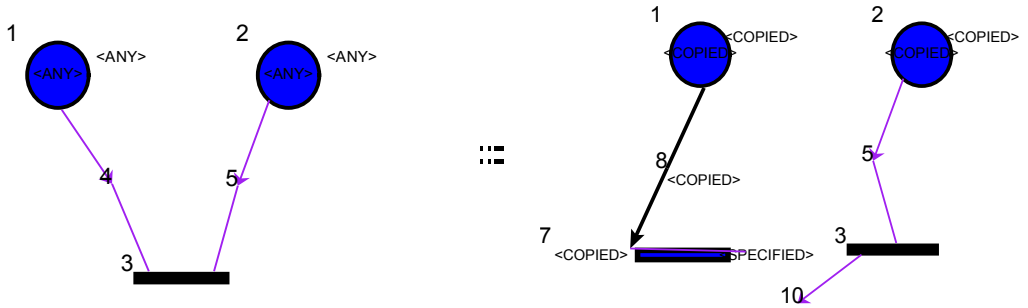

(a) LHS (b) RHS

FIGURE 13 – Join2Join



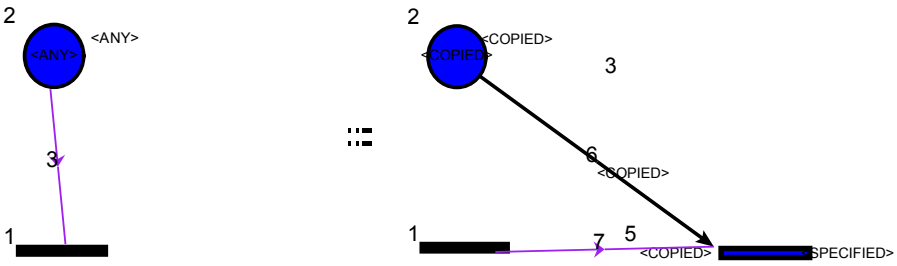

(a) LHS (b) RHS

FIGURE 14 – Eplace2Join_1



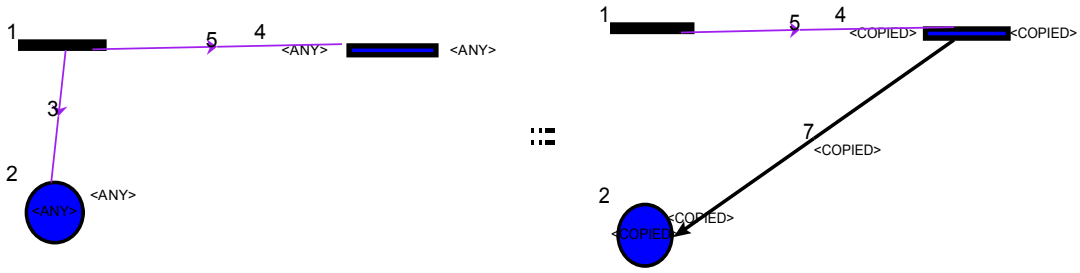

(a) LHS (b) RHS

FIGURE 15 – Eplace2Join_2




(a) LHS (b) RHS

FIGURE 16 – Eplace2Join_3

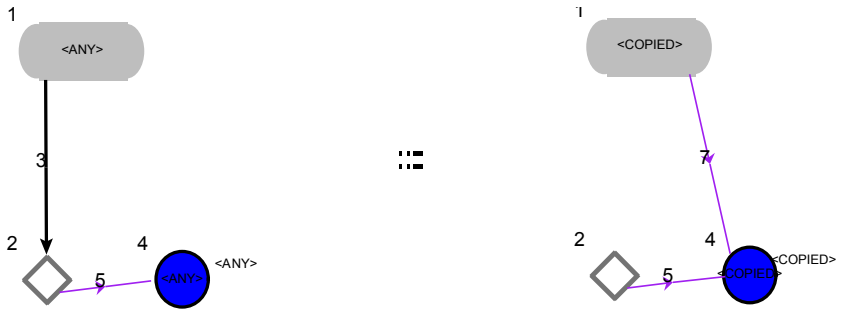



(a) LHS (b) RHS

FIGURE 17 – Eplace2Join_4

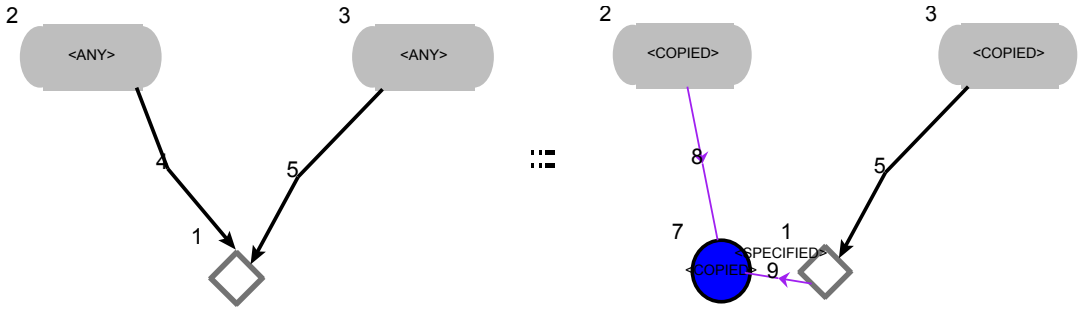

(a) LHS (b) RHS

FIGURE 18 – Eplace2Join_5




(a) LHS (b) RHS

FIGURE 19 – DecisionMerge2Eplace_1

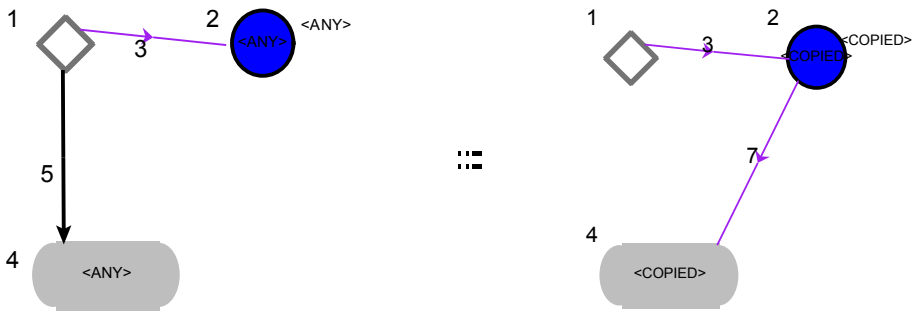



(a) LHS (b) RHS

FIGURE 20 – DecisionMerge2Eplace_2


(a) LHS (b) RHS

FIGURE 21 – DecisionMerge2Eplace_3

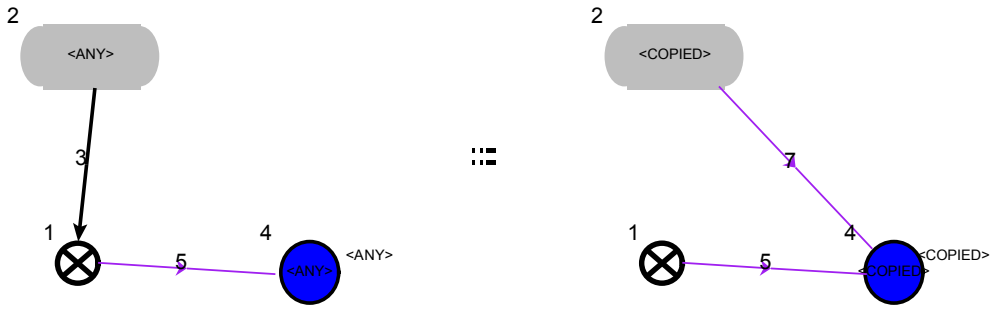



(a) LHS (b) RHS

FIGURE 22 – DecisionMerge2Eplace_4

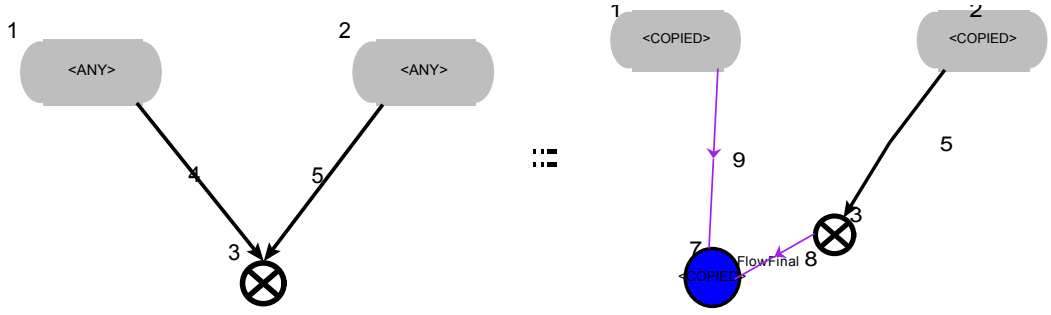

(a) LHS (b) RHS

FIGURE 23 – DecisionMerge2Eplace_5



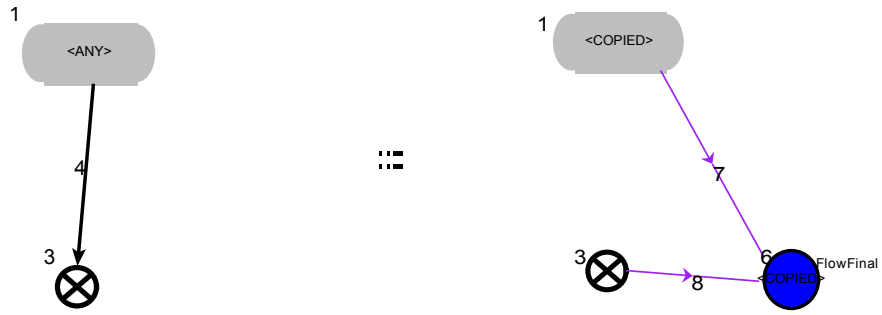

(a) LHS (b) RHS

FIGURE 24 – Action2FlowFinal_1



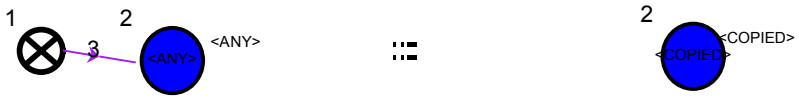

(a) LHS (b) RHS

FIGURE 25 – Action2FlowFinal_2




(a) LHS (b) RHS

FIGURE 26 – Action2FlowFinal_3




(a) LHS (b) RHS

FIGURE 27 – Action2FlowFinal_4_RHS

**ملخص**

**يلعب الوكلاء المتنقلون دورا مهما في مختلف المجالات. حيث يؤثر دقة نمذجتهم على أداء النظم اين يتم استخدامها. في هذه الدراسة ، اقترحنا نهجا جديدا يسمح بنمذجة الوكلاء المتنقلون باستخدام الرسومات البيانية للنشاطات المتنقلة . فهي مفهومة وسهلة لكنها تفتقر الى الدلالات الرياضية ، فهي بذالك لا تسمح بالقيام بالتحليل و التدقيق. قواعد الرسوم البيانية الناتجة باستخدام أداة $ATOM^3$ يسمح التحويل التلقائي للرسومات البيانية للنشاطات المتنقلة الي نظرائهم من شبكات بيتري المتداخلة، الذي يمكن استخدامه كأساس لإجراء مراجعة وتحليل.**

**الكلمات الرئيسية : لغة النمذجة الموحدة، شبكات بتري المتداخلة، الميتا نمذجة، الرسم البياني للنشاطات المتنقلة، القواعد البيانية، التحويلات البيانية، اداة $ATOM^3$**

**ABSTRACT**

Mobile agents play an important role in various fields. Therefor the correctness of their modeling affects the performance of systems where they are used. In this study, we proposed a new approach for modeling mobile agents with the UML mobile activity diagram formalism. They are understandable and easy but lacks of formal semantics, they do not allow the analysis and verification. A graph grammar generated with the tool $ATOM^3$ allows the transformation of UML mobile activity diagram to their equivalent nested Petri nets. This last, can be used as the basis for a possible verification and analysis.

**Keywords :** UML, Mobile Activity Diagram, Nested Petri Net, Meta-modeling, Graph Grammars, Graph Transformations, $ATOM^3$.

**RÉSUMÉ**

Les agents mobiles jouent un rôle important dans divers domaines. Cependant l'exactitude de leur modélisation influe sur la performance des systèmes où ils sont utilisés. Dans cette étude, nous avons proposé une nouvelle approche, permettant la modélisation des agents mobiles avec le formalisme de diagramme d'activités mobile. Ils sont compréhensible et facile mais souffre d'un manque de sémantique formelle, cependant ils ne permettent pas l'analyse et la vérification. Une grammaire de graphe généré avec l'outil $ATOM^3$ permet la transformation automatique des diagrammes d'activités mobiles vers leurs équivalents des réseaux de Petri imbriqués, ces derniers peuvent être utilisés comme base pour une éventuelle vérification et analyse.

**Mots clés :** UML, Diagramme d'Activité Mobile, Réseau de Petri Imbriqué, Méta-modélisation, Grammaire de Graphe, Transformation de Graphe, $ATOM^3$.