Table des matières

Table des figures

Liste des tableaux

Introduction Générale

L'ingénierie système est une démarche méthodologique générale qui englobe l'ensemble des activités adéquates pour concevoir, faire évoluer et vérifier un système apportant une solution économique et performante aux besoins d'un client tout en satisfaisant l'ensemble des parties prenantes.

Depuis longtemps, les ingénieurs système ont utilisé des techniques de modélisation. Parmi les plus connues, on trouve SADT et SA/RT, qui datent des années 80, ainsi que de nombreuses approches basées sur les réseaux de Pétri ou les machines à états finis.

Mais ces techniques sont limitées par leur portée et leur expressivité ainsi que par la difficulté de leur intégration avec d'autres formalismes, ainsi qu'avec les exigences système.

L'essor d'UML dans le domaine du logiciel et l'effort industriel de développement d'outils qui l'accompagne ont naturellement conduit à envisager son utilisation en ingénierie système. Cependant, du fait de sa conception fortement guidée par les besoins du passage à la programmation par objets, le langage était, tout au moins dans ses premières versions, peu adapté à la modélisation des systèmes complexes et donc au support de l'ingénierie système (IS).

La version 2 d'UML, officialisée en 2005, a introduit plusieurs nouveaux concepts et dia- grammes utiles pour l'IS. Mais il restait toujours la barrière psychologique du vocabulaire orienté informatique : classe, objet, héritage, etc. La version 1.0 du nouveau langage de modélisation SysML a été adoptée officiellement par l'OMG le 19 septembre 2007 !

Cependant, SysML 1.0 manque d'outils pour analyser et vérifier des propriétés. D'autre part, les méthodes formelles telles que les réseaux de Pétri offrent un ensemble d'outils pour analyser et vérifier les propriétés.

donc nous proposons une approche permettant de transformer les diagrammes d'états-transitions vers les réseaux de Pétri de type GSPN. Notre approche est basée sur la transformation de graphes sous l'outil de méta-modélisation AtoM3.

Ce mémoire est subdivisé en trois chapitres :

Le premier chapitre présent les concepts de base de l'utilisation du langage SysML 1.0 et surtout les diagrammes d'activité. Le second chapitre est une présentation des réseaux de Pétri. Dans le troisième chapitre, nous présenterons l'outil de modélisation AToM3 et notre approche de transformation des diagrammes d'activité d'SysML 1.0 vers les réseaux de Pétri. Ces modèles PN seront par la suite exploités par l'outil d'analyse PIPE 3.

Chapitre 1

SysML

1.1 Introduction

L'ingénierie système (IS) est une démarche méthodologique pour maîtriser la conception des systèmes et produits complexes. On peut aussi la définir comme « une approche interdisciplinaire rassemblant tous les efforts techniques pour faire évoluer et vérifier un ensemble intégré de systèmes, de gens, de produits et de solutions de processus de manière équilibrée au fil du cycle de vie pour satisfaire aux besoins client ». Les pratiques de cette démarche sont aujourd'hui répertoriées dans des normes, réalisées à l'aide de méthodes et supportées par des outils.

Les normes d'IS décrivent les pratiques du métier en termes de processus et d'activités de manière invariante par rapport aux domaines d'application de l'ingénierie système. Les méthodes d'ingénierie système fournissent des démarches techniques pour réaliser ces activités générales. Contrairement aux normes, elles dépendent des secteurs d'application et résultent de choix industriels. La mise en œuvre des processus et des méthodes est assistée par des outils, aujourd'hui très généralement informatisés.

1.2 Les normes d'IS

Trois normes générales d'ingénierie système décrivant les processus du métier d'IS sont actuellement disponibles. Elles en définissent les types d'activité à réaliser et les types de résultat produit. Elles recouvrent des champs différents, de manière d'autant plus approfondie que leur champ est limité, et ainsi elles se complètent[2].

• **IEEE 1 220 :** (Standard for Application and Management of the Systems Engineering Process) Issue du standard militaire MIL STD 499B, cette norme de l'IEEE, dont la version initiale date de 1994, se focalise sur les processus techniques d'ingénierie système allant de l'analyse des exigences jusqu'à la définition physique du système.

EIA 632 : (Processes for Engineering a System) Cette norme de l'EIA complète les processus techniques de définition du système en couvrant la réalisation des produits jusqu'à leur mise en service (transfert vers l'utilisation). De plus, elle incorpore les processus contractuels d'acquisition et de fourniture.

ISO 15 288 : (Systems engineering – System life cycle processes) Inspirée sur le plan de la forme par la norme ISO/CEI 12 207 – AFNOR Z67 - 150 (Typologie des processus du cycle de vie du logiciel), cette norme de l'ISO étend les processus techniques à tout

le cycle de vie du système (elle couvre ainsi les processus d'exploitation, de maintien en condition opérationnelle et de retrait de service).

1.3 Pourquoi SysML ?

Le monde du logiciel a fini par se mettre d'accord à la fin des années 90 sur un formalisme de modélisation unifié : UML.Par contre, de leur côté, les ingénieurs système n'ont pas réussi à faire émerger un langage de modélisation uniformisé, malgré les tentatives de standardisation des processus d'ingénierie système évoquées précédemment[2].

En 2003, l'INCOSE a décidé de faire d'UML ce langage commun pour l'IS. UML contenait en effet déjà à l'époque nombre de diagrammes indispensables, comme les diagrammes de séquence, d'états, de cas d'utilisation, etc. Le travail sur la nouvelle version UML 2, entamé à l'OMG à peu près à la même période, a abouti à la définition d'un langage de modélisation très proche du besoin des ingénieurs système, avec les améliorations notables aux diagrammes d'activité et de séquence, ainsi que la mise au point du diagramme de structure composite[2].

Cependant, il restait une barrière psychologique importante à l'adoption d'UML par la communauté de l'IS : sa teinture « logicielle » indélébile ! La possibilité d'extension d'UML, grâce au concept de stéréotype, a permis d'adapter le vocabulaire pour les ingénieurs système. En éliminant les mots « objet » et « classe » au profit du terme plus neutre « bloc », bref en gommant les aspects les plus informatiques d'UML, et en renommant ce langage de modélisation, l'OMG veut promouvoir SysML comme un nouveau langage différent d'UML, tout en profitant de sa filiation directe[2].

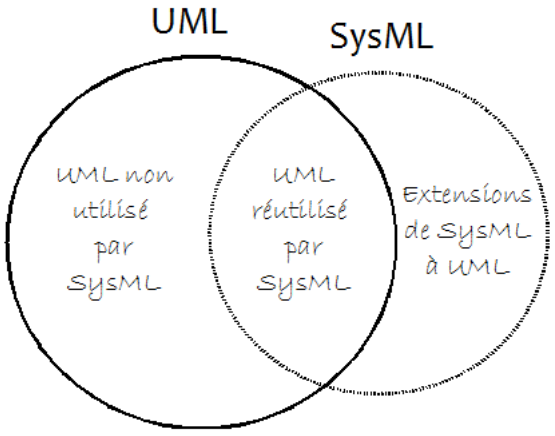


FIGURE 1.1 – SysML en tant que profil UML 2

L'OMG a annoncé l'adoption de SysML en juillet 2006 et la disponibilité de la première version officielle(SysML v. 1.0) en septembre 2007. Très récemment, une nouvelle spécification SysML v. 1.1 a été rendue publique (décembre 2008)[2].

SysML s'articule autour de neuf types de diagrammes, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système. Ces types de diagrammes sont répartis par l'OMG en trois grands groupes :

- **quatre diagrammes comportementaux :**

1. diagramme d'activité (montre l'enchainement des actions et décisions au sein d'une activité complexe) .

2. diagramme de séquence (montre la séquence verticale des messages passés entre blocs au sein d'une interaction) .

3. diagramme d'états (montre les différents états et transitions possibles des blocs dynamiques) .

4. diagramme de cas d'utilisation (montre les interactions fonctionnelles entre les acteurs et le système à l'étude) .

- **un diagramme transverse :** diagramme d'exigences (montre les exigences du système et leurs relations) .

- **quatre diagrammes structurels :**

1. diagramme de définition de blocs (montre les briques de base statiques : blocs, compositions, associations, attributs, opérations, généralisations, etc.) .

2. diagramme de bloc interne (montre l'organisation interne d'un élément statique complexe) .

3. diagramme paramétrique (représente les contraintes du système, les équations qui le régissent) .

4. diagramme de packages (montre l'organisation logique du modèle et les relations entre packages).

La structure du système est représentée par les diagrammes de blocs. Le diagramme de définition de blocs (block definition diagram) décrit la hiérarchie du système et les classifications système/composant. Le diagramme de bloc interne (internal block diagram) décrit la structure interne du système en termes de parties, ports et connecteurs.

Le diagramme de packages est utilisé pour organiser le modèle. Les diagrammes comportementaux incluent le diagramme de cas d'utilisation, le diagramme d'activité, le diagramme de séquence et le diagramme de machines à états. Le diagramme de cas d'utilisation fournit une description de haut niveau des fonctionnalités du système. Le diagramme d'activité représente les flots de données et de contrôle entre les actions. Le diagramme de séquence représente les interactions entre les parties du système qui collaborent. Le diagramme de machines à états décrit les transitions entre états et les actions que le système ou ses parties réalisent en réponse aux événements.

Le diagramme d'exigences capture les hiérarchies d'exigences, ainsi que leurs relation de dérivation, de satisfaction, de vérification et de raffinement. Ces relations fournissent la capacité de relier les exigences les unes aux autres, ainsi qu'aux éléments de conception et aux cas de tests.

Le diagramme paramétrique permet de représenter des contraintes sur les valeurs de paramètres système tels que performance, fiabilité, masse, etc. Il s'agit d'une spécialisation du diagramme de bloc interne où les seuls blocs utilisables sont des contraintes entre paramètres permettant de représenter graphiquement des équations et relations mathématiques. Ce nouveau diagramme fournit ainsi un support pour les études d'analyse système.

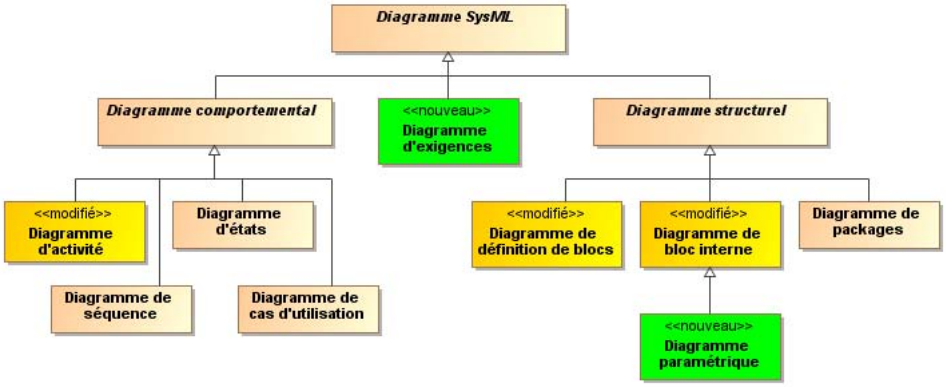


FIGURE 1.2 – Les 9 types de diagrammes SysML

SysML inclut également une relation d'allocation pour représenter différents types d'allocation, comme celles de fonctions à des composants, de composants logiques à composants physiques, ainsi que de software à hardware[2].

1.4 Différences UML 2/SysML

Les diagrammes UML 2 qui n'ont pas été retenus par SysML, principalement par souci de simplification, sont :

- **le diagramme d'objets .**
- **le diagramme de composants .**
- **le diagramme de déploiement .**
- **le diagramme de vue d'ensemble des interactions .**
- **le diagramme de communication .**

Récapitulons : 13 diagrammes (UML 2) – 6 + 2 (exigences, paramétrique) = 9 diagrammes SysML !

1.5 Le diagramme d'activité

Le diagramme d'activité est l'un des diagrammes dynamiques proposés par SysML. Il ressemble fondamentalement à un traditionnel diagramme fonctionnel (à la SADT ou SA/RT), montrant le flot de contrôle d'action en action. Mais il propose des capacités supplémentaires importantes comme celle de pouvoir faire le lien avec les blocs de la modélisation structurelle et celle de pouvoir modéliser des flux continus.

1.5.1 Éléments de diagramme

Les principaux éléments qui composent les diagrammes d'activités sont présentés à la figure 1.3 .

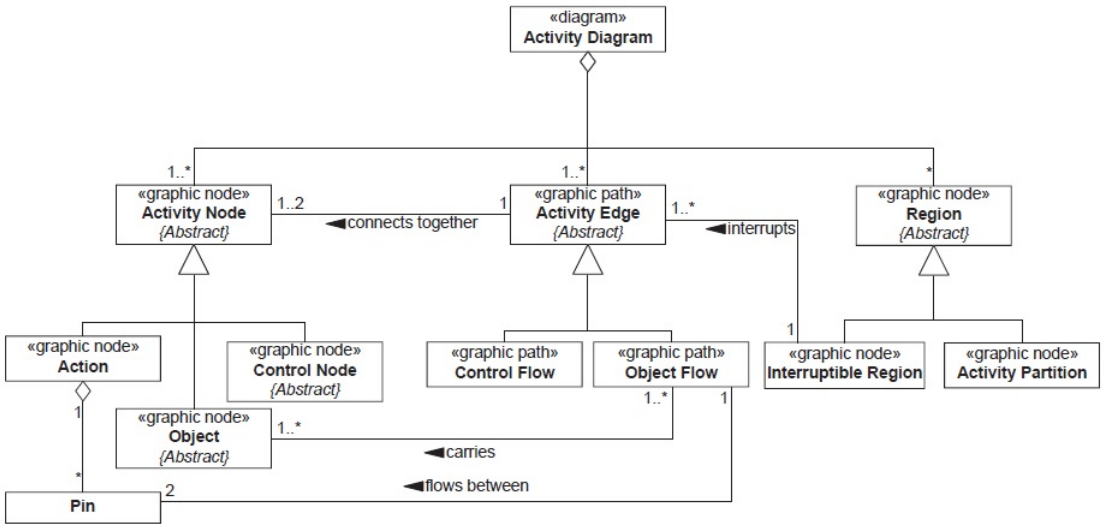


FIGURE 1.3 – Méta-modèle partiel pour le diagramme d'activité

La Figure 1.3 montre un méta-modèle partiel pour les diagrammes d'activité. Il montre qu'un «Diagramme d'activité» est constitué de trois éléments de base : un ou plusieurs «Nœud d'activité», un ou plusieurs «Avantage d'activité» et zéro ou plus «Région». Il y a trois types principaux de 'Nœud d'activité', qui sont les 'Action', les 'Object' et les 'Nœud de Contrôle' qui seront tous discutés plus en détail plus loin dans cette section. L '«Action» est l'endroit où l'accent est mis sur ces diagrammes et représente une unité de comportement sur le «Diagramme d'activité». Il existe de nombreux types d'actions disponibles dont la discussion dépasse la portée de ce livre. Nous les traiterons tout de même, mais pour une discussion complète. Une «action» peut également avoir zéro ou plus «épingle», qui peut être utilisé pour montrer un «flux d'objets» qui porte un «objet». Ceci est discuté plus loin[6].

Un «bord d'activité» relie un ou deux «nœuds d'activité» ; il peut relier un «nœud d'activité» à lui-même, d'où la multiplicité d'un ou deux, plutôt que seulement deux. L'élément 'Edge d'activité' a deux types principaux : 'Flux de contrôle' et 'Flux d'objets'. Un «flux de contrôle» est utilisé pour afficher les routes principales à travers le «diagramme d'activité» et relie ensemble un ou deux «nœuds d'activité». Un «flux d'objets» est utilisé pour montrer le flux d'informations entre un ou plusieurs «nœuds d'activité» et le fait en portant le type «objet» de «nœud d'activité»[6].

L'autre élément majeur d'un diagramme d'activités dans la «Région» comprend deux types principaux : «Région interruptible» et «Partition d'activité». Une «région interruptible» permet de placer une limite dans un diagramme d'activités qui contient les actions qui peuvent être interrompues. Ceci est particulièrement puissant pour les systèmes où le comportement peut être interrompu par des conditions atypiques, telles que des interruptions logicielles et des situations d'urgence. Par exemple, par une interaction directe avec l'utilisateur ou une sorte d'événement d'urgence. La 'Partition d'activité' est le mécanisme utilisé pour visualiser les couloirs de natation qui permettent de regrouper

différentes actions pour une raison quelconque, généralement pour montrer la responsabilité des actions[6].

Le diagramme de la Figure 1.4 montre une vue étendue des types de 'Nœud de Contrôle' qui existent dans SysML. La plupart d'entre eux vont ensemble dans deux ou trois, donc seront discutés ensemble.

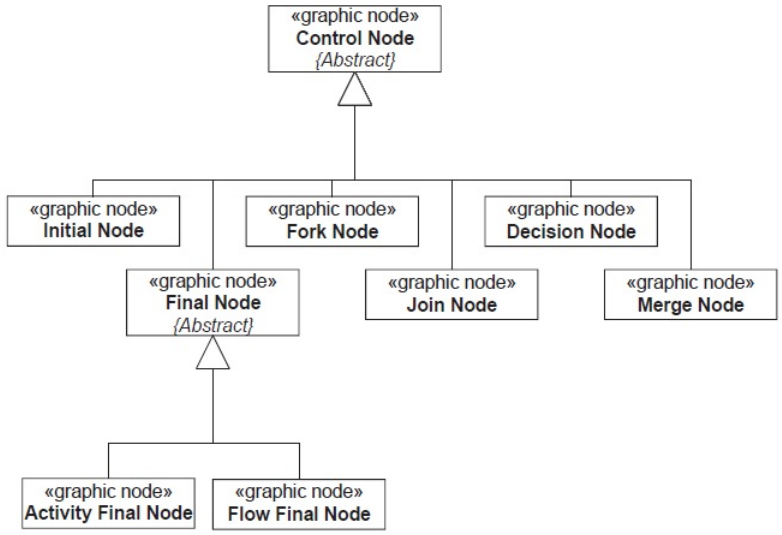


FIGURE 1.4 – Méta-modèle partiel étendu du diagramme d'activité, en se concentrant sur le 'Noeud de contrôle'

- Le 'Noeud initial' indique où commence le diagramme d'activité. Inversement, la fin du diagramme d'activité est indiquée par le 'Nœud Final d'Activité'. Le 'noeud final de flux' permet de terminer un flux particulier sans terminer le diagramme. Par exemple, imaginez une situation où il y a deux flux de contrôle parallèles dans un diagramme et où l'un doit être arrêté tandis que l'autre continue. Dans ce cas, un nœud de flux final serait utilisé car il termine un seul flux, mais permet au reste du diagramme de continuer[6].

- Le «nœud de fourchette» et le «nœud de jonction» permettent de diviser le flux d'un diagramme d'activités en plusieurs chemins parallèles, puis de les rejoindre à un point ultérieur du diagramme. Les noeuds fork et join (ou les forks et jointures comme on les appelle généralement) utilisent un concept de passing token, ce qui signifie que lorsqu'un flux est divisé en flux parallèles par une fourche, alors imaginez que chaque flux a reçu un jeton. Ces flux ne peuvent être recollés que lorsque tous les jetons sont présents sur le flux de jointure. Il est également possible de spécifier une condition booléenne sur la jointure pour créer des règles plus complexes permettant de rejoindre les flux[6].

- Les nœuds de décision et de fusion se complètent également. Un «noeud de décision» permet à un flux de se ramifier sur une route particulière en fonction d'une condition de garde, tandis qu'un «nœud de fusion» permet à plusieurs flux d'être fusionnés en un seul flux[6].

Il existe trois types de symboles qui peuvent être utilisés sur un diagramme d'activité pour montrer le flux d'informations véhiculées par un «flux d'objets» : le «nœud d'objet», le «signal» et l '«événement». Voir la Figure 1.5 Le 'Object Node' est utilisé pour représenter les informations qui ont été représentées ailleurs dans le modèle par un bloc

et qui forment une entrée ou une sortie d'une action. On peut penser à une représentation d'une spécification d'instance. Le symbole 'Événement' est utilisé pour afficher un événement entrant dans un diagramme d'activité, tandis qu'un 'Signal' est utilisé pour montrer un événement quittant un diagramme d'activité. Ils correspondent à des événements de réception et envoient des événements d'un diagramme de machine d'état. Il existe un type spécial d '«événement», connu sous le nom d' «événement temporel», qui permet la visualisation d'événements de synchronisation explicites.

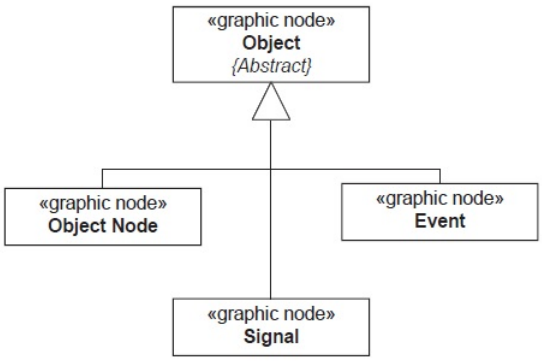


FIGURE 1.5 – Méta-modèle partiel étendu pour le diagramme d'activité, en se concentrant sur le 'noeud d'objet'

Chacun de ces éléments de diagramme peut être réalisé par des nœuds graphiques ou des chemins graphiques, comme indiqué par leurs stéréotypes, et est illustré à la Figure 1.6 .

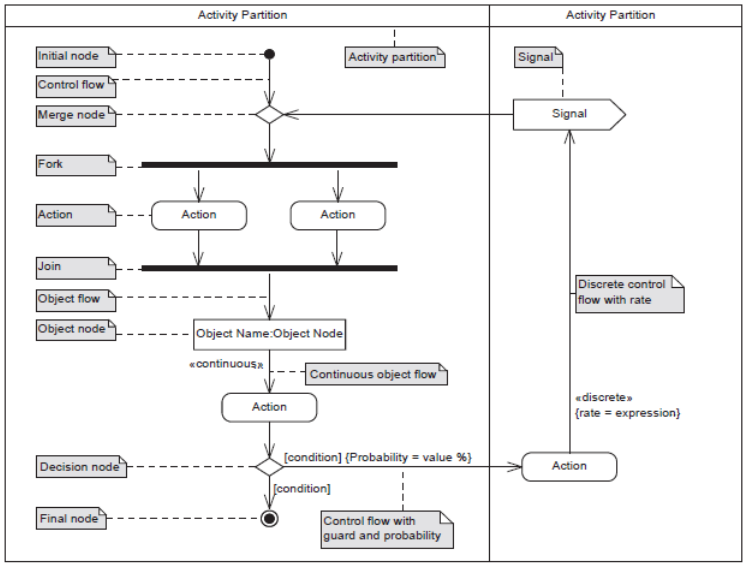


FIGURE 1.6 – Résumé de la notation du diagramme d'activité

En plus des éléments mentionnés jusqu'à présent, SysML a une notation qui peut être appliquée à un «bord d'activité» et un «nœud d'objet». Cette notation utilise la notation de contrainte et de stéréotype existante qui est déjà présente dans SysML et définit simplement certaines contraintes et stéréotypes standard à utiliser sur les diagrammes d'activité.

La première de ces notations permet d'appliquer un taux à un «bord d'activité» (et, plus spécifiquement, à un «flux d'objets») afin de donner une indication de la fréquence à laquelle l'information circule le long du bord. Les flux peuvent être discrets ou continus. Ceci est démontré par l'utilisation des stéréotypes «discrets» ou «continus» placés sur le flux. Alternativement, le taux réel peut être affiché en utilisant une contrainte de la forme : rate = expression. Par exemple, si des données ou du matériel sont transmis le long d'un flux toutes les minutes, cela peut être indiqué en plaçant la contrainte rate = per 1 minute sur le flux.

La deuxième notation permet d'appliquer une probabilité à un «bord d'activité» (généralement sur les bords du «flux de contrôle» laissant un «nœud de décision») et indique la probabilité que le bord soit traversé. Il peut être représenté par un nombre compris entre 0 et 1 ou en pourcentage. Toutes les probabilités sur les bords avec la même source doivent totaliser 1 . Il est important de noter que le bord réel traversé est régi par les conditions de garde sur le «nœud de décision» et non par la probabilité. La probabilité n'est rien de plus qu'une information supplémentaire qui peut être ajoutée au diagramme.

L'autre notation modifie le comportement d'un 'Object Node' et est indiquée par l'utilisation des stéréotypes «nobuffer» et «overwrite». Si un nœud d'objet est émis par une action et n'est pas immédiatement consommé par son action de réception, ce nœud d'objet peut bloquer l'opération de l'action d'origine jusqu'à ce qu'elle soit consommée par l'action de réception. «Nobuffer» et «overwrite» modifient ce comportement :

- «nobuffer» signifie que le nœud d'objet marqué est immédiatement supprimé si l'action de réception n'est pas prête à le recevoir. L'action d'origine ne sera pas bloquée et peut continuer à générer des noeuds d'objet, qui seront supprimés si elle n'est pas encore nécessaire.

- «écraser» signifie que le nœud de l'objet marqué est écrasé si l'action de réception n'est pas prête à le recevoir. L'action d'origine ne sera pas bloquée et peut continuer à générer des nœuds d'objet. Le dernier généré écrasera le précédent s'il n'est pas encore nécessaire.

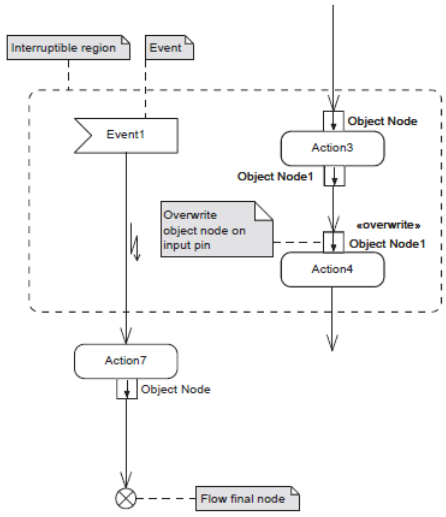


FIGURE 1.7 – Notation de diagramme d'activité pour montrer des régions interruptibles et utilisation de broches plutôt que de noeuds d'objets

La Figure 1.7 montre une notation supplémentaire qui couvre les régions interruptibles et l'utilisation de broches plutôt que de nœuds d'objets.

Les régions interruptibles sont représentées par une boîte à lumière en pointillés entourant la région à interrompre. Il doit toujours y avoir un flux de contrôle normal à travers la région interruptible. Dans cet exemple, le flux est en 'Action3' puis en 'Action4' et ensuite en dehors de la région. Il doit également y avoir un événement qui provoque l'interruption : 'Event1' dans l'exemple. L'événement est connecté par un flux de contrôle à une action en dehors de la région interruptible, qui agit comme un gestionnaire d'interruption : 'Action7' dans l'exemple. Le flux de contrôle est soit annoté avec un symbole de la foudre, comme ici, ou peut être dessiné comme un tel éclair. Dans l'exemple ci-dessus, la région interruptible montre que pendant que 'Action3' ou 'Action4' ont lieu, ils peuvent être interrompus par 'Event1', ce qui entraînera le transfert du contrôle vers 'Action7'.

Le diagramme montre également la notation d'un nœud final de flux et montre comment les broches peuvent être utilisées à la place des nœuds d'objets explicites. La partie du diagramme impliquant 'Action3' et 'Action4' est équivalente à celle montrée dans la Figure 1.8 .

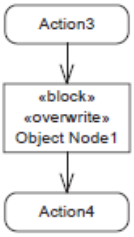


FIGURE 1.8 – Notation de noeud d'objet équivalente à la notation de broche

Quelle notation est la meilleure, les pins ou les nœuds d'objet, est une question de préférence personnelle (et peut-être des directives et des options de diagrammes organisationnels disponibles dans votre outil SysML). Les auteurs sont nettement en faveur des nœuds d'objets explicites plutôt que de la version utilisant des pins.

1.6 Conclusion

Les diagrammes d'activité sont des diagrammes comportementaux SysML très puissants, qui peuvent être utilisés pour afficher à la fois les comportements de bas niveau, tels que les opérations, et les comportements de haut niveau, tels que les processus. Ils sont très bons pour aider à assurer la cohérence du modèle .

Chapitre 2

Réseaux de Petri

2.1 Introduction

Les Réseaux de Pétri (RdP) permettent de modéliser des systèmes séquentiels. Ils ont été inventés par Carl Adam Pétri, un mathématicien Allemand contemporain (d'où l'absence d'accent dans Pétri). Il a défini un outil mathématique très général permettant de décrire les relations existant entre des conditions et des évènements et de modéliser le comportement de systèmes dynamiques à évènements discrets. Ces RdP datent de 1960-1962. C'est un outil très général, modélisant aussi bien les protocoles de communication informatiques que des systèmes de production. Il est à l'origine du Grafcet (ce dernier étant spécialisé dans la description de la commande de systèmes automatisés).[15]

2.2 Définition sous la forme d'un graphe

Un réseau de Pétri est un graphe biparti orienté avec deux types de nœuds et un comportement dynamique (une sémantique opérationnelle). La première façon de définir formellement les réseaux de Pétri est donc de s'appuyer sur les graphes. exemple de réseau de Petri est illustré par la figure 2.1 .

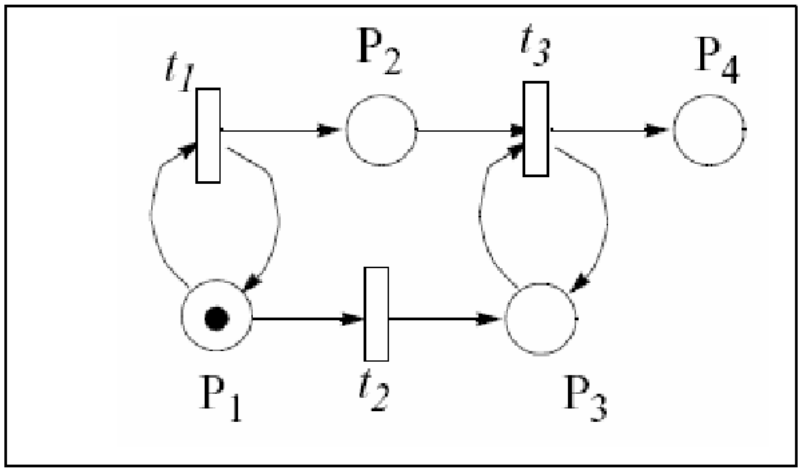


FIGURE 2.1 – Exemple de réseau de Petri

2.3 Structure du réseau de Pétri

On définit un Rdp par le 5-uplet (P, T, Pre, Post, M0) où :[15]

- **P :** est l'ensemble de toutes les places du réseau.
- **T :** est l'ensemble de toutes les transitions du réseau.
- **Pre :** est la matrice d'incidence avant, c'est la matrice qui contient les poids sur les arcs reliant les places aux transitions.
- **Post :** est la matrice d'incidence arrière, c'est la matrice qui contient les poids sur les arcs reliant les transitions aux places.
- **M0 :** est le marquage initial.

2.4 Règles de franchissement

Le nombre et la disuibution des jetons dans un réseau de Petri peut changer au cours de l'évolution du système. Ce changement est réalisé par des franchissements de transitions. Ces franchissements se font en accord avec des règles que nous présentons plus loin. Avant de présenter ces règles, nous définissons quelques terrnes en relation avec les fonctions avant et arrière d'une transition ou d'une place.

Si O(t, p) * O (resp. l(p, t) * 0), on dit que p ost une place de sortie (resp. place d'entrée) de la transition t, et que t est une transition d'entrée (resp. transition de sortie) de la place p. On utilisera les notations suivantes :

Places d'entrée d'une transition. On note $\bullet t = p|l(p,t)>0$ I'ensernble des places d'entrée de la transition t.

Places de sortie d'une transition. On note $\bullet t = p|O(p,t)>0$ I'ensemble des places de sortie de la ransition t.

Transitions d'entrée d'une place. On note $\bullet p = t|l(t,p)>0$ I'ensemble des transitions d'entrée de la place p.

Transitions d'entrée d'une place. On note $\bullet p = t|O(t,p)>0$ I'ensemble des transitions de sortie de la place p.

2.4.1 Notion de changement d'état de règle de franchissement[26]

Un changement d'état est dénoté par un mouvement de jetons (points noirs) de places d'entrée vers des places de sortie d'une transition. Ce mouvement est causé par le franchissement d'une transition. Le franchissement représente une occurrence d'un événement ou d'une action. Le franchissement est conditionné par la présence de jetons dans les places d'entrée de la transition. Quand toutes les places d'entrée à une transition sont suffisamment marquées, la transition peut tirer et alors les jetons sont retirés des places d'entrée (ancien état) et ajoutés à toutes les places de sortie (Nouvel état).

Une règle de franchissement est une simple relation de transition qui définit le changement d'état dans un réseau marqué lors de l'exécution d'une action. Afin de définir une règle de franchissement, il est nécessaire de formaliser quand le réseau peut exécuter une action : on dit qu'une transition t T peut être franchie à partir d'un marquage M (qui représente l'état du system à un instant donné) si et seulement si chaque place d'entrée p *t de la transition t contient au moins un nombre de jetons qui est supérieur ou égale au poids de l'arc reliant cette place d'entrée p avec la transition t tel que : M (p) W(p, t) p P.

Une règle de franchissement est définie par M'(p) = M (p) - W(p, t) + W (t, p) pour tout p p ce qui veut dire que lorsque la transition t est franchie à partir d'un marquage M, il faut saisir W(p, t) jetons à partir de chaque place d'entrée à la transition t et déposer W(t, p) jetons dans chaque place de sortie de la transition t ce qui permet de produire un nouveau marquage M'

Le franchissement d'une transition t, dénoté par M [t M,'est dit l'occurrence de t. On dit que deux transitions t1, t2 (pas certainement distinctes) sont franchies en concurrence par un marquage M si et seulement si M (p) W (p, t1) + W (p, t2) pour toute p P.

Cette vision de l'exécution concurrente de deux transitions dans un RdP est contradictoire avec celle qui impose que deux occurrence de transition sont parallèles si et seulement si : elles sont causalement indépendantes et n'ont pas une relation de conflit entre elles. Deux occurrences sont en conflit si l'une des deux peut avoir lieu mais pas toutes les deux.

2.4.2 Exécution d'un réseau de Petri : Notion de marquage

L'exécution d'un réseau de Petri est définie par un ensemble de séquences d'occurrence.

Une séquence d'occurrence est une séquence de transitions franchissables dénotée par =M0 t1 M1 t2...tel que Mi -1 [ti Mi. Une séquence t1 t2... est une séquence de transitions (commencée par le marquage M) si et seulement si il existe une séquence d'occurrence M0 t1 M1... avec M=M0. Si la séquence finie t1 t2 ...tn conduit à un nouveau marquage M ' à partir du marquage M, on écrit M [t1 t2...tn M' ou simplement M[t1t2...tn si on ne veut pas spécifier le marquage résultat[26].

L'ensemble de marquages accessibles d'un réseau marqué (P, T, W, M0) est défini par

[M0 = M t2...tn : M0 [t1t2...tn M.

Exemple

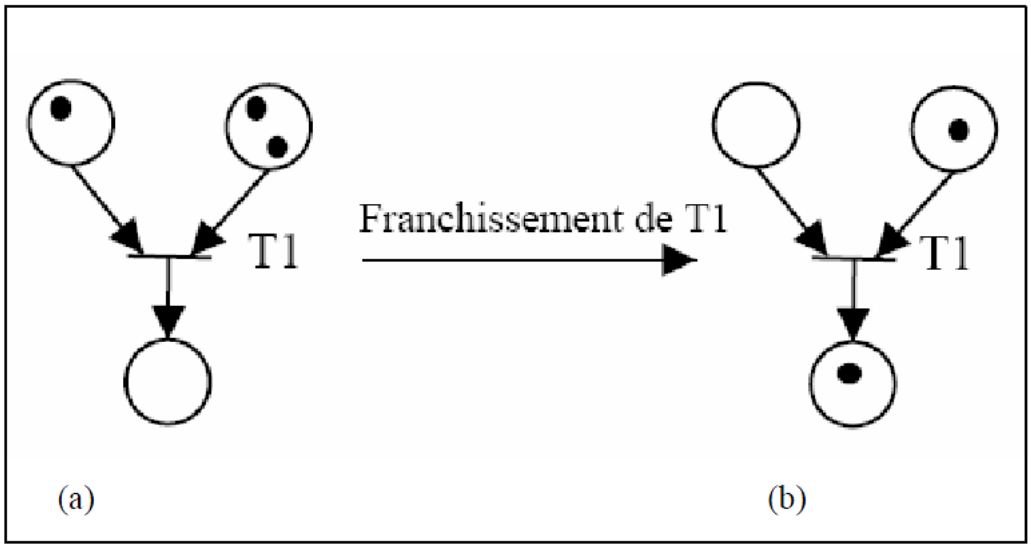


FIGURE 2.2 – RdP marqué avant franchissement de T1 (b) RdP après franchissement de T1[26].

2.4.3 Représentation matricielle

Une représentation matricielle d'un RdP est offerte afin de simplifier les tâches d'analyse et de vérification effectuées sur un modèle RdP. Agir sur une représentation graphique d'un modèle RdP est une tâche délicate en la comparant avec une représentation matricielle. Il est possible de représenter la fonction W (fonction de poids) par des matrices.

a) Définition

Soit Un réseau de Petri R= (P, T, W) avec P= p1, p2, ..., pm et T= t1, t2, ..., tn On appelle matrice des pré-conditions pré, la matrice m × n à coefficients dans N telle que pré (i,j)= W(pi, tj) indique le nombre de marques que doit contenir la place pi pour que la transition tj devienne franchissable. De la même manière on définit la matrice des postconditions post, la matrice n × m telle que post (i,j)= W(tj, pi) contient le nombre de marques déposées dans pi lors du franchissement de la transition tj. La matrice C= post – pré est appelée matrice d'incidence du réseau (m représente le nombre de places d'un réseau de Petri et le nombre de transitions).[26]. Marquage d'un réseau de Petri est représenté par un vecteur de dimension m à coefficients dans N. La règle de franchissement d'un réseau de Petri est définie par :

M' (p) =M (p) + C (p, t)

Exemple

Pour le réseau de la figure 2.1 :

P= p1, p2, p3, p4 T= t1, t2, t3

$$Pré = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad Post = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

La matrice d'incidence C est :

$$C = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Le vecteur de marquage *M* est :

$$M = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

FIGURE 2.3 – Matrice d'incidence et vecteur de marquage du RdP de l'exemple de la figure 2.1[26]

2.5 Modélisation Avec les Réseaux de Petri

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

2.5.1 Parallélisme

Dans le réseau de Petri représenté par la figure 2.4 le franchissement de la transition T1 met un jeton dans la place P1 et un jeton dans la place P2 .

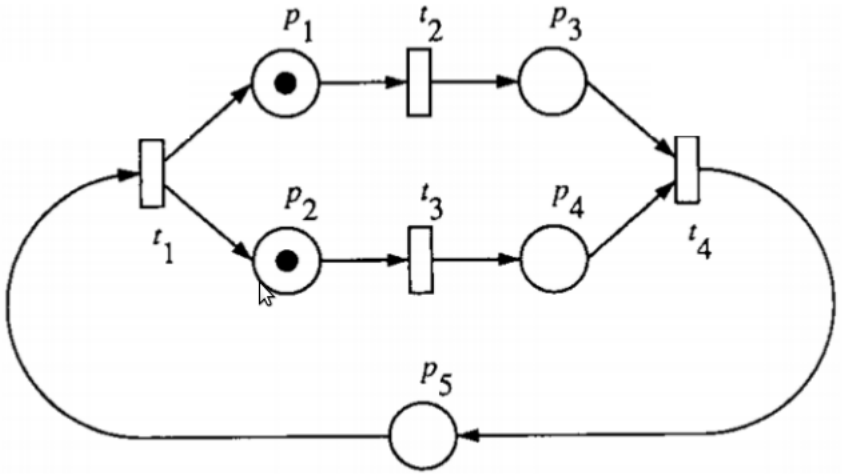


FIGURE 2.4 – Parallélisme dans les Réseaux de Petri

2.5.2 Synchronisation

Ce problème peut être résolu par un réseau de Petri comme celui de la figure 2.5. 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 p1 ou p2 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 t1 et t2 seront en conflit. Seulement l'une d'eux peut être franchie. Le franchissement de t1 désactive t2 , 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.

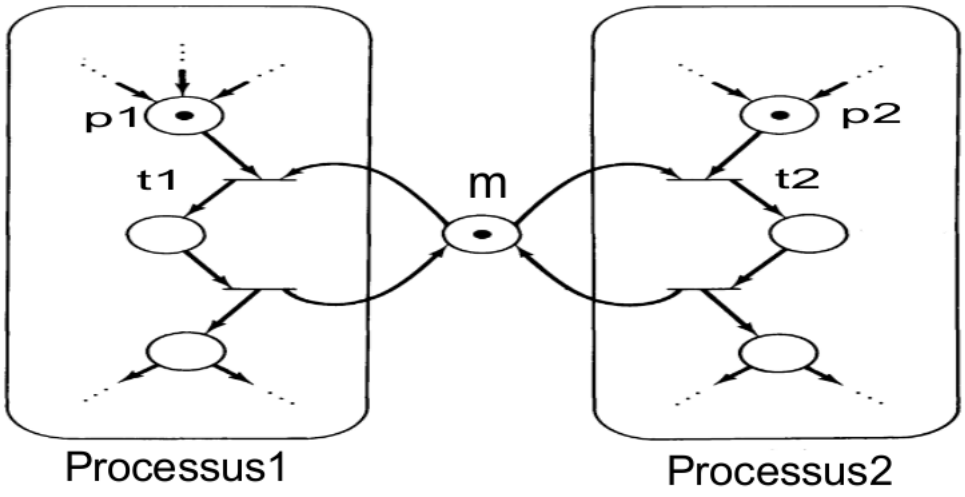


FIGURE 2.5 – Exclusion mutuelle

2.6 Quelques propriétés

Dans ce qui suit, nous allons définir quelques propriétés utiles :[27]

• **Sureté (Safety) :** cette propriété assure qu'un mauvais comportement ne pourra jamais arriver dans un système.

• **Vivacité (Liveness) :** un comportement valide finira par se réaliser.

• **Adjacent :** pour t T, Adj(t) = t' T | (Pre(t) + Post(t)) X (Pre(t') + Post(t')) 0 désigne l'ensemble de transitions qui sont connectées à au moins une place p qui est en entrée ou en sortie de t.

• **Visibilité :** une transition t est dite visible par rapport à une propriété si l'une de ses places en entrée ou en sortie est contenue dans cette propriété.

• **Inter-blocage (Deadlock) :** C'est atteindre un marquage où on pourrait plus franchir des transitions. C'est-à-dire qu'il existe un marquage Md tel que l'ensemble de transitions sensibilisées ou franchissables dans ce marquage est vide : En(Md) = .

2.7 Méthodes d'analyse des réseaux de Petri

les réseaux de Petri peuvent modéliser des systèmes à événemenrs discrets. Ces modèles doivent ensuite être analysés. De nombreux travaux ont été consacrés àI'analyse des réseaux de Petri. Nous passons certains d'enrre eux en revue dans ce paragraphe.

Læs méthodes d'analyse des réseaux de Petri peuvent être classées en quatre catégories :[26]

1) méthodes faisant intervenir I'algèbre linéaire.

2) méthodes basées sur I'arbre des marquages atteignables ou sur I'arbre de recouvrement.

3) techniques de réduction.

4) méthodes utilisant les verrous et les trappes.

2.7.1 Analyse par I'arbre des marquages

L'idée la plus naturelle pour étudier les propriétés d'un RdP est de construire le graphe de tous ses marquages accessibles. Le graphe des marquages accessibles est un graphe dont chaque sommet correspond à un marquage accessible et dont chaque arc correspond au franchissement d'une transition permettant de passer d'un marquage à l'autre. Deux situations peuvent alors se présenter :

1. Le graphe est fini

C'est la situation la plus favorable car dans ce cas toutes les propriétés peuvent être déduites simplement par inspection de celui-ci. Nous avons déjà vu plusieurs exemples de cette utilisation.

2. Le graphe est infini

Dans ce cas, on construit un autre graphe appelé "graphe de couverture" permettant de déduire certaines propriétés.

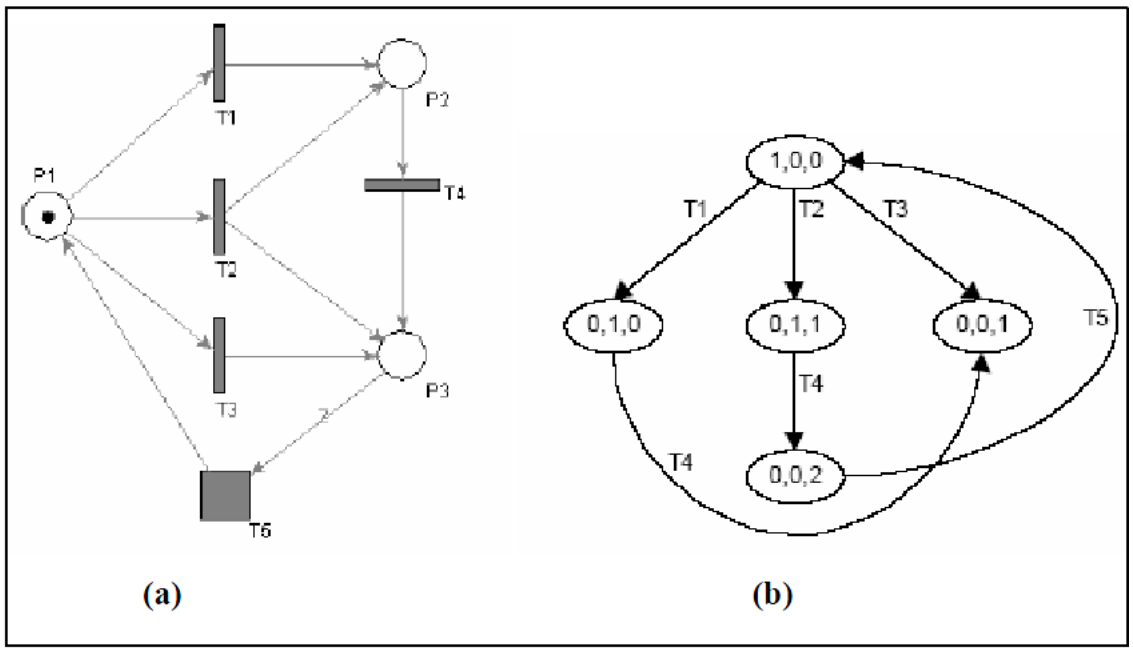


FIGURE 2.6 – (a)Réseau de Petri (b) Graphe des marquages fini

2.7.2 Analyse par algèbre linéaire

L'analyse par algèbre linéaire permet d'étudier des propriétés d'un réseau (caractère borné, vivacité) indépendamment d'un marquage initial. De ce fait, on parlera de propriétés structurelles du réseau. Par exemple, on pourra dire qu'un réseau est structurellement borné s'il est borné pour tout marquage initial fini. De la même façon, si pour tout marquage initial, le réseau est vivant, on dira que le réseau est structurellement vivant.

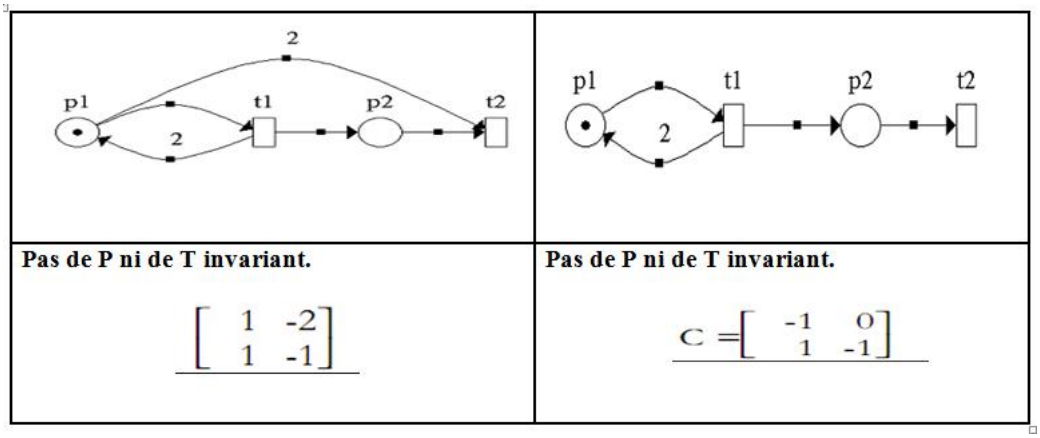


FIGURE 2.7 – exemple Analyse par algèbre linéaire

2.7.3 Analyse par réduction

Pour l'analyse des propriétés d'un RdP, des difficultés peuvent apparaître dans l'utilisation du graphe de marquages, et même dans l'algèbre linéaire dans le cas d'un RdP de taille significative. L'objectif de la réduction est de présenter des règles permettant d'obtenir à partir d'un RdP marqué, un RdP marqué plus simple, c'est-à- dire avec un nombre réduit de places et un nombre réduit de transitions (règles de réduction).

Ce dernier doit être :

1.équivalent au RdP marqué de départ (pour les propriétés étudiées).

2. suffisamment simple pour que l'analyse de ses propriétés soit simple.

En général, le RdP simplifié ne peut pas s'interpréter comme le modèle d'un système.

On peut distinguer deux types de règles de réduction :

1. règles de réduction préservant la vivacité et la born étude (et leurs propriétés associées).

2. règles de réduction préservant les invariants de marquage.

2.7.4 Analyse par les verrous et les trappes

[26]

Considérons S (resp. Q) $\subseteq Punensemblenonvidedeplacesdansunréseaude Petri N = (P, T, I, O >, Lesdéfinitionsdevef fou(resp.trappe)sontlessuivanres$:

Verrou (ou blocage) : S est un verrou si pour chaque transition t, le fait que t ait une place de sortie dans S implique qu'il a aussi une place d'entrée dans S. PIus simplement, S est un verrou si et seulement si $\bullet$S$\subseteq$ ĹS..

Si les places d'un verrou ne sont pas marquées par le marquage initial, alors elles restent toujours non marquées, quelle que soit l'évolution du réseau. Lrurs transitions d'entrée et leurs transitions de sortie ne sont donc pas vivantes. Ceci implique donc que le réseau n'est pas vivant.

Trappe : Q est une trappe si elle satisfait les deux conditions suivantes :

1) Si une transition t a une place d'entrée dans Q, alors elle a aussi une place.de sortie dans Q, i.s. $\bullet$Q$\supseteq$ QĹ;

2) $\forall t \in Q$Ĺ, $\exists p \in Q \bigcap t$Ĺ$telquep$Ĺ$= \emptyset ouO(t, p) \geqslant minI(p, t')$.

Dans les réseaux de Petri ordinaires, la deuxième condition est satisfaite automatiquement.

Si une trappe contient des jetons, alors elle contiendra des jetons cluelle que soit l'évolution ultérieure du réseau.

2.8 Conclusion

Nous avons présenté dans ce chapitre les réseaux de Petri de façon générale et ses propriétés et nous avons vue aussi l'extension de réseau de Petri stochastique et celui stochastique généralisé. Nous avons discuté comment introduire la reconfiguration dans RdP de sorte que les propriétés restent décidable. Aussi, nous avons abordé brièvement les notions de base de la transformation de graphe et plus précisément celles de l'INRS. Dans le chapitre suivant, nous allons présenter l'approche INRS avec plus de détails, ainsi que notre extension de cette approche pour les PN.

Chapitre 3

Approche de Transformation

3.1 Introduction

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 vers les réseaux de Petri. Enfin, nous conclurons par un rappel sur les principes de la méthode proposée.

3.2 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 [9].

• L'idée centrale de la modélisation est de produire une version réduite du système désiré afin de déterminer et d'évaluer ses propriétés saillantes [10].

• 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 [11].

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) [12].

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 modélisation lui-même.

Un méta-modèle est aussi un modèle, il a deux principales caractéristiques distinctives [13] :

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.

3.2.1 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 [14]

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.

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

3.3 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 modèles. 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 [15].

3.3.1 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 [26]

3.3.2 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 [26] :

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.

3.4 Principe de transformation de graphes

Le processus de transformation de graphes consiste en l'application itérative d'une règle à un graphe. Chaque application de règle remplace une partie du graphe par une autre suivant la définition de la règle. La mécanique de la transformation de graphe fonctionne de la façon suivante : sélectionner une règle applicable de l'ensemble des règles ; appliquer cette règle au graphe d'entrée ; rechercher une autre règle applicable (réitération) jusqu'à ce qu'aucune règle ne puisse plus être appliquée. Cette opération est basée sur un ensemble de règles respectant une syntaxe particulière, appelé modèle de grammaire de graphe.

Un modèle de grammaire de graphe est une généralisation des grammaires de Chomsky. C'est une composition de règles où chaque règle possède deux parties : une partie gauche (LHS : Left Hand Side) et une partie droite (RHS : Right Hand Side). La partie gauche LHS correspond au graphe d'entrée (appelé aussi Host Graph). La règle compare à chaque fois qu'elle est invoquée son LHS avec le graphe sur lequel nous appliquons la transformation. La règle remplace l'équivalent de son LHS dans le graphe à transformer par sa partie droite RHS. Le RHS décrit la modification du host graph [18, 17, 16]

Le principe de déroulement de la grammaire dans le processus de transformation de graphes montre dans la figure

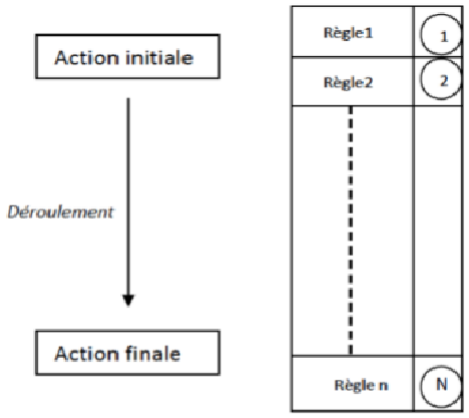


FIGURE 3.1 – Déroulement de grammaire

3.5 Grammaire de graphe

Une grammaire de graphe est généralement définie par un triplet [16] :

GG = (P, S, T).

Où :

- P : ensemble de règles.
- S : un graphe initial.
- T : ensemble de symboles.

3.6 Principe de déroulement de règles

Une règle de transformation de graphe est définie par :

R= (LHS, RHS, K, glue, emb, cond).

- LHS graphe de partie gauche.
- RHS graphe de partie droite.
- Un sous graphe K de LHS.
- Une occurrence glue de K dans RHS qui relie le sous graphe avec le graphe de partie droite.
- Une relation d'enfoncement emb qui relie les sommets du graphe de la partie gauche et ceux du graphe de la partie droite.
- Un ensemble cond qui indique les conditions d'application de la règle.

L'application d'une règle R= (LHS, RHS, K, glue, emb, cond) à un graphe G produit en résultat un graphe H suivant les cinq étapes suivantes :

1. Choisir une occurrence du graphe de partie gauche LHS dans G.

2. Vérifier les conditions d'application d'après cond.

3. Retirer l'occurrence de LHS (jusqu'à K) de G ainsi que les arcs pendillés (tous les arcs ayant perdu leurs sources et/ou leurs destinations). Ce qui fournit le graphe de contexte D de LHS qui a laissé une occurrence de K.

4. Coller le graphe de contexte D et le graphe de partie droite RHS suivant l'occurrence de K dans k = 1, .., D et dans RHS, c'est la construction de l'union de disjonction de D et RHS, et pour chaque point dans K, identifier le point correspondant dans D avec le point correspondant dans RHS.

5. Enfoncer le graphe de partie droite dans le graphe de contexte de LHS suivant la relation d'enfoncement emb : pour chaque arc incident retiré avec un sommet v dans D et avec un sommet v' dans l'occurrence de LHS dans G, et pour chaque sommet v" dans RHS, un nouvel arc incident est établi (même étiquette) avec l'image de v et le sommet v" à condition que (v', v") appartient à emb.

La figure 3.2 représente application dune règle de transformation :

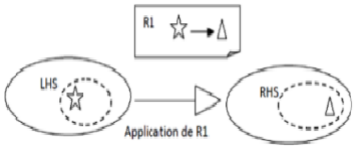


FIGURE 3.2 – Application de règle de transformation

3.7 Système de transformation de graphes

On définit un système de transformation de graphes comme un système de réécriture de graphe qui applique les règles de la grammaire de graphe sur son graphe initial de façon itérative jusqu'à ce que plus aucune règle ne soit plus applicable [17]

La figure 3.3 représente le principe du système de réécriture de graphes.

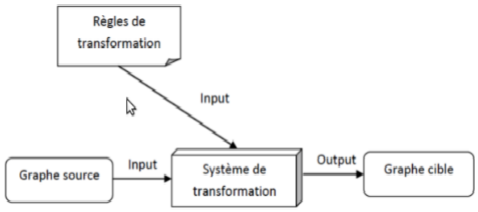


FIGURE 3.3 – Système de transformation de graphes

3.8 Outils de transformations de graphes

Plusieurs outils ont été développés pour la transformation de graphes, nous citons : ATOM3, AGG, PROGRES, FUJABA, plugins d'Eclipse, etc. Elles permettent toutes la génération d'un graphe orienté étiqueté cible à partir d'un autre graphe source en utilisant une grammaire bien définie.

• **AToM3 :** AToM3 (A Tool for Multi-formalism and Meta-Modelling) est un outil de transformation de modèles écrit en Python. Cet outil sert pour la modélisation, la méta-modélisation et la transformation de modèles en utilisant les concepts des grammaires de graphes. La spécification de formalisme présente des règles qui construisent les modèles. Ces modèles sont représentés dans un éditeur graphique [20]

ATo M3 possède donc une couche de méta-modélisation qui lui permet une modélisation graphique des différents formalismes. À partir de la méta-spécification (établie dans le modèle entité-association), AToM3 génère un outil pour la manipulation des différents modèles décrits dans le formalisme spécifié [19]

• **AGG :** AGG (Attributed Graph Grammar system) est un outil réalisé à l'Université Technique de Berlin (Technische Universität Berlin) et développé depuis 1997.C'est l'un des outils de transformation de graphes les plus aboutis et les plus cités dans la littérature [23]

AGG est écrit en langage de programmation Java. Il offre une interface graphique permettant de construire les graphes et les règles de transformation, et de réaliser des simulations pas à pas et quelques vérifications élémentaires. Les graphes considérés sont orientés et possèdent des noeuds et des arcs étiquetés par des objets Java.

• **PROGRES :** PROGRES (Programmed Graph Rewriting Systems) figure parmi les premiers outils à avoir permis les transformations de graphes. Il a été développé en Allemagne à l'université d'Aachen. Cet outil repose sur l'approche orientée logique des grammaires de graphes. Les règles sont décrites de manière visuelle par une partie gauche et une partie droite [22]

• **FUJABA :** FUJABA, (From UML to Java and Back Again) utilise UML comme langage de modélisation visuel, et a pour but de fournir un environnement de génération de code Java et de rétro-conception. FUJABA s'est développé et est devenu actuellement une base pour plusieurs activités de recherche, notamment dans le domaine des applications distribuées, les systèmes de bases de données ainsi que dans le domaine de la modélisation et de la simulation des systèmes mécaniques et électriques [21]

• **GReAT :** GReAT (Graph Rewriting and Transformation) utilise principalement une notation graphique pour définir les règles de transformation. Ces transformations sont unidirectionnelles de plusieurs modèles sources vers plusieurs modèles cibles. Ces modèles sont conformes à des méta-modèles spécifiés, et peuvent être créés avec l'outil de méta-modélisation GME.

Cependant, certaines parties sont spécifiées textuellement comme les expressions d'initialisation des attributs et les conditions d'application [24]

3.9 AToM3

3.9.1 Définition

AToM3 (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 2 en collaboration avec le professeur Juan de Lara de l'université Autónoma de Madrid (UAM), Espagne [19]

AToM3 est développé pour satisfaire deux fonctionnalités principales qui sont

1. La méta-modélisation.

2. La transformation des modèles.

Les formalismes et les modèles dans AToM3 sont décrits graphiquement. À partir d'une méta-spécification (exemple : dans le formalisme Entité-relation) d'un formalisme, AToM3 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 [26]

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. AToM3 traite les modèles de la même manière dans n'importe quel méta-niveau. L'idée principale d'AToM3 est : "tout est un modèle" [25]

Certains méta-modèles sont disponibles avec AToM3 :

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

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

La figure 3.4 illustre l'interface d'AToM3 avec le formalisme de diagramme de classe chargé.

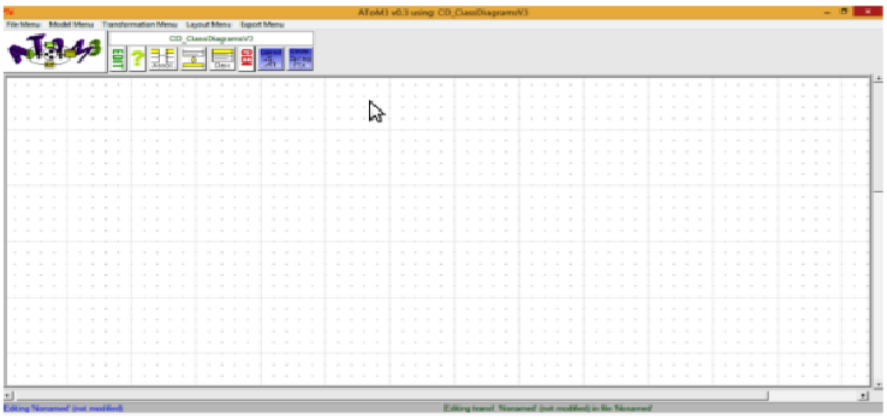


FIGURE 3.4 – Interface d'AToM3

3.9.2 Formalisme Diagrammes de Classes dans AToM3

Dans AToM3 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.

Contraintes

Les contraintes peuvent être spécifiées comme des expressions OCL (Object Constraint Language) ou Python [25]. Elles peuvent êtres locales associées à une entité, ou globales. Elles ont les propriétés suivantes :

- Un nom de contrainte
- Un évènement déclencheur : il peut être soit

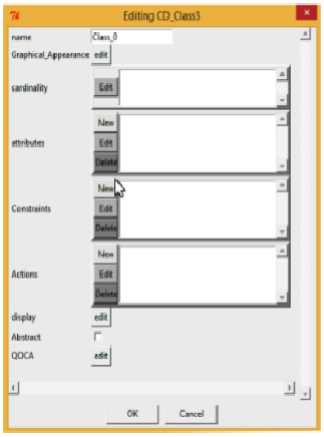


FIGURE 3.5 – Éditeur des propriétés

1. sémantique tel que la sauvegarde d'un modèle,...etc.
2. graphique ou structurel, tel que le déplacement ou la sélection d'une entité.

- L'évaluation soit :

1. avant l'évènement (pré-condition) ou
2. après (post-condition)

- Le code (soit en OCL ou Python).

Action

Une action est similaire à une contrainte sauf qu'elle a d'autres effets et elle est un code en Pyton seulement [27]

Elles ont les propriétés suivantes [25] :

- Un nom d'action.
- Un évènement déclencheur : Il peut être soit

1. Sémantique tel que la sauvegarde d'un modèle, etc.
2. Graphique ou structurel, tel que le déplacement ou la sélection d'une entité.

- L'exécution soit :

1. Avant l'évènement (pré-condition) ou
2. Après (pots-condition)

- Le code (soit en OCL ou Python).

L'éditeur des actions est similaire à l'éditeur des contraintes illustré par la figure 3.6 .

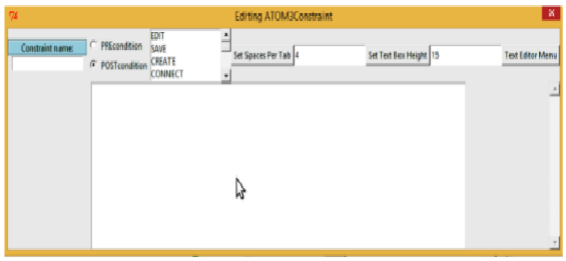


FIGURE 3.6 – Éditeur de contraintes

Attributs

Les entités (classes et relation d'association) qui doivent apparaitre 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 [25]

Deux types d'attributs existent dans AToM3 :

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 AToM3 :

- Type régulier tel que String, Integer, Float, Boolean. . . etc.

• Type générateur qui permet de générer des attributs, des contraintes, des attributs graphiques.

La figure 3.7 illustre l'éditeur des attributs.

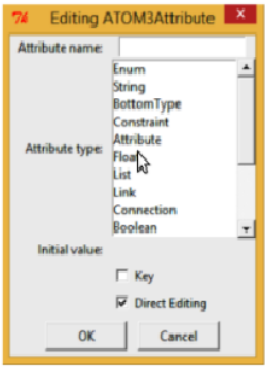


FIGURE 3.7 – Éditeur des attributs.

3.9.3 Transformation de Graphes

Dans AToM3 la grammaire est un modèle caractérisé par

• Une action initiale.

• Une action finale.

• L'ensemble des règles.

L'éditeur de grammaire (figure 3.8) permet l'édition, la génération et l'exécution de la grammaire.

FIGURE 3.8 – Éditeur de 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.

- 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 3.9) 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 3.6.

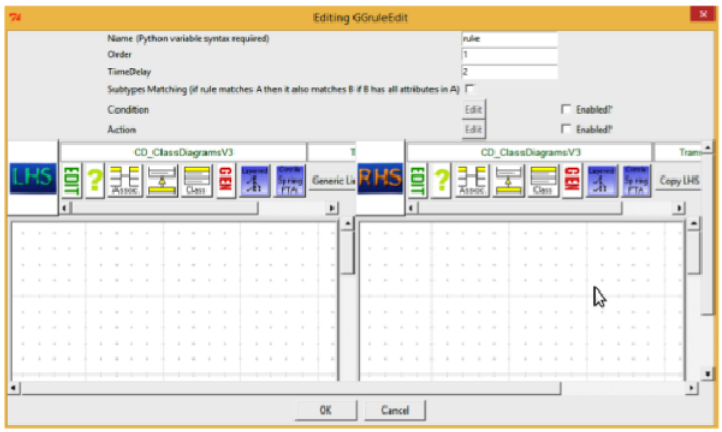


FIGURE 3.9 – Éditeur de règle

3.10 Présentation de l'Approche

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

1. Construction de méta-modèle des diagrammes d'activités SysML.

2. Construction de méta-modèle des réseaux de Petri.

3. Définition des règles de la transformation.

3.10.1 Méta-Modèle des Diagrammes d'activités SysML

Le diagramme d'activité est composé de :

- **Nœuds d'Object**

- **Action**

- **Nœuds de Control**

Nous utilisons AToM3 pour construire le méta-modèle du diagramme d'activités.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'AToM3.

2. Concevoir le méta-modèle de diagramme d'activités comme il est expliqué précédemment.La figure illustre le méta-modèle construit.Il est composé de 14 classes et 05 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 et selon la définition des diagrammes d'activités mobiles. Nous donnerons la signification des classes les plus importantes dans le méta-modèle.

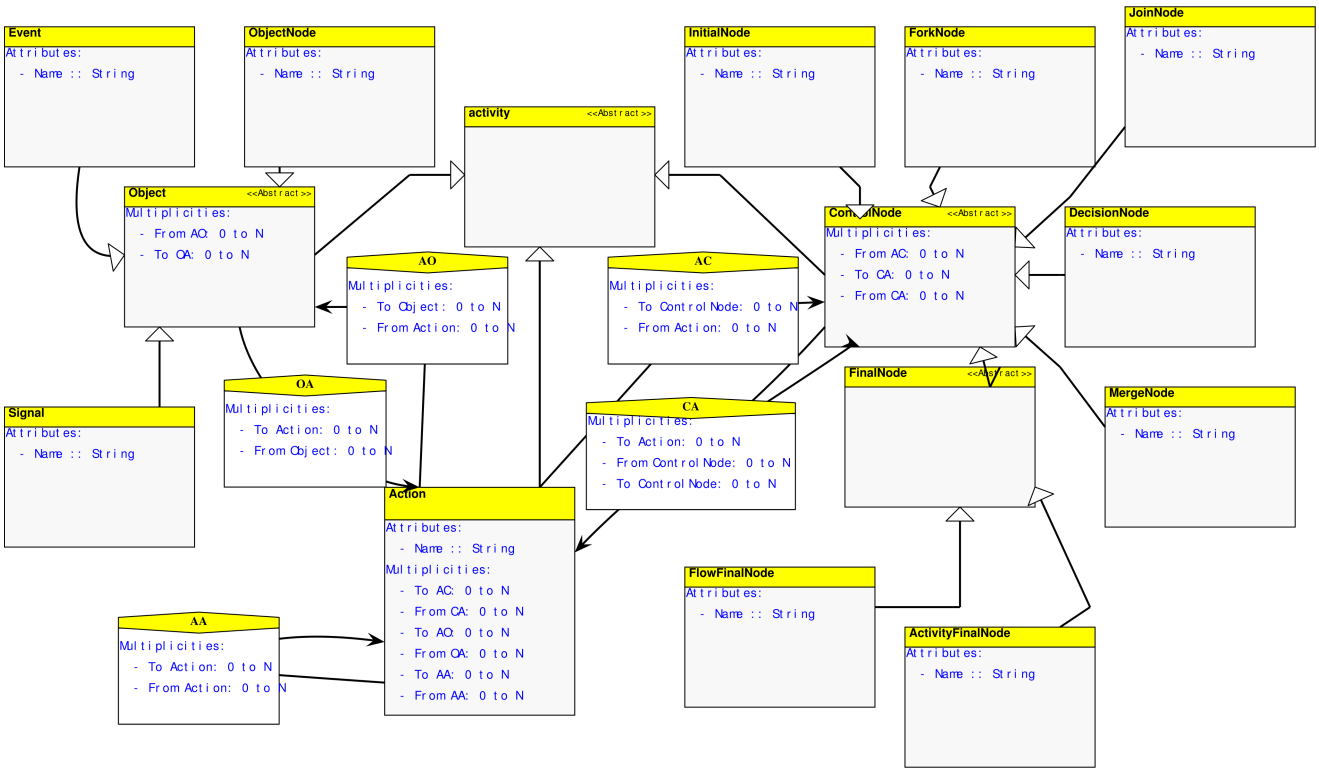


FIGURE 3.10 – Méta-modèle de diagramme d'activités SysML.

3. À partir de ce méta-modèle, AToM3 génère un outil de modélisation du diagramme d'état de transition. Cet outil offre un ensemble de boutons de manipulation de ce diagramme, comme le montre la figure 3.10 .

3.10.2 Méta-modèle de réseau de Petri

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

Notre méta-modèle est composé de 02 classes et 02 associations.

- **La classe Place :** est la classe qui représente les places dans le réseau de petri , elle a deux attributs “Nom” qui représente le nom de la place et “jeton” qui représente le nombre de jeton dans la place.

- **La classe Transition$_T$:** est la classe qui représente la transition dans le réseau de pétri, elle a un attribut “Nom” qui représente le nom de cette transition.

L’outil de modélisation généré à partir de méta-modèle de réseau de Petri est représenté par la figure 3.11 .

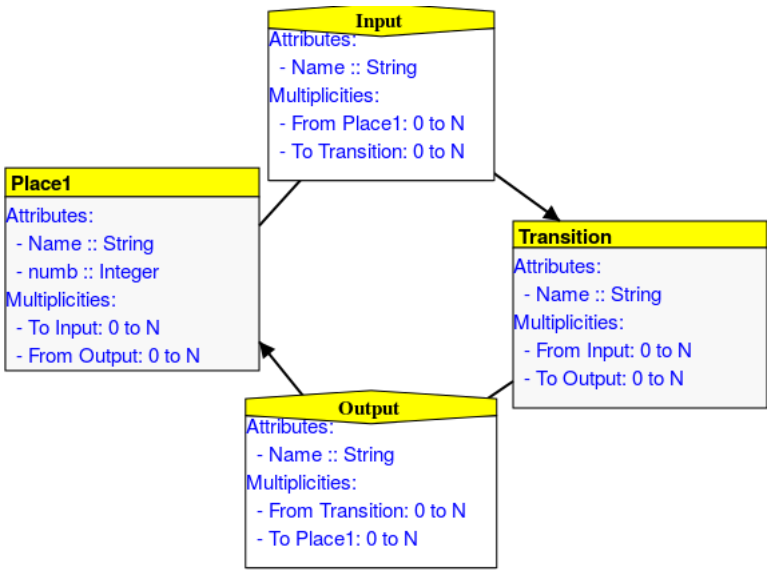


FIGURE 3.11 – Méta-modèle de PN.

3.10.3 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 différentes. 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 comme graphe source à transformer et le formalisme de réseau de Pétri comme graphe destinataire de transformation.

Cette grammaire est définie en utilisant l’outil AToM3 selon les étapes suivantes :

1. Charger les deux méta-modèles définis précédemment qui sont le méta-modèle du diagramme d’activités et le méta-modèle du réseau de Pétri.

2. Définir les règles de la grammaire.

3. Générer le fichier exécutable de la grammaire.

Notre grammaire graphes est composée de N règles. Chaque règle est caractérisée par un nom et une priorité d'exécution. Elles sont classés en 03 catégories :

1. Règles pour transformer les relations entre les Nœuds.

2. Règles pour transformer les Nœuds.

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 :

Nous citerons ici les règles les plus importantes :

1 • Ce sont certaines des règles de priorité 1 et sont d'ajouter une transition entre différents noeuds :

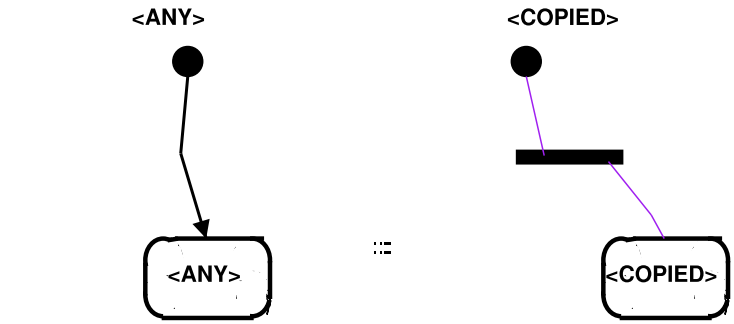


FIGURE 3.12 – inial2action

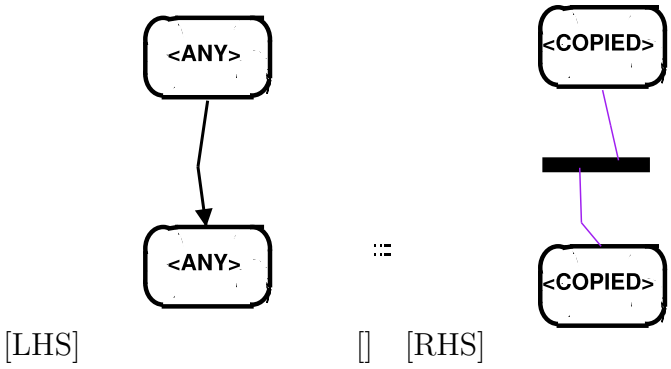


FIGURE 3.13 – regle action2action

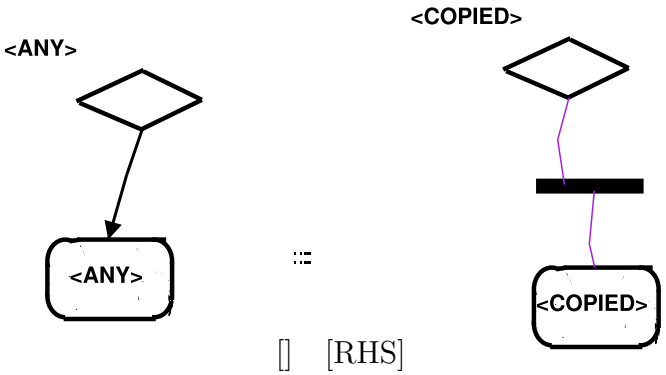


FIGURE 3.14 – regle decision2action

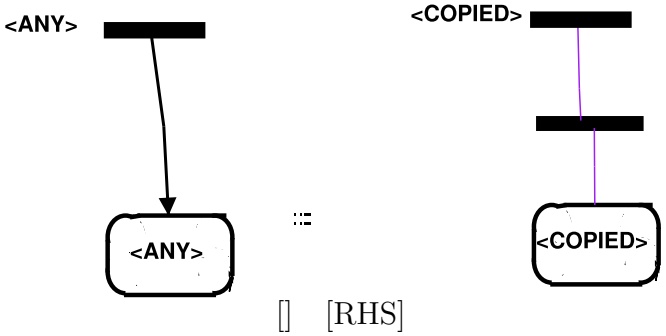


FIGURE 3.15 – regle fork2action

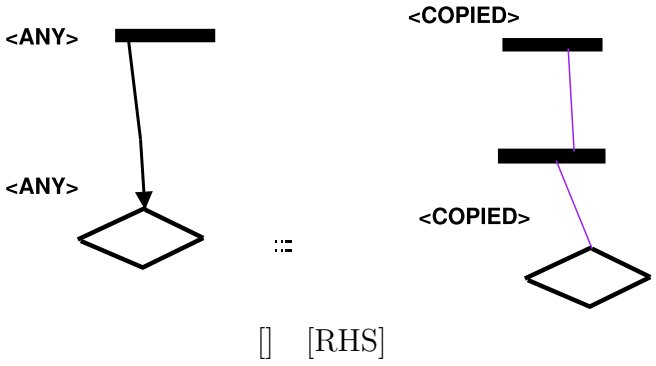
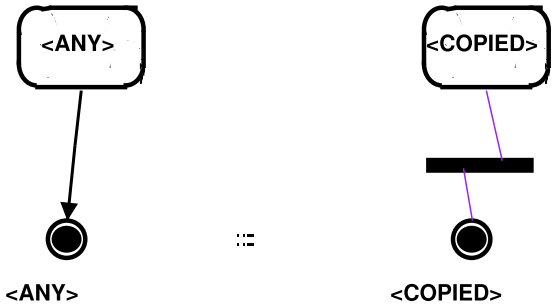


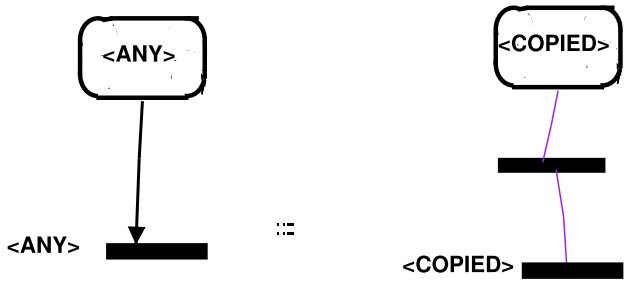
FIGURE 3.16 – regle join2decision

2 • Ces règles sont des cas spéciaux, car le noeud final ou join peut être lié à plusieurs arcs, mais représente une seule transition,et il suffit d'en appliquer un seul, donc ils diffèrent en priorité.



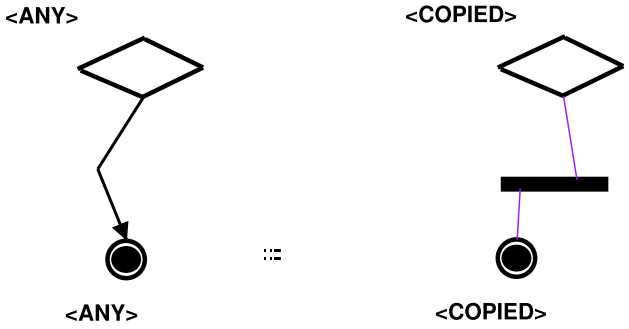
[LHS] [] [RHS]

FIGURE 3.17 – regle action2final(priorité 2)



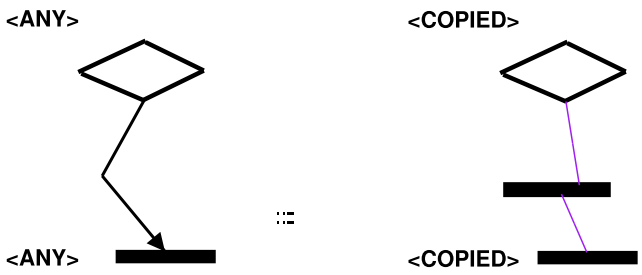
[LHS] [] [RHS]

FIGURE 3.18 – regle action2join(priorité 2)



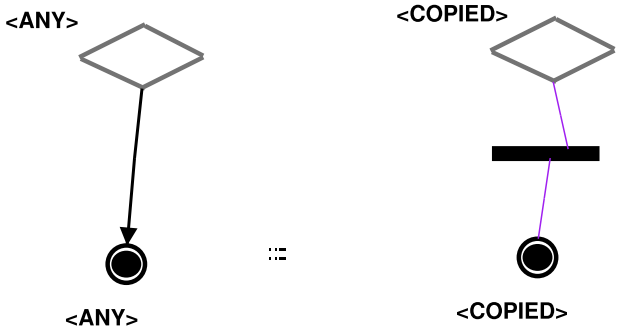
[LHS] [] [RHS]

FIGURE 3.19 – regle decision2final(priorité 3)



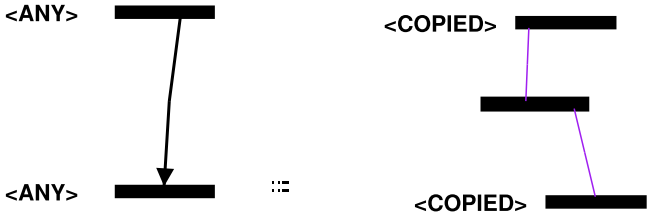
[LHS] [] [RHS]

FIGURE 3.20 – regle decision2join (priorité 3)



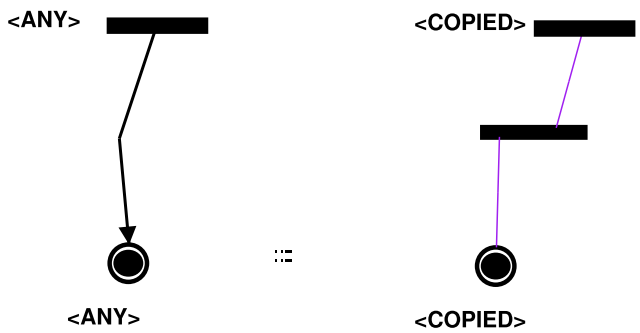
[LHS] [] [RHS]

FIGURE 3.21 – regle merge2final(priorité 4)



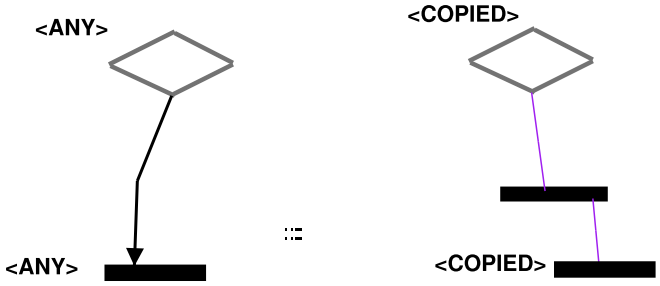
[LHS] [] [RHS]

FIGURE 3.22 – regle fork2join(priorité 4)



[LHS] [] [RHS]

FIGURE 3.23 – regle join2final(priorité 5)



[LHS] [] [RHS]

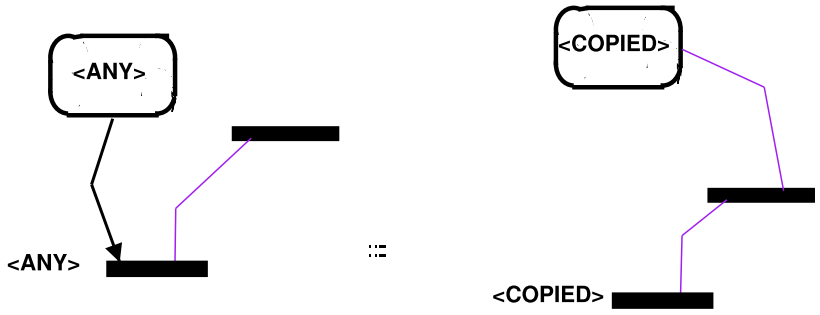
FIGURE 3.24 – regle merge2join(priorité 5)

[LHS] [] [RHS]

FIGURE 3.25 – regle join2join(priorité 6)

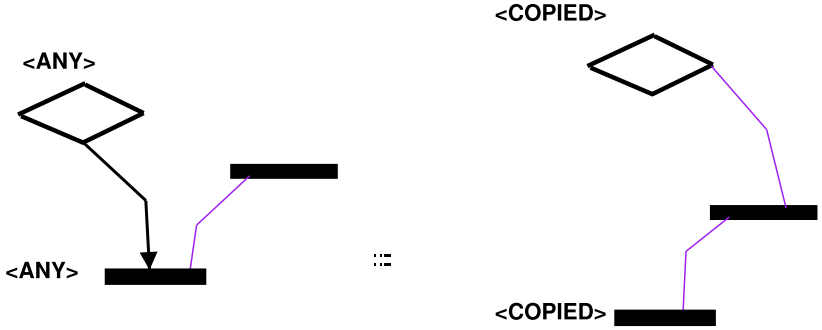
3 • Ces règles pour le précédent cas particulier.

Après avoir ajouté une transition, les autres nœuds doivent être connectés à cette transition.Par conséquent, les règles ont une priorité égale à 1, pour garantir qu'une seule transition est ajoutée .



[LHS] [] [RHS]

FIGURE 3.26 – regle actionTransition2join(priorité 3)



[LHS] [] [RHS]

FIGURE 3.27 – regle decisionWithTransition2join(priorité 3)

[LHS] [] [RHS]

FIGURE 3.28 – regle forkWithTransition2join(priorité 3)

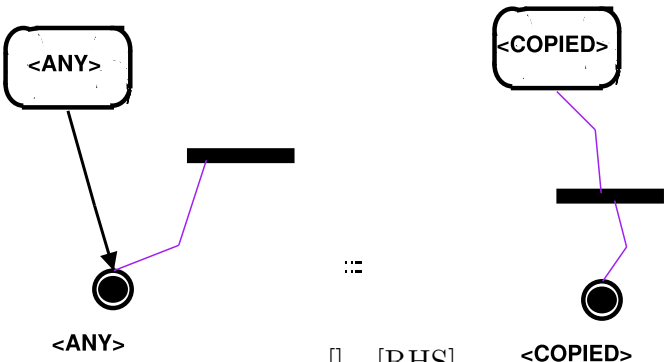


FIGURE 3.29 – regle ActionWithTransition2final(priorité 3)

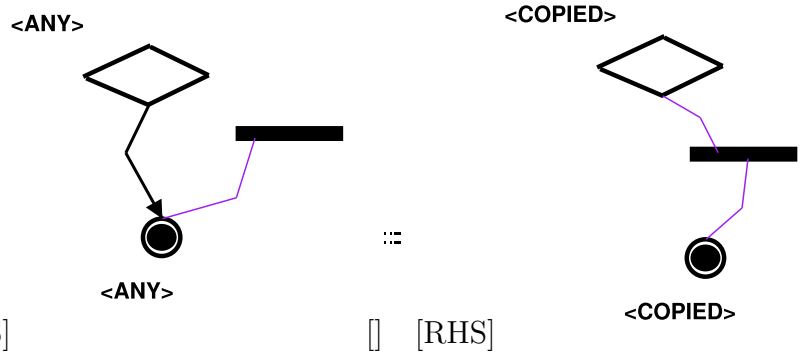


FIGURE 3.30 – regle DecisionWithTransition2final(priorité 3)

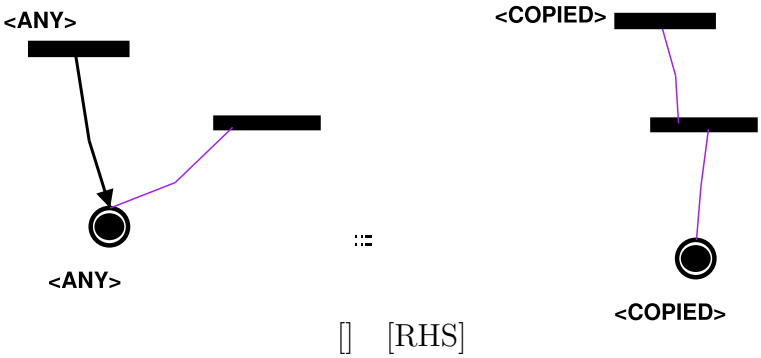


FIGURE 3.31 – regle JoinWithTransition2final(priorité 3)

4 • Ces règles pour la transformation.

nœud d'activité Pour chaque nœud d'activité, nous représentons une transition entre deux places, comme dans les règles suivantes :

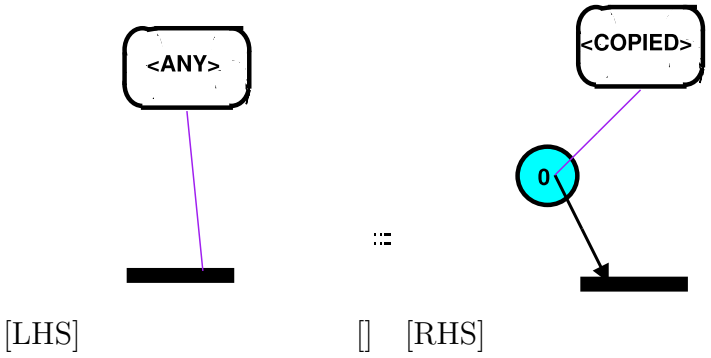


FIGURE 3.32 – regle Action2Transition(priorité 2)

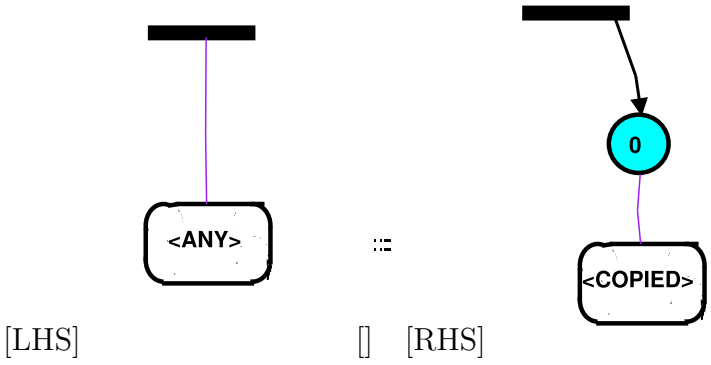


FIGURE 3.33 – regle transition2action (priorité 2)

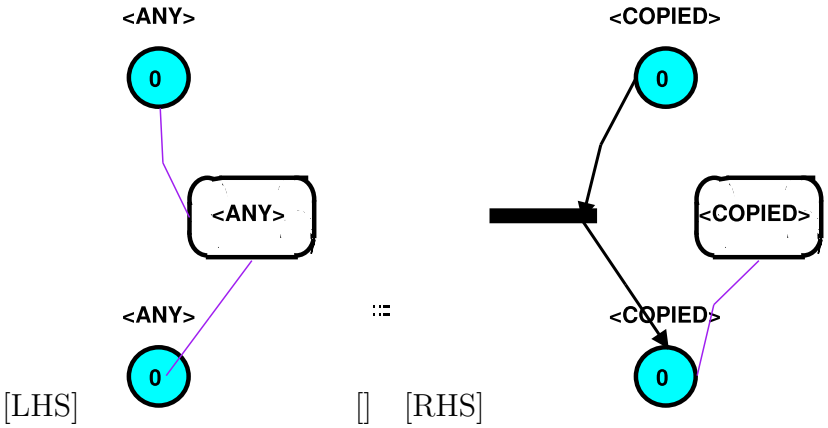


FIGURE 3.34 – regle place2action2place (priorité 1)

nœud de décision Pour transformer un nœud de décision, il suffit de le représenter uniquement en place comme suit :

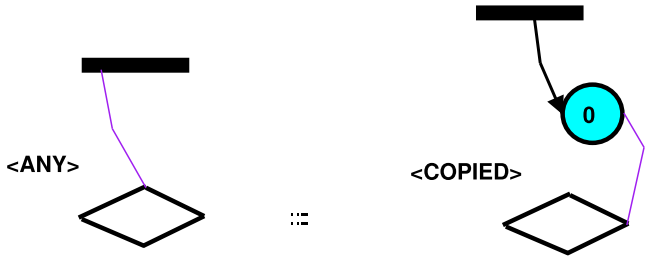


FIGURE 3.35 – regle transition2decision(priorité 9)

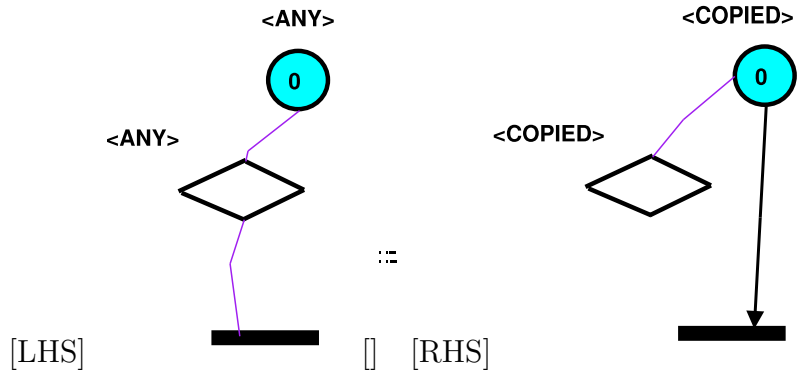


FIGURE 3.36 – regle place2decision2transition(priorité 10)

fork pour la représenter en ajoutant une place comme suit :

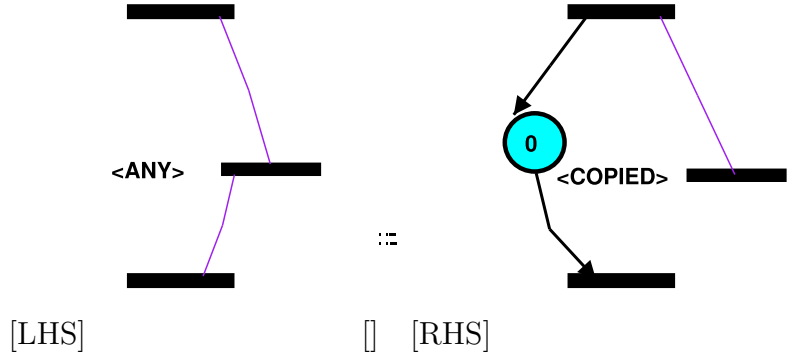


FIGURE 3.37 – regle fork2transition2fork (priorité 1)

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 SysML en utilisant cette approche.

3.10.4 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 des exemple un peu plus compliquer.

3.10.5 Approche de transformation (PN vers fichier XML)

Dans la section 3.10.5 nous avons présenté notre approche de transformation qui permet de transformer les diagrammes d'activités vers les réseaux de petri. Le but de cette transformation est la vérification et pour cela nous avons proposé une autre approche de transformation qui permet de transformer les réseaux de petri généralisé stochastique vers des fichiers XML pour vérifier par un outil de vérification choisis qui qui nous avons illustré dans cette section.

Grammaire de Graphes

Cette 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 le noeud dans le graphe qui a été déjà traité et aussi de créer la partie première de fichier XML.. La figure 3.38 représente le code de l'action initiale dans l'éditeur des contraintes.

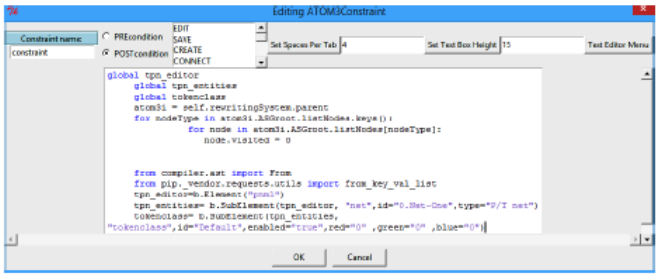


FIGURE 3.38 – Action initiale

- **Action initiale :** L'action finale permet de remplir le fichier .xml par les balises XML après avoir teterminé la transformation de tous les éléments du réseau de petri généralisé stochastique et le sauvegarder sur un fichier " page.xml". La figure3.39 représente le code de l'action finale dans l'éditeur des contraintes.

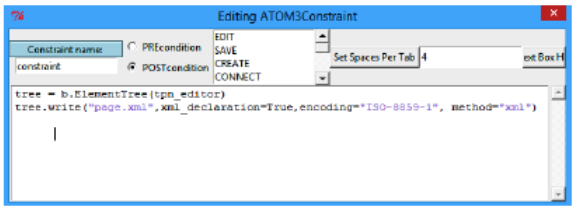


FIGURE 3.39 – Action finale

• **Ensemble de règles :** Notre grammaire graphes est composée de dix règles. Chaque règle est caractérisée par un nom et une priorité d'exécution. Elles sont classées en 02 catégories :

– Règles pour transformer les éléments du réseau de petri .

– Règles pour le nettoyage du résultat de la transformation.

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

• Chaque place dans le PN est trransformée vers un code XML correspondant. • Chaque transition dans le PN est transformée vers un code XML correspondant. • Chaque arc entre place et transition est transformé vers un code XML correspondant.

Nous citerons ici les règles les plus importantes :

1. Place$_{TO_XMLCode}$(*priorité*1) : Cette règle (figure 3.40) à la priorité égale à "1", c à d, elle est la première règle appliquée dans le PN. Elle permet de transformer la place vers un code XML.

FIGURE 3.40 –

(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 3.41 .

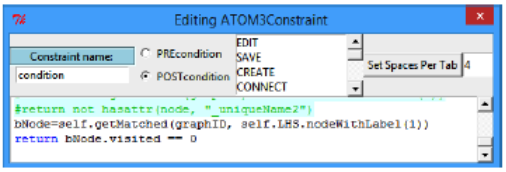


FIGURE 3.41 – Condition d'application de la règle 1

(b) Action : Une fois que la règle est appliquée, son action permet de crée un code XML de la place et marquer l'objet de la partie gauche (traité). Le code de cette action est exprimé en python comme l'illustre la figure 3.42

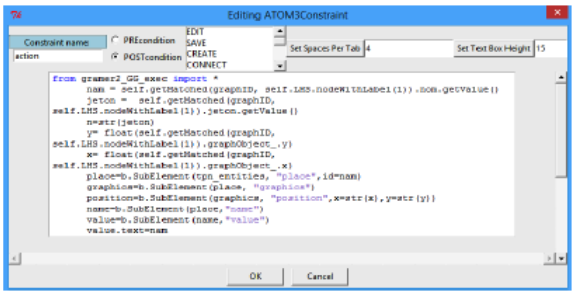


FIGURE 3.42 – Action de la règle 1

Transition$_{TO_XMLCode}$(*priorité*2) : Cette règle (figure 3.43) à la priorité égale à "2". Elle permet de transformer la transition vers un code XML.

:=

[LHS]^1 [] [RHS]^1

FIGURE 3.43 –

(a) condition : On utilise l'éditeur de condition, comme on a fait avec la règles précédente pour le spécifier. Le code de cette condition est exprimé en python comme le présente la figure 3.44.

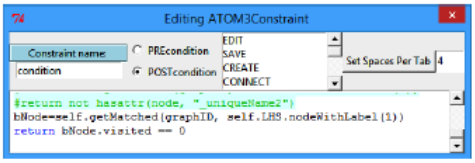


FIGURE 3.44 – Condition d'application de la règle 2

(b) action : L'application de cette règle permet à l'action de crée un code XML pour la transition et marqué cette objet comme un objet traité. La figure 3.45 illustre le code écrit en python de cette action.

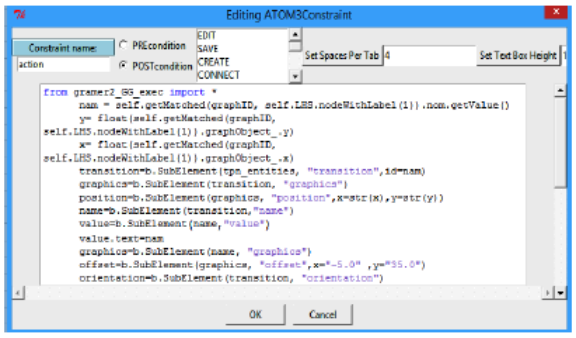


FIGURE 3.45 – Action de la règle 2

3. $\mathbf{Arc_TOX}ML_Code_1(priorité3)$: Cette règle permet de transformer l'arc qui relie une place avec une transition immédiate vers un code XML. La figure 3.49 représente cette règle.

FIGURE 3.46 –

(a) action : L'application de cette règle permet à l'action de crée un code XML pour l'arc qui relie la place avec la transition . La figure 3.50 illustre cette règle.

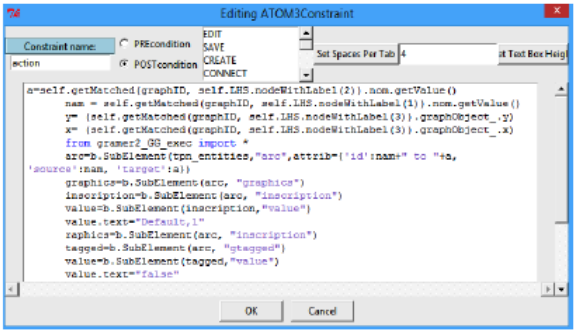


FIGURE 3.47 – Action de la règle 4

7. Eleminated Transition (priorité 5) : Cette règle nettoie le résultat de transformation est se spécialise dans l'élimination des transitions . La figure 3.54 illustre cette règle.

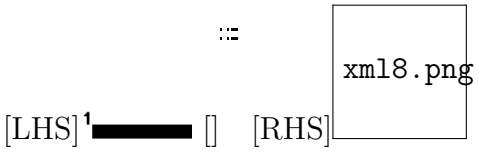


FIGURE 3.48 –

3.10.6 Conclusion

Nous avons vue dans ce chapitre les principes de transformations de graphes et ensuit nous avons présenté notre approche de transformations des diagrammes d'activités SysML vers les réseaux de petri et nous avons présenté aussi notre 2eme approche qui permet de transformer les réseaux de petri vers les fichiers XML par la suite nous avons vue l'outil pipe3 et vérifier un exemple qu'on a proposé.

Conclusion Générale

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

Nous avons introduit dans le premier chapitre les concepts de base de l'utilisation du langage SysML et nous avons aussi présenté quelque notion des diagrammes ce chapitre qui ont été terminé par une présentation des diagrammes d'activités SysML qui étudie la dynamique et le comportement d'un système.

Le deuxième chapitre a détaillé les réseaux de Petri. Les concepts de base et les principales propriétés des réseaux de Petri ordinaires ont été évoqués. Puis les réseaux de Petri ont été présentés. Ces derniers sont des réseaux de Petri permettant de donné une vue globale au système.

Dans le troisième chapitre, nous avons présenté notre approche pour la transformation des diagrammes d'activités SysML vers les réseaux de Petri . Nous avons vu que cette approche se divise en trois étapes : la première consiste à proposer un méta modèle des diagrammes d'activités SysML. Ce dernier est utilisé par l'outil AToM3 pour générer un outil de modélisation des diagrammes d'activités SysML. La deuxième étape consiste aussi à proposer un méta-modèle pour les réseaux de Petri . A partir de ce dernier, AToM3 génère un outil de modélisation des réseaux de Petri . La troisième étape consiste à définir une grammaire de graphes qui permet de transformer un graphe source décrit avec le formalisme de diagramme d'état transition vers un graphe de réseau de Petri malgré que le résultat de la transformation du diagramme d'activités SysML soit un réseau de Petri , 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. Pour cela nous avons proposé une autre approche de transformations des réseaux de petri vers des fichiers XML pour utiliser ces fichiers dans des outils de vérification et nous avons choisis l'outil pipe3 pour faire l'analyse des réseaux de petri.

Nous proposons, comme perspective à ce travail, la définition des règles pour transformer l'état composite dans les diagrammes d'état transition vers son équivalent dans le réseau de petri .

Bibliographie

[1] KOREN, Yoram. General RMS characteristics. Comparison with dedicated and flexible systems. In : Reconfigurable manufacturing systems and transformable factories. Springer Berlin Heidelberg, 2006. p. 27-45.

[2] V. Tran, V. Moraru, Réseau de Petri, Institut de la Francophonie pour l'Informatique Promotion 10 15 juillet 2005. .

[3] Pascal roques,SysML par l'exemple,Consultant senior A2 (Artal Innovation),février 2009,ISBN : 978-2-212-85006-2.

[4] T. Murata. "Petri nets : Properties, Analysis and Applications", Proceedings the IEEE,Vol. 77, No 4, April 1989. 1] E. Best, R. Devillers and J. G. Hall. The Box Nets 92, LNCS Vol. 609, 2169. Springer, 1992. Calculus : a New Causal Algebra with Multi-Label Communication. Advances in Petri..

[5] Diaz, Michel (2001) Les Réseaux de Petri – Modèles fondamentau x, Hermes Science Publications : Paris.

[6] Jon Holt and Simon Perry,SysML for Systems Engineering 2nd Edition : A model-based approach,IET PROFESSIONAL APPLICATIONS OF COMPUTING SERIES 10,Published by The Institution of Engineering and Technology, London, United Kingdom,2013,ISBN 978-1-84919-652-9.

[7] S. Roch and P.H Starke P.H, "Integrated Net Analyzer", User manual, 2002.

[8] E. Best and B. Grahlmann. PEP : Documentation and User Guide version 1.4 Universitat Hildesheim institut für informatik Novembre 1995.

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

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

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

[12] 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.

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

[14] 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.

[15] 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.

[16] Marc Andries, Gregor Engels, Annegret Habel, Berthold Hoffmann, Hans-Jörg Kreowski, Sabine Kuske, Detlef Plump, Andy Schürr, and Gabriele Taentzer. Graph

transformation for specification and programming. Science of Computer programming, 34(1) :1–54, 1999.

[17] Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg. Handbook of Graph Grammars and Computing by Graph Transformation : Volume 2 : Applications, Languages and Tools. world Scientific, 1999.

[18] Gabor Karsai and Aditya Agrawal. Graph transformations in omg's model-driven architecture. In Applications of Graph Transformations with Industrial Relevance, pages 243–259. Springer, 2003.

[19] AToM3.Homepage :http ://atom3.cs.mcgill.ca/LastConsultation :Mai2016

[20] Juan de Lara and Hans Vangheluwe. Atom3 : A tool for multi-formalism modelling and meta-modelling. In IN PROC. FASE'02, SPRINGER LNCS 2306. Citeseer, 2002.

[21] Wilhelmshöher Allee. Tool integration at the meta-model level within the fujaba tool suite. In TIS 2003 Workshop on Tool Integration in System Development, 2003.

[22] Ulrike Ranger and Erhard Weinell. The graph rewriting language and environment progres. In Applications of Graph Transformations with Industrial Relevance, pages 575–576. Springer, 2007.

[23] Gabriele Taentzer. Agg : A graph transformation environment for modeling and validation of software. In Applications of Graph Transformations with Industrial Relevance, pages 446–453. Springer, 2003.

[24] Daniel Balasubramanian, Anantha Narayanan, Christopher van Buskirk, and Gabor Karsai. The graph rewriting and transformation language : Great. Electronic Communications of the EASST, 1, 2007.

[25] Juan De Lara, Hans Vangheluwe, and Manuel Alfonseca. Meta-modelling and graph grammars for multi-paradigm modelling in atom3. Software and Systems Modeling, 3(3) :194–209, 2004.

[26] Robert Valette. Les réseaux de petri. LAASCNRS Toulouse, Septembre, 2000.

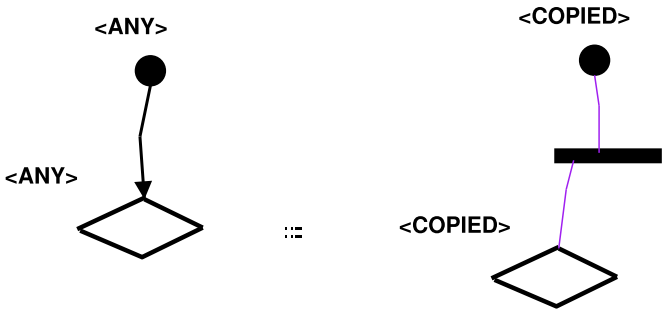
[27] J. de Lara and H. Vangheluwe, "AToM3 : A tool for multi-formalism and metamodelling," in FASE (R.-D. Kutsche and H. Weber, eds.), vol. 2306 of Lecture Notes in Computer Science, pp. 174–188, Springer, 2002.

[28] WESLATI Mohamed Karim , VERS L'INTÉGRATION DE PARAMÈTRES TEMPORELS STATIQUES ET DYNAMIQUES DANS LES TECHNIQUES DE VÉRIFICATION PAR ORDRE PARTIEL ,pages 14,DÉCEMBRE 2014 .

[29] Feng CHU , CONCEPTION DBS SYSTBMBS DE PRODUCTIONA L'AIDB DES RBSEAUX DE PETRI : VERIFICATION INCREMENTALB DBS PROPRIBTES QUALITATIVBS,pages 24-32, Avril 1995.

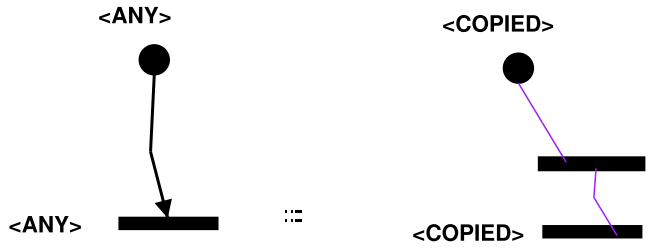
[30] BENMOUSSA FATIMA ZOHRA and HOUIDI FARIDA, Une approche basé sur la transformation du graphe sous ATom3 pour l'intégration des deux outils de réseau de petri , page 6–13, 2015.

Annexe



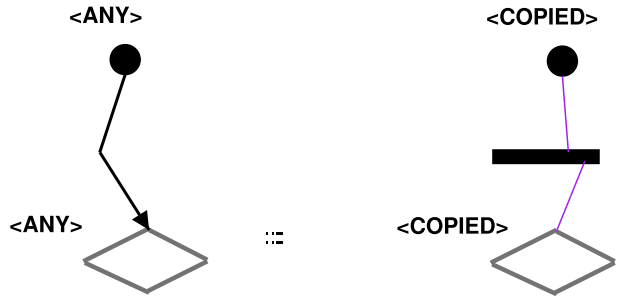
[LHS] [] [RHS]

FIGURE 3.49 – regle inial2decision (priorité 3)



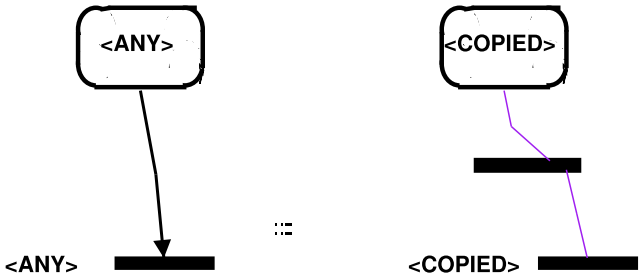
[LHS] [] [RHS]

FIGURE 3.50 – regle inial2fork (priorité 3)



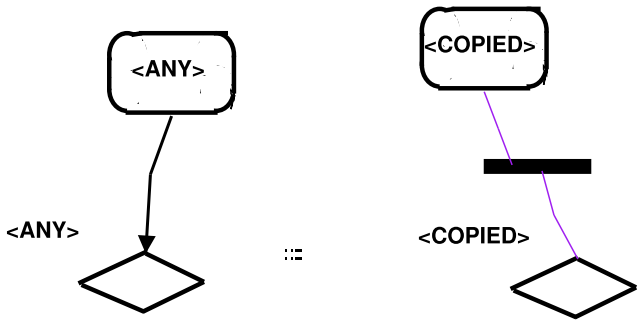
[LHS] [] [RHS]

FIGURE 3.51 – regle inial2merge (priorité 3)



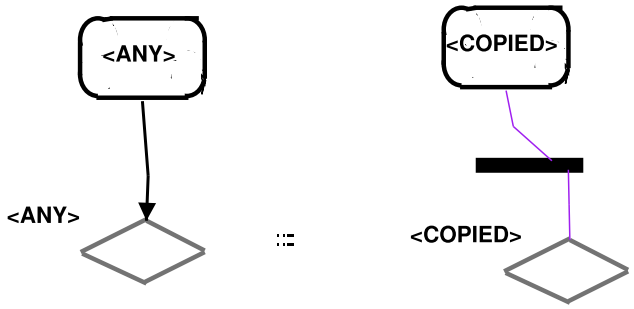
[LHS] [] [RHS]

FIGURE 3.52 – regle action2fork (priorité 3)



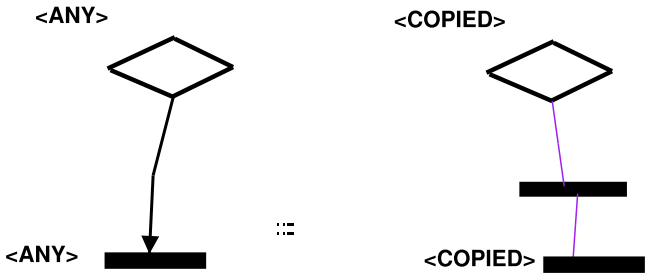
[LHS] [] [RHS]

FIGURE 3.53 – regle action2decision (priorité 3)



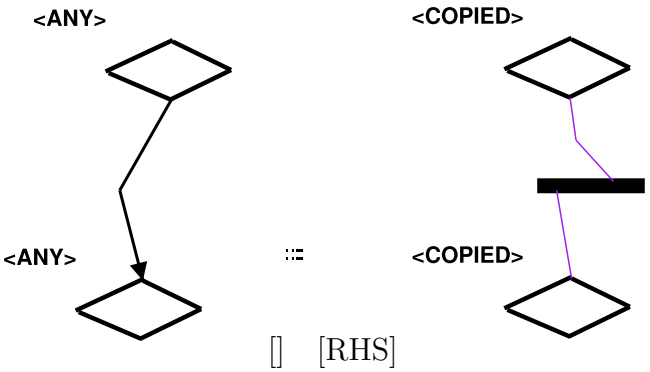
[LHS] [] [RHS]

FIGURE 3.54 – regle action2merge (priorité 3)



[LHS] [] [RHS]

FIGURE 3.55 – regle decision2fork (priorité 3)



[LHS] [] [RHS]

FIGURE 3.56 – regle decision2decision (priorité 3)

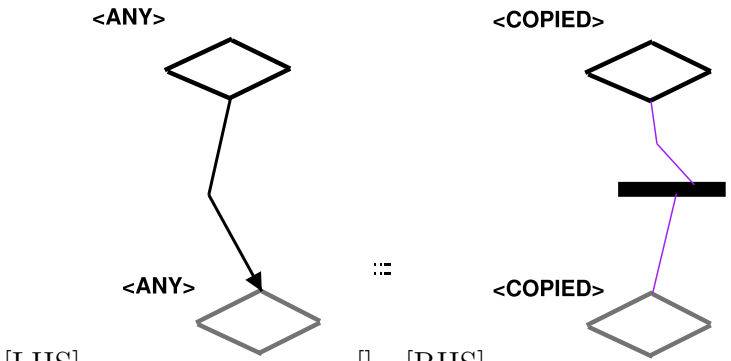


FIGURE 3.57 – regle decision2merge (priorité 3)

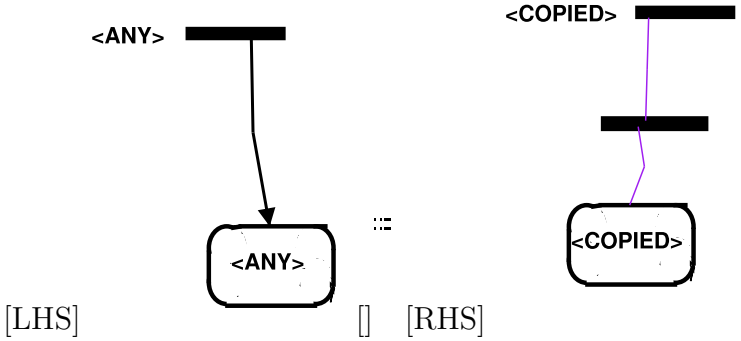


FIGURE 3.58 – regle fork2action (priorité 3)

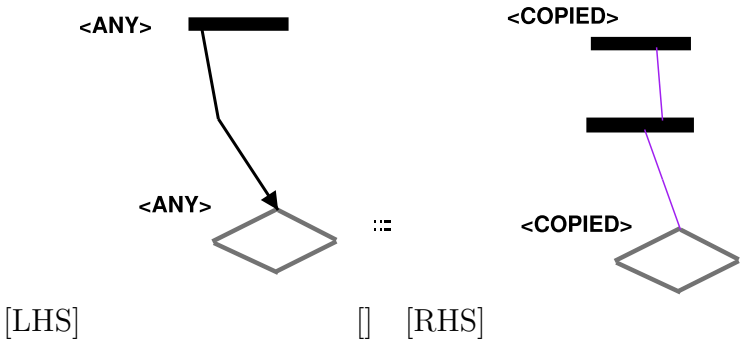


FIGURE 3.59 – regle fork2merge (priorité 3)

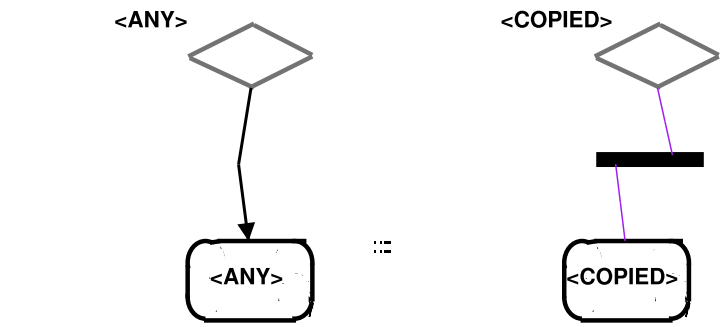


FIGURE 3.60 – regle merge2action (priorité 3)

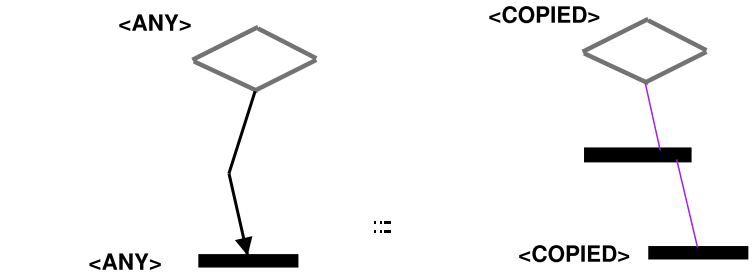


FIGURE 3.61 – regle merge2fork (priorité 3)

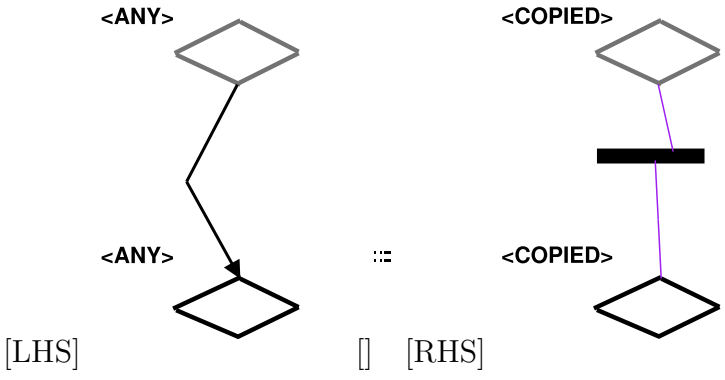
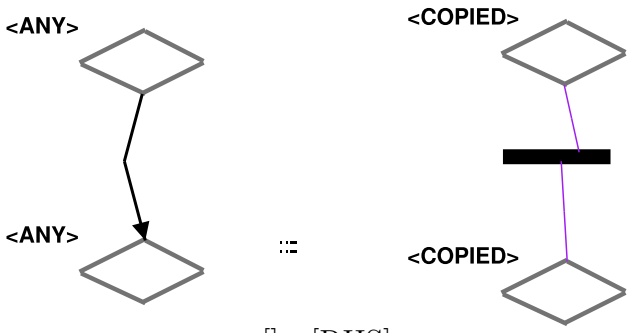
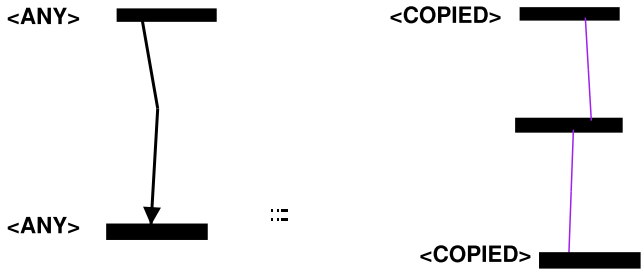


FIGURE 3.62 – regle merge2decision (priorité 3)



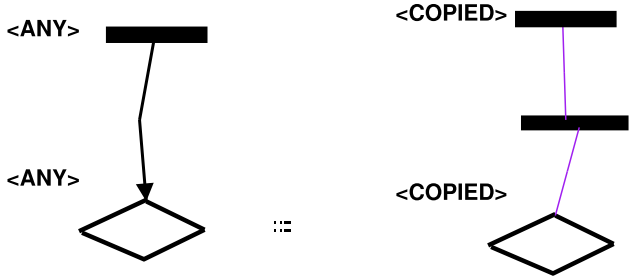
[LHS] [] [RHS]

FIGURE 3.63 – regle merge2merge (priorité 3)



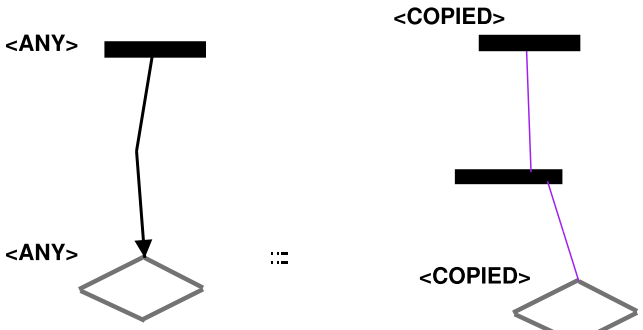
[LHS] [] [RHS]

FIGURE 3.64 – regle fork2fork (priorité 3)



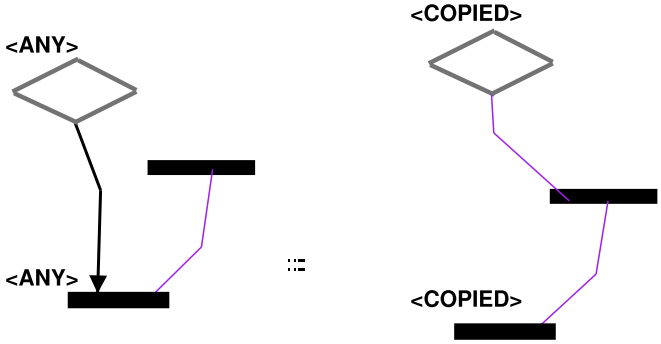
[LHS] [] [RHS]

FIGURE 3.65 – regle fork2merge (priorité 3)

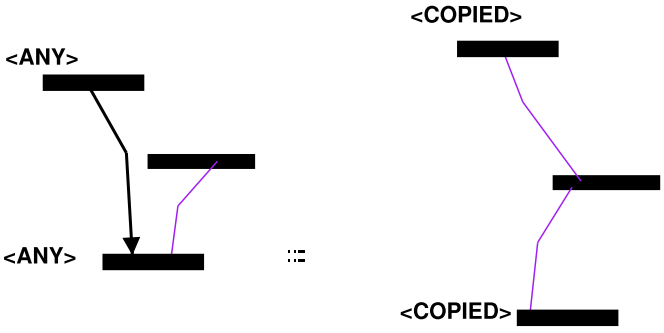


[LHS] [] [RHS]

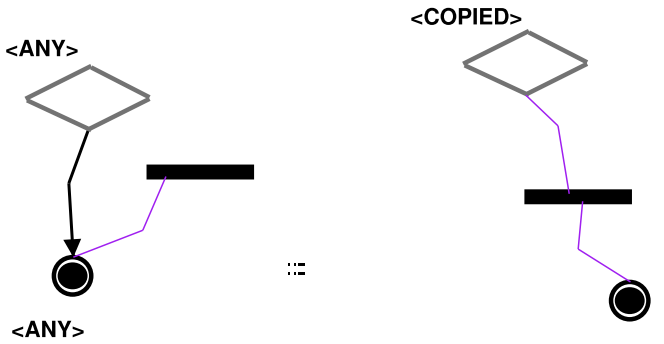
FIGURE 3.66 – regle join2merge (priorité 3)



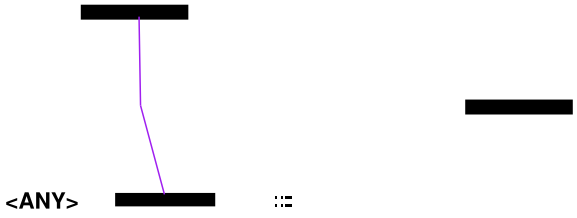
[LHS] [] [RHS]



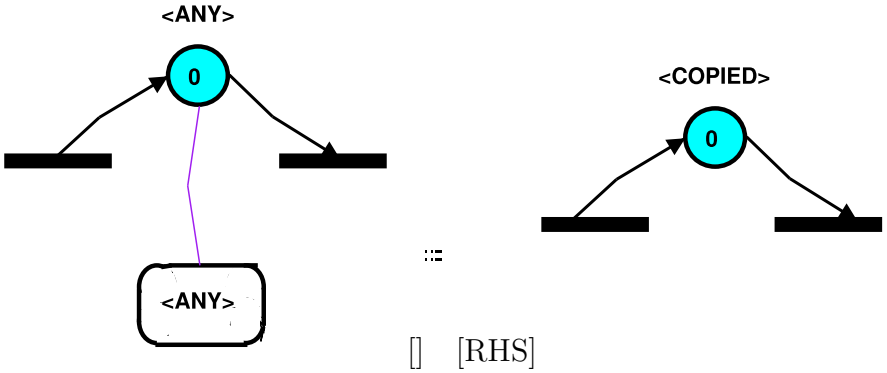
[LHS] [] [RHS]



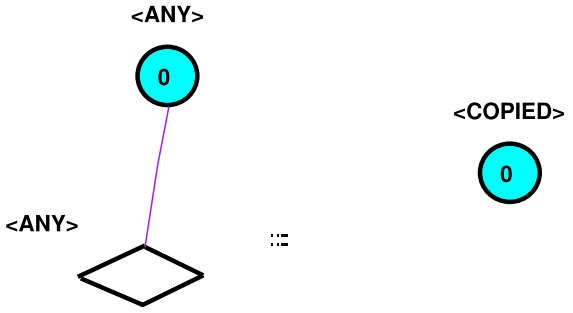
[LHS] [] [RHS]



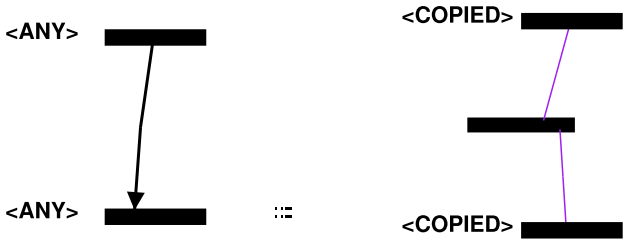
[LHS] [] [RHS]



[LHS] [] [RHS]

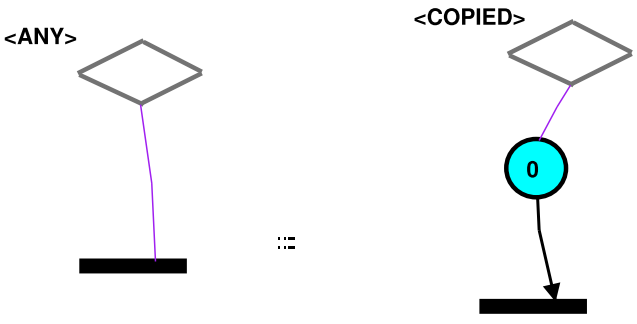


[LHS] [] [RHS]



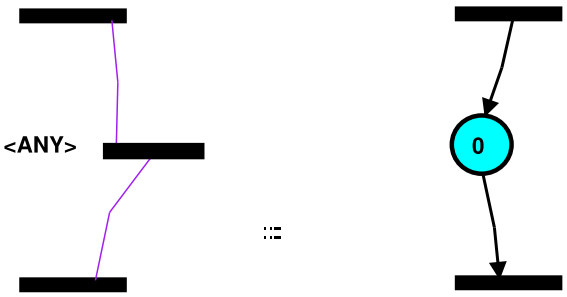
[LHS] [] [RHS]

FIGURE 3.73 – regle join2join (priorité 8)

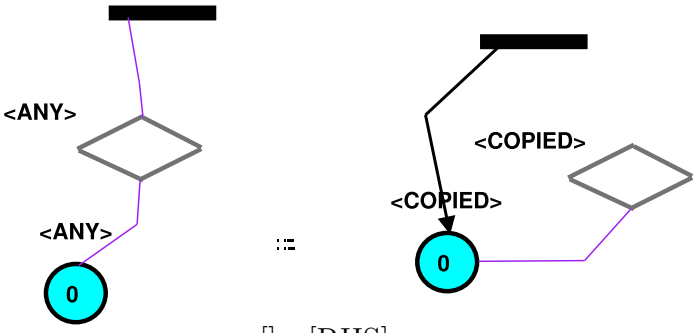


[LHS] [] [RHS]

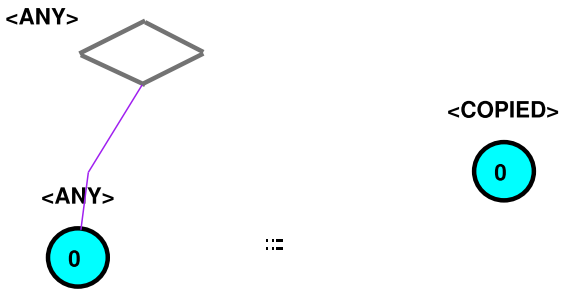
FIGURE 3.74 – regle merge2transition (priorité 9)



[LHS] [] [RHS]



[LHS] [] [RHS]



[LHS] [] [RHS]