الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

وزارة التعليم العالي و البحث العلمي

Ministère de L'Enseignement Supérieur et de la Recherche Scientifique

N°Ref :……

Centre Universitaire Abdelhafid BOUSSOUF- Mila

Institut des Sciences et de la Technologie

Département de Mathématiques et Informatique

Mémoire préparé en vue de l'obtention du diplôme de

Master

En : Informatique
Spécialité : Sciences et Technologies de l'Information et de la Communication (STIC)
Thème :

Transforming UML statchert diagrams into nets within nets formalism for verification purpose.

Préparé par :

- Laifa Silia
- Boukheche Ichrak

Soutenue devant le jury

Encadré par	**Dr. Hettab Abdelkamel**	**Grade : Maitre de conférence B**
Président	**Dr. Aouag Mouna**	**Grade : Maitre de conférence B**
Examinateur	**Dr. Djaaboub Salim**	**Grade : Maitre de conférence B**

Année Universitaire : 2022/2023

الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

وزارة التعليم العالي و البحث العلمي

Ministère de L'Enseignement Supérieur et de la Recherche Scientifique

N°Ref :……

Centre Universitaire Abdelhafid BOUSSOUF- Mila

Institut des Sciences et de la Technologie

Département de Mathématiques et Informatique

Mémoire préparé en vue de l'obtention du diplôme de

Master

En : Informatique
Spécialité : Sciences et Technologies de l'Information et de la Communication (STIC)

Thème :

Transformation des diagrammes d'états-transition UML en modèles des réseaux de Petri nets within nets pour des fins de vérification.

Préparé par :

- Laifa Silia
- Boukheche Ichrak

Soutenue devant le jury

Encadré par	**Dr. Hettab Abdelkamel**	**Grade : Maitre de conférence B**
Président	**Dr. Aouag Mouna**	**Grade : Maitre de conférence B**
Examinateur	**Dr. Djaaboub Salim**	**Grade : Maitre de conférence B**

Année Universitaire : 2022/2023

Dédicaces

louange à dieu seul
ce modeste travail est dédie spécialement :

À ma chère mère, à ma cher père,Qui n'ont jamais cessé,de formuler des prières à mon égard ,de me soutenir et de m'épauler pour que je puisse atteindre mes objectifs ,en témoignage de ma reconnaissance pour leur patience,leur amour.

À l'âme de mon grand-père, Ô Dieu, protège-le dans son paradis, lui qui m'a toujours encouragé et motivé, pour son soutien, son amour et ses précieux conseils.

À mon frère Moustapha, à Ma chère sœur Abir et son mari Redouane, Pour ses soutiens moral et leurs conseils précieux tout au long de mes études.

À ma chère binôme Ichrak,Pour sa entente et sa sympathie.

À mon cher neveu Yahia , À mes amis, mes enseignants et pour ceux qui m'ont donné de l'aide un jour.

À toute ma famille. Pour finir, à tous ceux que j'aime et qui m'aiment.

Silia

louange à dieu seul
ce modeste travail est dédie spécialement :

À l'âme de mon père que dieu le garde dans son paradis ,à ma chère maman, ma raison de vivre, en témoignage de ma reconnaissance pour leur patience, leur amour ,leur affection ainsi que les sacrifices qu'ils on consentis pour mon éducation et ma formation .

Aux âmes de mon grand-père, et de ma grand-mère que dieu les garde dans son paradis ,qui m'ont toujours poussé et motivé,Pour leur soutiens, leur amour et leurs conseils précieux.

À mes frères Islam,Younes,et AbdRhamn à Ma chère sœur Misoun et son mari Sami,Pour ses soutiens moral et leurs conseils précieux tout au long de mes études.

À mes chères amies Sana et Yasmin, Pour leurs aides et supports dans les moments difficiles.

À ma chère binôme Silia,Pour sa entente et sa sympathie.

À mes chers neveu et nièce Iyad et Amina.

À mes amis, mes enseignants et pour ceux qui m'ont donné de l'aide un jour.

À toute ma famille. Pour finir, à tous ceux que j'aime et qui m'aiment.

Ichrak

Remerciements

Je tiens tout d'abord à exprimer ma profonde gratitude envers toutes les personnes qui m'ont soutenu tout au long de la réalisation de ce mémoire. Votre encouragement, vos conseils et votre soutien constant ont été d'une valeur inestimable pour moi.

Je tiens à remercier chaleureusement mon directeur de mémoire, Dr.Hettab, pour ses précieuses orientations, sa disponibilité et sa patience tout au long de ce processus. Vos idées éclairées et vos retours constructifs m'ont permis d'enrichir mes réflexions et d'améliorer la qualité de mon travail.

Mes remerciements vont également à Dr.Hettab, dont les discussions stimulantes ont contribué à élargir mon horizon de compréhension et à approfondir mes analyses.

Un grand merci à tous les participants de mon étude, dont la contribution a été essentielle pour recueillir des données significatives. Votre temps et vos insights ont été d'une importance cruciale pour les conclusions de ce mémoire.

Je tiens également à exprimer toute ma grande gratitude aux membres de jury :

Aouag Mouna.
Salim Djaaboub.

d'avoir accepté de juger ce travail.

Je tiens à remercier tous mes collègues et amis qui m'ont soutenue toutes ses années.

Enfin, je tiens à adresser mes remerciements à l'ensemble du corps professoral et du personnel de l'universit e Abdelhafid Boussouf-Mila, ainsi qu'à toutes les personnes qui ont contribué de près ou de loin à mon parcours académique.

La réalisation de ce mémoire n'aurait pas été possible sans l'appui de chacun d'entre vous. Vos contributions ont enrichi mon expérience de recherche et m'ont permis d'atteindre ce point de culmination dans mon parcours académique. Merci du fond du cœur.

ملخص :

تتناول هذه الأطروحة موضوع تحويل مخططات انتقال حالة UML إلىPetri Nets Within Nets ، لأغراض التحقق. الهدف الرئيسي من هذا البحث هو اقتراح طريقة فعالة لتحويل مواصفات النظام الممثلة في شكل مخططات انتقال حالة UML إلى نماذج رسمية تعتمد على Petri Nets Within Nets. يوفر التحول إلى هذه الشكلية تمثيلاً رسميًا قويًا، مما يجعل من الممكن التقاط الديناميكيات والمنافسة الموجودة في الأنظمة المعقدة. إن استخدام Petri Nets Within Nets يسهل التحليل الرسمي والتحقق من خصائص النظام. في هذه الورقة، يتم تقديم منهجية مفصلة لتحويل مخططات انتقال الحالة إلى Petri Nets Within Nets. يتم استخدام أمثلة ملموسة لتوضيح هذا النهج، وبالتالي إظهار قدرته على الحفاظ على الخصائص الأساسية للنظام الأولي مع تقديم تمثيل رسمي وقابل للتحقق. تسلط النتائج التي تم الحصول عليها الضوء على فعالية وأهمية نهج التحول هذا، والذي يسمح للمهندسين بنمذجة الأنظمة المعقدة بشكل أكثر دقة وتسهيل التحليل الرسمي لخصائص النظام. تساهم هذه الأطروحة بشكل كبير في تقدم الهندسة المعتمدة على النماذج وتقدم مزايا مهمة لنمذجة الأنظمة المعقدة والتحقق منها، وبالتالي فتح آفاق جديدة لتصميم وتطوير الأنظمة الحيوية.

كلمات مفتاحية :

تحويل النماذج، مخططات حالات الانتقالUML ، Petrie Nets Within Nets، التحقق الرسمي، هندسة النماذج الموجهة، الأنظمة المعقدة.

Résumé :

Ce mémoire aborde le thème de la transformation des diagrammes d’´états-transitions UML en Réseaux de Petri nets within nets, `a des fins de vérification. L’objectif principal de cette recherche est de proposer une approche efficace pour convertir les spécifications de systèmes représentées sous forme de diagrammes d’états-transitions UML en Modèles formels basés sur les Réseaux de Petri nets within nets. La transformation vers ce formalisme offre une représentation formelle puissante, permettant de cap- turer la dynamique et la concurrence présentes dans les systèmes complexes. L’utilisation des Réseaux de Petri nets within nets facilite l’analyse formelle et la vérification des propriétés du système. Dans ce mémoire, une méthodologie détaillée pour la transformation des diagrammes d’´états-transitions en Réseaux de Petri nets within nets est présentée. Des exemples concrets sont utilisés pour illustrer cette approche, démontrant ainsi sa capacité `a préserver les propriétés essentielles du système initial tout en fournissant une représentation formelle et vérifiable. Les résultats obtenus mettent en évidence l’efficacité et la pertinence de cette approche de transformation, qui permet aux ingénieurs de modéliser des systèmes complexes de manière plus précise et de faciliter l’analyse formelle des propriétés du système. Ce mémoire contribue significativement `a l’avancement de l’Ingénierie Dirigée par les Modèles et présente des avantages importants pour la modélisation et la vérification des systèmes complexes, ouvrant ainsi de nouvelles perspectives pour la conception et le d´développement de systèmes critiques.

Mots-clés :

Transformation de Modèles , Diagrammes d’états-transitions UML, Réseaux de Petri nets within nets, Vérification formelle, Ingénierie Dirigée par les Modèles , Systèmes complexes.

Abstract

This memory addresses the topic of transforming UML Statechart diagrams into Petri nets within nets for verification purposes. The main objective of this research is to propose an efficient approach to convert system specifications represented as UML Statechart diagrams into formal models based on Petri nets within nets. The transformation to this formalism provides a powerful formal representation, allowing for capturing the dynamics and concurrency present in complex systems. The use of Petri nets within nets facilitates formal analysis and verification of system properties. In this thesis, a detailed methodology for transforming Statechart diagrams into Petri nets within nets is presented. Concrete examples are used to illustrate this approach, demonstrating its ability to preserve the essential properties of the initial system while providing a formal and verifiable representation. The results obtained highlight the efficiency and relevance of this transformation approach, enabling engineers to model complex systems more accurately and facilitate formal analysis of system properties. This thesis significantly contributes to the advancement of Model-Driven Engineering and presents significant benefits for modeling and verifying complex systems, opening new perspectives for the design and development of critical systems.
Modern software and systems are becoming increasingly intricate, requiring comprehensive and effective methods for modeling, analyzing, and verifying their dynamic behavior. UML state transition diagrams are widely used to represent the dynamic aspects of software and system designs, offering a visual and intuitive means of depicting state transitions and system behaviors. However, as systems grow in complexity, so does the challenge of effectively analyzing and verifying their behavior using traditional UML state diagrams.

This thesis explores a novel approach to address this challenge, focusing on the transformation of UML state transition diagrams into Petri nets within nets models. Petri nets within nets extend traditional Petri nets by providing hierarchical structures that facilitate the modeling of complex systems and behaviors. The transformation process involves capturing the intricacies of a UML state transition diagram and mapping them into a corresponding Petri nets within nets model, preserving the essential behavioral properties of the system.

Throughout this research, we delve into the theoretical foundations, methodologies, and practical applications of this transformation process. We discuss the advantages of using Petri nets within nets for system verification, emphasizing their ability to handle system complexity while enabling rigorous analysis and verification. Case studies and examples are presented to illustrate the effectiveness of this approach across various domains, showcasing its utility and impact.

Furthermore, this thesis explores the integration of formal verification techniques with the transformed models, highlighting how Petri nets within nets can be leveraged for verification purposes. We discuss verification challenges and solutions, demonstrating how the transformed models can be subjected to formal analysis and model checking to ensure system correctness and reliability.

The goal of this thesis is to provide a comprehensive understanding of the transformation of UML state transition diagrams into Petri nets within nets models for verification. By doing so, it aims to contribute to the advancement of methods for analyzing and verifying the dynamic behavior of complex software and systems. This research holds the potential to enhance the quality, reliability, and safety of modern software systems, making it a valuable asset in both industry and academia.

Keywords :

Model Transformation, UML Statechart Diagrams, Petri Nets within Nets, Formal Verification, Model-Driven Engineering, Complex Systems.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Contexte générale

Au sein de l'Ingénierie Dirigée par les Modèles, la modélisation multi-paradigme offre une approche novatrice permettant de combiner différents formalismes de modélisation pour représenter des aspects complexes des systèmes. Les diagrammes d'´états-transitions UML sont couramment utilisés pour décrire le comportement des systèmes, mais leur représentation peut parfois manquer de précision et de rigueur dans certains cas. Les nets whithin nets, quant à eux, offrent une représentation formelle permettant de capturer de manière plus explicite les aspects concurrents et parallèles des systèmes complexes.

Ce mémoire se penche sur l'étude de cette transformation des diagrammes d'état de transition vers les "nets within nets". Nous explorerons les principes fondamentaux de cette approche, ses avantages potentiels et les défis qu'elle peut résoudre. Nous examinerons également les outils, les techniques et les méthodologies associés à cette transformation. Enfin, nous présenterons des exemples concrets d'application de cette approche dans différents domaines, démontrant ainsi son utilité et sa pertinence dans des contextes réels.

Au fil des chapitres qui suivent, nous plongerons plus profondément dans les détails de cette transformation, en abordant les aspects théoriques et pratiques. Nous analyserons également les cas d'utilisation spécifiques où les "nets within nets" ont montré leur efficacité, ainsi que les résultats obtenus lors de leur mise en œuvre.

L'objectif de ce mémoire est de fournir une vision d'ensemble complète et approfondie de la transformation de diagrammes d'état de transition vers les "nets within nets", tout en mettant en évidence son importance dans la conception, la modélisation et la gestion de systèmes informatiques complexes. Nous espérons que cette étude contribuera à éclairer les chercheurs, les concepteurs de systèmes et les praticiens sur cette approche prometteuse et favorisera son adoption dans l'industrie et la recherche.

Cette introduction générale établit le contexte du mémoire en expliquant la pertinence du thème de recherche et en présentant les objectifs et la structure du mémoire. Elle suscite l'intérêt du lecteur en montrant l'importance de la transformation de diagrammes d'état de transition vers les "nets within nets" dans le domaine de la conception de systèmes informatiques complexes.

Problématique

Chaque jour, les systèmes informatiques et industriels deviennent de plus en plus complexes, rendant ainsi la compréhension et l'analyse de leur comportement de plus en plus difficiles. C'est pourquoi de nombreux modèles formels et semi-formels ont été d´enveloppés pour représenter et analyser ces systèmes. Parmi ces modèles, on trouve les "diagrammes d'états-transitions UML" et les "Réseaux de Petri (RdP)", largement utilisés pour représenter la structure dynamique et concurrentielle des systèmes.

Dans ce mémoire, nous nous concentrons sur la transformation efficace et automatisée des diagrammes d'´états-transitions UML en "Réseaux de Petri (RdP) nets within nets" dans l'environnement de modélisation AToMPM. Etant donné l'importance de ces deux modèles dans l'étude des systèmes complexes et interconnectés, leur transformation représente un défi majeur dans le domaine de la modélisation et de l'analyse.

La question centrale de cette problématique est comment peut-on effectuer la transformation des diagrammes d'états-transitions UML en Réseaux de Petri "nets within nets" de manière efficace et automatisée dans l'environnement AtomPM, tout en préservant les significations et les caractéristiques originales du modèle ? Et comment cette transformation peut-elle contribuer à fournir un moyen pour étudier et analyser de manière plus approfondie les systèmes complexes et interconnectés ?

Objectif (Contribution)

L'objectif de ce mémoire sur le thème "Transformation des diagrammes d'´états-transition UML en Réseaux de Petri nets within nets pour des fins de vérification" est de proposer une méthode efficace pour convertir les diagrammes d'états-transitions UML en modèles formels basés sur les Réseaux de Petri nets within nets.

La contribution principale de ce travail est de fournir une approche de transformation qui permettra aux ingénieurs de modéliser des systèmes complexes de manière plus précisé et de faciliter l'analyse formelle des propriétés du système. Cette transformation vers le formalisme des Réseaux de Petri nets within nets offre une représentation formelle puissante, capable de capturer la dynamique et la concurrence présentes dans les systèmes complexes.
Les Réseaux de Petri, également appelés "Petri nets within nets", sont un formalisme bien adapté pour représenter les spécifications des systèmes et permettent une analyse formelle ainsi qu'une vérification efficace des propriétés du système.
En transformant les diagrammes d'états-transitions en Réseaux de Petri nets within nets, ce mémoire contribue significativement à l'avancement de l'Ingénierie Dirigée par les Modèles dans le domaine de la vérification des systèmes.
Le travail présenté dans ce mémoire démontre sa pertinence et son efficacité dans la modélisation des systèmes complexes, et ouvrir de nouvelles perspectives pour la conception et le développement de systèmes critiques en utilisant des méthodes de vérifications formelles.

Organisation (Structure du manuscrit)

En commence par un introduction générale ,ce mémoire est structuré en quatre chapitres. Le premier chapitre se consacre à l'exploration des concepts relatifs aux diagrammes d'états-transitions UML et aux réseaux de Petri, notamment en mettant l'accent sur le formalisme "Nets Within Nets" pour la modélisation et l'analyse du comportement des systèmes. Par la suite, nous présentons quelques travaux connexes sur le domaine de la transformation des diagrammes d'états-transitions UML vers des méthodes formelles.

Le deuxième chapitre est dédié à la modélisation et à la méta-modélisation. Nous présentons les méta-modèles pour les diagrammes d'états-transitions UML ainsi que pour les réseaux de Petri nets whithin nets. De plus, nous présentons l'outil de transformation de graphes ATomPM, qui joue un rôle essentiel dans la mise en œuvre de ces transformations.

Le troisième chapitre est dédié à la présentation de nos règles de transformation de graphe, lesquelles permettent de convertir efficacement tout diagramme d'états-transitions UML en réseaux de Petri Nets Within Nets.

Le quatrième chapitre présente des études de cas pour illustrer notre approche de transformation de graphe.

La conclusion résume les points essentiels de ce travail et présente les perspectives de recherches suggérées par ceux-ci.

PARTIE 1 : état de l’art

Chapitre 1

contexte et état de l'art

Introduction :

La modélisation des systèmes informatiques est un processus essentiel pour les développeurs de logiciels, car elle permet de décrire le comportement attendu du système et de s'assurer que tous les éléments du système fonctionnent ensemble de manière cohérente. UML est un langage de modélisation visuelle très utilisé dans le domaine du développement logiciel. Il offre une variété de diagrammes qui permettent aux développeurs de modélisation de divers aspects des systèmes informatiques.

Parmi les diagrammes UML les plus utiles, citons les diagrammes d'états-transitions, qui permettent de modéliser le comportement des objets et des systèmes réactifs. Les diagrammes d'états-transitions permettent de modéliser le comportement d'un objet ou d'un système. Ils montrent comment un objet ou un système passe d'un état à un autre en réponse à des événements spécifiques. En outre, le formalisme de Nets whithin nets constitue une technique de modélisation avancée pour la description de systèmes complexes en les divisant en sous-systèmes modélisés séparément. Cette technique est particulièrement utile aux systèmes de modélisation qui ont des interactions complexes entre différents sous-systèmes.

1.1 UML 1.0 :

UML 1.0 est un langage graphique utilisé pour visualiser, spécifier, construire et documenter les systèmes logiciels. Il offre un moyen standard de planification d'un système, en incluant des éléments conceptuels tels que les processus opérationnels et les fonctions, ainsi que des éléments concrets comme les énoncés de programmation et les composants logiciels réutilisables. UML est le résultat de la fusion de trois méthodes de modélisation objet : OMT, Booch et OOSE. La version 1.0 d'UML a été soumise à l'OMG en 1997 et a été approuvée comme norme. UML est utilisé dans de nombreux projets à travers le monde et est considéré comme un langage de modélisation uniforme. Il propose une variété de diagrammes pour représenter différents aspects des systèmes et facilite l'analyse des exigences de conception et de mise en œuvre. Les treize types de diagrammes UML, y compris les nouveaux introduits par UML2, se complètent mutuellement pour offrir une vue d'ensemble du système modélisé.[OMG]

1.2 Diagramme d'états-transition UML :

Pour mieux comprendre un système, il faut d'abord en examiner la structure statique, en d'autres termes, la structure des objets et les rapports entre eux. une période donnée (le modèle de classe).

Il est alors préférable d'examiner ces amendements les objets et leurs liens dans le temps (le modèle d'état).

Le modèle d'état consiste en plusieurs schémas de transition d'état, un pour Chaque classe présente un comportement temporel important en ce qui concerne l'application. Le diagramme d'état de transi-

tion est une représentation graphique d'automates d'état fini qui relie des événements et des états. Les événements constituent des stimuli extérieurs et les états présentent les valeurs des objets.[10]

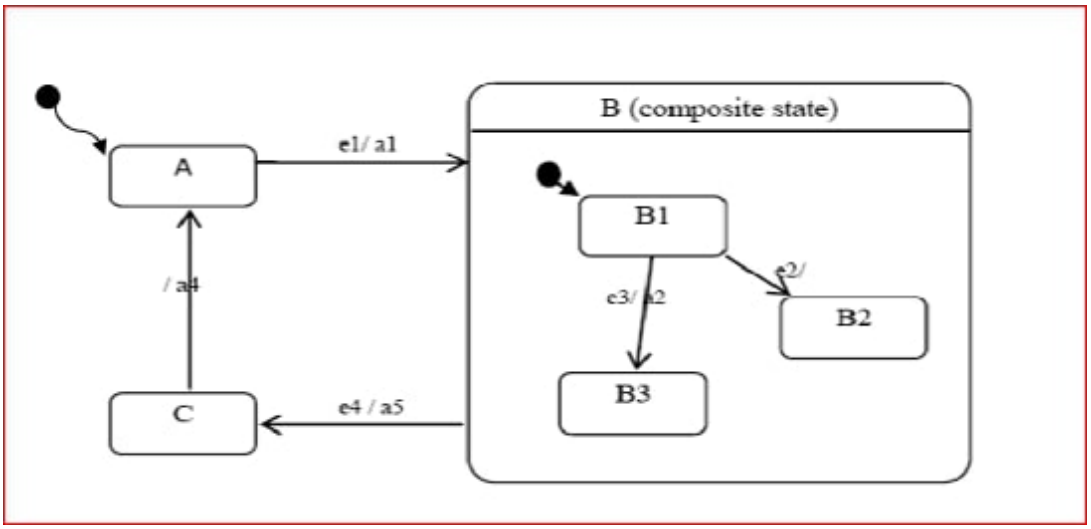

FIGURE 1.1 – Exemple de diagrammes d'états-transitions.[10]

1.2.1 Généralités :

Le diagramme d'états-transitions UML est un diagramme utilisé dans l'ingénierie logicielle pour décrire le comportement interne d'un objet qui utilise un automate à état fini. Ça représente les états et les séquences possibles d'actions qu'une instance de classe peut traiter au cours de son cycle de vie en réaction à des événements discrets. Le diagramme d'états-transitions donne un aperçu global du comportement de l'élément auquel il est attaché. Mais cela ne nous permet pas de comprendre le fonctionnement du système global puisqu'il ne s'intéresse qu'à un seul élément du système. Ce qui fait la différence d'un diagramme d'interaction qui permet de lier des éléments du système et qui n'offre qu'une vue partielle correspondant à un scénario.
Concrètement, un diagramme d'états-transitions est un graphe qui représente un automate, il commence généralement par un rond noir plein il s'appelle l'état initiale et se termine par un rond noire plein encerclé indiquant l'état finale. Autrement dit, une machine dont le comportement de production ne fonctionne pas. Ne dépend pas seulement de la sollicitation de l'entrée, mais aussi d'un historique des sollicitations passées. La machine modifie son état pour réagir aux ´événements externes entraînant une transition entre les états. [1]

1.2.2 Applications :

Les diagrammes d'états-transitions s'appliquent de deux manières :

• pour modéliser le comportement d'un objet réactif complexe en réponse à des événements. Par exemple : le téléphone, la voiture, la commande, la personne ...etc.

• pour modéliser des séquences d'opérations légales – spécialisations de protocole ou de langage ou processus. Par exemple : le protocole TCP, la navigation, la session. . .etc.

Le second moyen d'application peut être considéré comme la spécialisation du premier si l'objet est une langue, un protocole ou un processus. Utilisez seulement les diagrammes de transition d'état pour les objets dépendant d'un état avec un comportement complexe. [2]

1.2.3 Concepts :

Cette partie présente quelque notation de base sur le diagramme d'états-transitions UML.

État simple :

Un état est la condition d'un objet au-dessous duquel l'objet donne la même réponse. Cette situation se caractérise par une série de valeurs des attributs d'un objet.
Un objet peut passer à travers un ensemble d'états au cours de sa vie. Un état est une période de la vie d'un objet durant laquelle il attend un événement ou exécute une activité. La configuration de l'état général de l'objet est la série d'états qui sont actifs à un moment donné.[3]

FIGURE 1.2 – Représentation d'un état simple.

État initial :

L'état initial d'un objet est défini à l'aide d'un rond noir plein.
L'état initial, pseudo-état, indique l'état de départ par défaut, quand les transitions du diagramme d'état, ou l'état d'emballage, est invoqué.[3]

FIGURE 1.3 – Représentation graphique de l'état initial.

État final :

L'état final est spécifié avec un rond noir encerclé. L'état final, état particulier, indique que le diagramme d'états-transitions, ou l'état enveloppant, est terminé.[3]

FIGURE 1.4 – Représentation graphique de l'état final.

Événement :

Un événement est une occurrence ou un fait qui survient pendant l'exécution d'un système et qui mérite une modélisation. Lorsqu'un évènement est reçu, une transition peut être déclenchée et l'objet peut être changé d'état. Il est possible de diviser les événements en plusieurs types explicites et implicites : signal, appel, changement et temporel[4]

. Événement de type signal :

Un signal est un type explicite destiné à transmettre une communication asynchrone unilatérale entre deux objets. [5]

Un évènement de signal est un évènement dans lequel un signal est émis ou reçu. Il peut s'agir à la fois de l'expéditeur et du destinataire. [4]

. Événement d'appel :

Un événement d'appel correspond à la réception d'un appel de transaction par un objet. Les paramètres de l'opération sont ceux de l'événement d'appel. [3]

Remarquez que la syntaxe d'un évènement d'appel est identique à celle d'un signal. En revanche, les évènements d'appel sont des méthodes rapportées au niveau du diagramme de classe.

. Événement de changement :

Un évènement change est généré en satisfaisant une expression booléenne. L'expression booléenne est testée sans interruption.[3]

. Événement temporel :

Les événements temporels proviennent du passage du temps absolu ou du temps relatif.[3] Elle est soit absolue , soit relative .

Par défaut, la durée écoulée commence lors de la saisie de l'état actuel.

Le tableau 1.1 associe plusieurs types d'événements.

Type d'événement	Description	Syntaxe
Appel	Recevoir une demande d'appel explicite synchrone à un objet.	Op (a :T)
Changement	Changement de la valeur de l'expression booléenne	When (exp)
Signal	Recevez une connexion externe asynchrone nommée entre les objets.	sname (a :T)
Temporel	Arrivée d'un temps absolu ou un laps de temps relatif.	after (time)

Table 1.1 – Tableaux des types d'évenements.

Transition :

Une transition est la transition immédiate d'un état à un autre dans l'éventualité d'un événement. Elle lie un état source et un état cible et indique qu'un objet dans un état source peut exécuter certaines activités et entrer dans l'état cible, si un événement déclencheur se produit et que la condition de garde est vérifiée.[3]

Le même événement peut provoquer des transitions multiples. Ces transitions doivent s'accompagner de dispositions différentes en matière de garde. Si deux transitions sont activées en même temps par le même évènement, un seul choix aléatoire sera déclenché.
Le tableau 1.2 résume les différents types de transitoires ainsi que leurs conséquences.

Type de transition	Description	Syntaxe
Transition entry	L'activité est exécutée lorsque l'objet atteint cet état.	Entry/activité
Transition exit	L'activité est exécutée lorsque l'objet quitte cet état.	Exit /activité
Transition externe	Réponse à un événement extérieur ou à une auto-transition.	a :T)[condition]/activit
Transition interne	Réponse à un événement qui n'entraîne pas de changement de statut.	e (a : T) [condition]/acti

TABLE 1.2 – Tableaux des types de transition.

Point de choix :

Il est possible de représenter des alternatives pour le franchissement d'une transition. Des pseudo-états spéciaux sont utilisés : points de jonction et points de décision.[4]

FIGURE 1.5 – Représentation graphique d'une Point de choix.

Point de jonction :

Les points de jonction, représentés par un petit cercle plein, sont un artefact graphique qui permet de partager des segments de transition, l'objectif étant d'aboutir à une notation plus compacte ou plus lisible des chemins alternatifs.

Il peut y avoir plusieurs segments de transition entrants et plusieurs segments de transition sortants. Cependant, il ne peut y avoir d'activité interne ou de transitions sortantes avec déclencheurs d'événements[7]

Il n'est pas un état qui peut être actif pendant une période de temps finie. Quand un chemin traversant un point de jonction est pris tous les gardes le long de ce chemin doivent s'évaluer à la vérité dès que le premier segment est franchi.

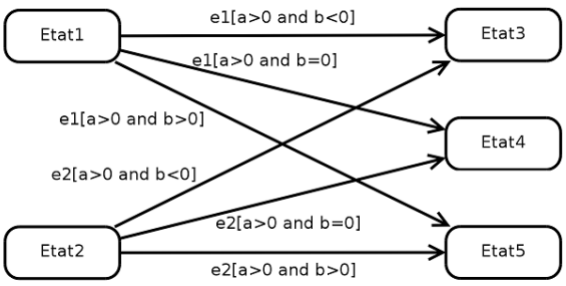

FIGURE 1.6 – Représentation d'un diagramme sans point de jonction [4].

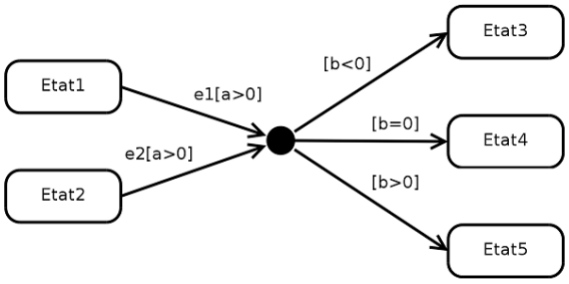

FIGURE 1.7 – Représentation d'un diagramme équivalent en utilisant un point de jonction [4].

Point de décision :

Un point de décision, représenté par un losange, possède une entrée et au moins deux sorties. Contrairement à un point de jonction, les protecteurs situés après le point de décision sont évalués au moment où celui-ci est atteint. On peut ainsi baser le choix sur les résultats obtenus en traversant le segment avant le point de choix. Si le point de décision est atteint et aucun segment en aval n'est accessible, le modèle est mal formé. [3]

Il est possible d'utiliser une garde particulière, notée [else], sur un des segments en aval d'un point de choix. Ce segment est seulement réalisable si les protections des autres segments sont toutes fausses. Il est recommandé d'utiliser une clause [else] après un point de décision parce qu'elle garantit un modèle bien formé.

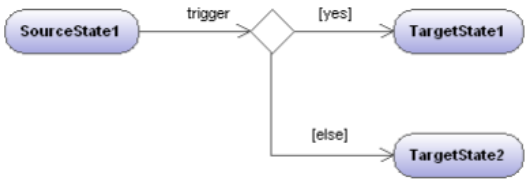

FIGURE 1.8 – Représentation graphique d'une Point de desision.

État composite :

Un état composite est un état décomposé en zones comprenant un ou plusieurs sous-états.

un état composite est un état avec une ou plusieurs régions, Une région est simplement un conteneur pour les sous-états et Un état composite comportant au moins deux régions est dénommé orthogonal. Chaque région à un état composite est réalisée en parallèle une transition à l'état final d'une région indique la fin de son activité.

Lorsque toutes les régions sont complètes, l'état composé déclenche un événement d'achèvement et une transition d'achèvement déclenche.

Implicitement, tout diagramme des états transitoires est contenu dans un état externe qui n'est pas normalement représenté. Ceci fournit une plus grande homogénéité dans la description : n'importe quel diagramme d'états de transition est implicitement un état composite. [6]

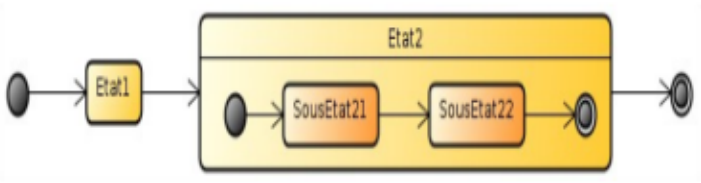

FIGURE 1.9 – Représentation graphique d'une état composite.

État historique :

Un état historique, aussi appelé état historique plat, est un pseudo-état qui stocke le dernier état secondaire actif d'un état composite. Sur le plan graphique, il est représenté par un cercle avec un H.

Vous pouvez aussi définir une histoire profonde représentée graphiquement par un cercle contenant un H*. Cet état historique profond permet d'atteindre le dernier état visité dans la région, quel que soit son niveau d'imbrication, alors que le l'état historique plat limite l'accès aux états de son niveau d'imbrication [3].

FIGURE 1.10 – Représentation d'un état historique plat et d'un état historique profond.

Une transition qui cible l'Etat historique est l'équivalent d'une transition qui cible le dernier Etat visité de l'Etat englobant. Un État historique peut avoir une transition sans étiquetage indiquant l'État à exploiter si la région n'a pas encore été visitée.

Points de connexion :

Pour entrer ou sortir d'un état composite sans passer par l'état initial ou l'état final, il faut utiliser les points de connexion et les placer sur la frontière de l'état composite.

Les points de connexion permettent une transition à travers la limite de l'état composite et visent directement un état interne ou externe à cet état composite.

Les points de connexion sont des points d'accès et de sortie situés à la limite d'un état composite. Ceux-ci sont respectivement représentés par un cercle vide et un cercle traversé par une croix. C'est juste une référence à un état défini en état composite.[3]

FIGURE 1.11 – Représentation des points de connexion.

Ils sont interprétés comme des références à un état cible. Lorsqu'une transition est dirigée vers le point de liaison, l'extension de cette transition indique l'état cible. Ils autorisent également une notation plus compacte ou lisible des chemins alternatifs.

Etat orthogonal :

Lorsqu'un état composite a plusieurs régions, on l'appelle un état orthogonal. Lorsqu'un état orthogonal est actif, un sous-état direct de chaque région est simultanément actif, il y a donc concurrence. Un état composite comportant une seule région est considéré comme un état non orthogonal [4].

Graphiquement, à l'état orthogonal, les différentes régions sont séparées par une ligne pointillée horizontale entre la gauche et le bord droit de l'état composite.

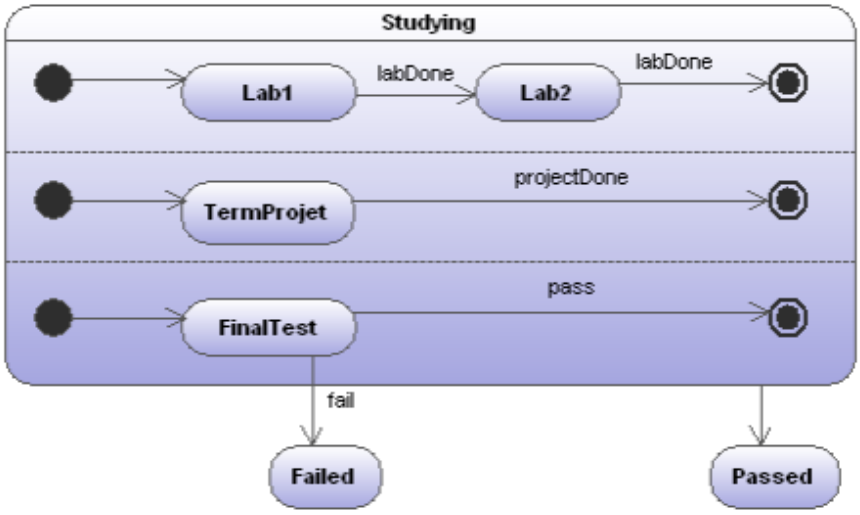

FIGURE 1.12 – Exemple d'utilisation d'un état orthogonal pour modéliser le fait que la réussite d'un cours implique trois travaux en parallèle [8].

Chaque région présente un flux d'exécution, peut avoir un état initial et final. Une transition qui atteint la bordure d'un état composite orthogonal est équivalente à une transition qui atteint les états initiaux de toutes ses régions concurrentes. Toutes les régions concurrentes à l'état composite orthogonal doivent atteindre leur état final avant que l'état composite ne soit considéré comme complet [8].

1.3 Réseaux de pétrie :

Il est évident que les systèmes dynamiques ne peuvent être décrits uniquement en fonction de leur état initial et de leur état final. Une description correcte devrait prendre en compte leur comportement permanent qui est une série d'états. Un certain nombre de techniques officielles ont déjà été proposées pour la spécification, l'analyse et la vérification de tels systèmes. Les réseaux de Petri sont une technique formelle largement utilisée.[19]

1.3.1 Les concepts de base de réseaux de petri :

Dans cette section, introduisez les notions de base d'un réseau petri et définissez chaque notion. Un réseau Petri permet de modéliser le comportement des systèmes dynamiques à l'aide d'événements discrets, ainsi que la description des relations existantes entre les conditions et les événements.[19]

Un réseau de Petri est un graphe biparti orienté (comportant deux types de noeuds) : les places représentées par des c ercles et les transitions représentées par des rectangles.[19]

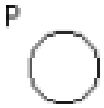

FIGURE 1.13 – Représentation d'une place.

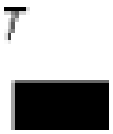

FIGURE 1.14 – Représentation d'une transition.

Les arcs du graphe ne peuvent relier que des places vers des transitions, ou des transitions vers des places.

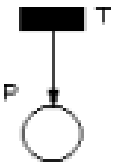

FIGURE 1.15 – Représentation d'un arc relie soit une transition à une place.

Un réseau de petri décrit un système dynamique avec des évènements distincts. Les places permettent la description des états possibles du système et les transitions permettent la description des événements ou les actions qui modifient la situation. Un réseau de Petri est un graphe muni d'une sémantique opérationnelle, c'est-à-dire qu'un comportement est associé au graphe, ce qui est utilisé pour décrire la dynamique du système présenté. Pour cela un quatrième élément est ajouté aux places et aux transitions c'est les jetons.[19]

FIGURE 1.16 – Représentation d'un arc relie soit une place à une transition.

1.3.2 Exemples de Réseaux de Petri :

La réaction chimique suivante se produisant en présence d'un catalyseur (platine) est maintenant prise en considération :[19]

$$H_2 + C_2H_4 \longrightarrow C_2H_6$$

L'état du système est caractérisé par :

H_2 :c'est le nombre de marque dans la place P1.
C_2H_4 :c'est le nombre de marque dans la place P2.
platine :c'est le nombre de marque dans la place P3.
C_2H_6 :c'est le nombre de marque dans la place P4.

L'état du système va évoluer quand La réaction chimique se produit (franchissement de la transition T1).

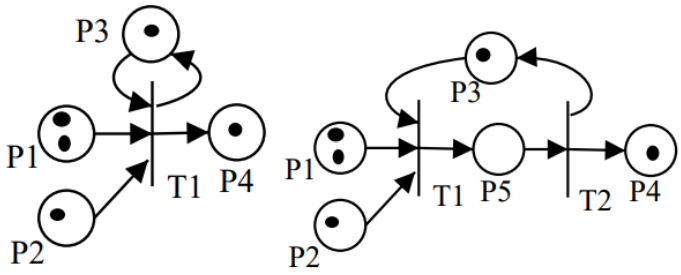

FIGURE 1.17 – Réaction chimique.

Nous supposons que dans un premier temps nous avons deux unités de H2, une unité de C2H4, une unité de C2H6 et que la plaque est libre. Cela donne le RdP indiqué dans la figur 1.17, à gauche. Le nombre de marques dans chaque place correspond ici à chaque produit chimique. La présence de platine est indispensable à la réaction chimique : le place P3 doit être au début de la transition T1. Toutefois, lorsque la réaction chimique a lieu, le platine étant un catalyseur, celui-ci n'est pas consommé. En conséquence, l'emplacement P3 doit sortir de la transition T1 afin de restaurer la ressource après que la réaction se soit produite.[19]

Cette réaction devait être instantanée. En fait, la réaction a une certaine durée caractérisée par un début (où les réactifs se fixent au catalyseur pour réagir) et par une fin (les réactifs ont fini de réagir et libère le catalyseur). Ainsi, un emplacement P5 supplémentaire est introduit qui indique si oui (une marque dans P5) ou pas (aucune marque dans P5) les produits réagissent sur la surface du catalyseur. Une autre transition T2 est introduite : le franchissement T1 correspond maintenant au début de la réaction chimique et le franchissement T2 à la fin de la réaction chimique. On obtient alors le RdP représenté Figure 1.17, droite.[19]

1.3.3 Le marquage d'un réseau de petri :

Chaque place contient un nombre entier positif ou nul de marques ou jetons. Le marquage M définit l'état du système décrit par le réseau à un instant donné. C'est un vecteur colonne de dimension le nombre de places dans le réseau. Le i éme élément du vecteur correspond au nombre de jetons contenus dans la place Pi.[19]

1.3.4 Le graphe de marquage :

Le graphe de marquage est utilisé lorsque le nombre de repères accessibles est terminé. Dans cet exemple, le graphe de marquage d'un RDP sera extrait :[19]

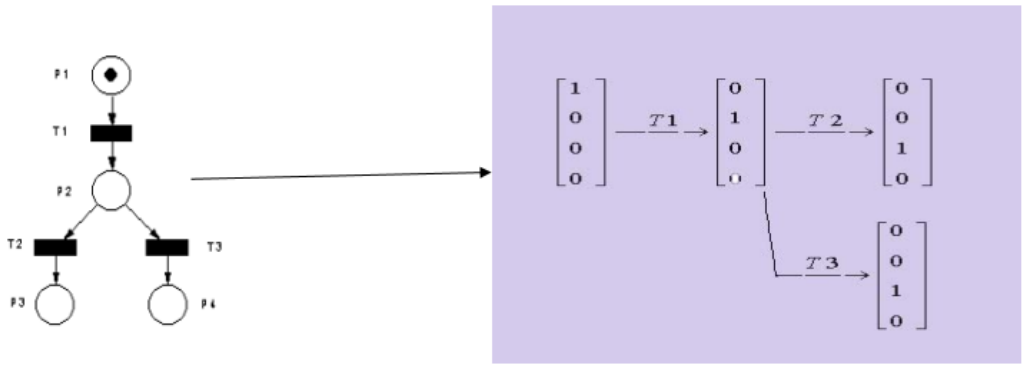

FIGURE 1.18 – Représentation d'un graphe de marquage.

1.3.5 Propriétés des Réseaux de Petri :

Les Propriétés Structurelles :

Les propriétés de structure dépendent seulement de la topologie du réseau. Il s'agit de faire ressortir les propriétés statiques du système étudié. Ces nombreuses propriétés sont. indépendantes du marquage. Ainsi, il est possible de faire apparaître, entre autres, les caractéristiques de synchronisation ou de précédence .[19]

. Les conflits :

Un Rdp sans conflit est un réseau dans lequel chaque place a au plus une transition de sortie. Un RdP avec conflit est un réseau qui possède donc une place avec au moins deux transitions de sorties. Un conflit est noté : [Pi , T1,T2,. . .,Tn] ; avec T1,T2,. . .,Tn étant les transitions de sorties de la place Pi.[19]

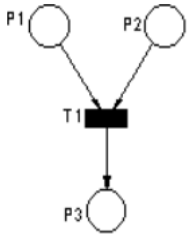

FIGURE 1.19 – Représentation d'un graphe sans conflit.

.RdP à choix libre :

Un RdP est à choix libre est un réseau dans lequel pour tout conflit [Pi , T1,T2,. . .,Tn] aucune des transitions T1,T2,. . .,Tn ne possède aucune autre place d'entrée que Pi .[19]

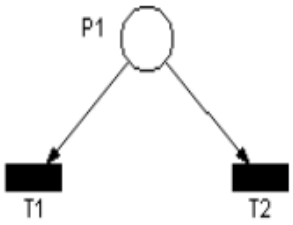

FIGURE 1.20 – Représentation d'un graphe avec conflit et avec choix libre.

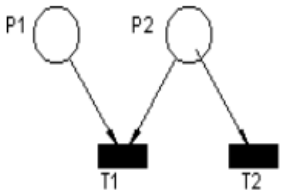

FIGURE 1.21 – Représentation d'un graphe avec conflit et sans choix libre.

. RdP simple :

Un RdP simple est un RdP dans lequel chaque transition ne peut être concernée que par un conflit au plus. Un réseau Petri simple est un RdP dans lequel chaque transition ne peut être affectée que par. un conflit au plus.[19]

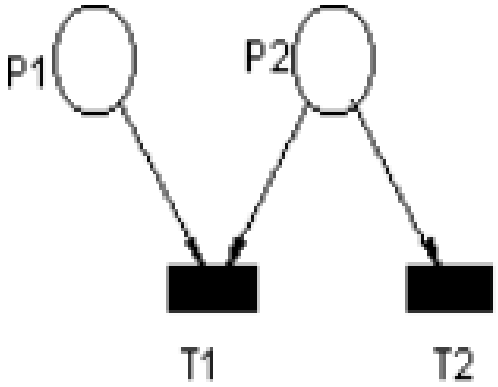

FIGURE 1.22 – Représentation d'un graphe avec conflit ,sans choix libre et simple.

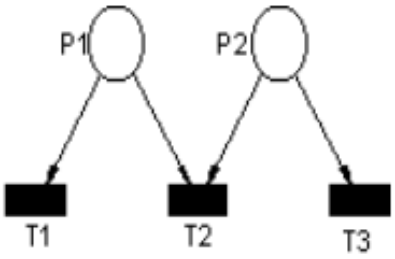

FIGURE 1.23 – Représentation d'un graphe avec conflit ,avec choix libre et non simple.

. RdP pur :

Un RdP pur est un réseau dans lequel il n'existe pas de transition ayant une place d'entrée qui soit à la fois place de sortie de cette transition.Un RdP simple est un RdP dans lequel chaque transition ne peut pas être concernée que par un conflit au plus.[19]

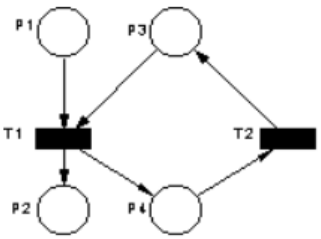

FIGURE 1.24 – Représentation d'un Rdp pur.

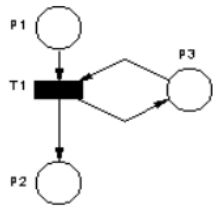

FIGURE 1.25 – Représentation d'un Rdp non pur.

. Un graphe d'état :

Un réseau de Pétri non marqué est un graphe d'état si et seulement si toute transition a exactement une seule place d'entrée et une seule place de sortie.

dans cette figure Chacune des transitions possède une seule place d'entrée et une seule place de sortie.[19]

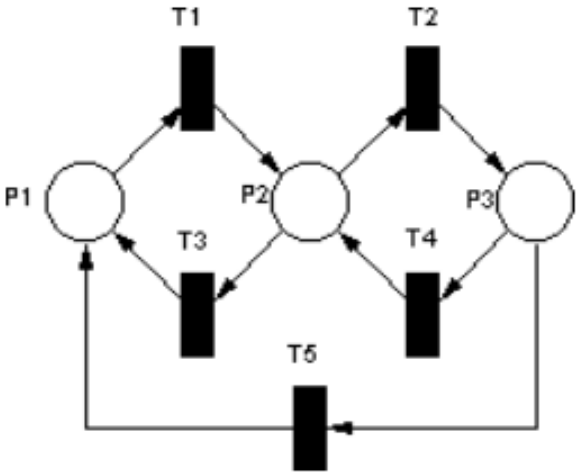

FIGURE 1.26 – Représentation d'un graphe d'état.

. Un graphe d'événement :

Un RdP est un graphe d'événement si et seulement si chaque place possède exactement une seule transition d'entrée et une seule transition de sortie.[19]

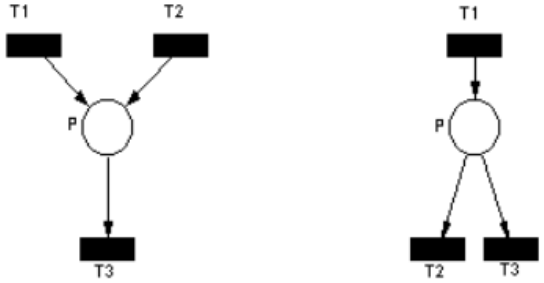

FIGURE 1.27 – Représentation d'un graphe non événement.

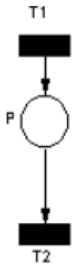

FIGURE 1.28 – Représentation d'un graphe événement.

. RdP généralisés :

Un RdP généralisé est un RdP dans lequel des poids (nombres entiers strictement positifs) sont associés aux arcs. Si un arc (Pi ,Tj) a un poids k : la transition Tj n'est franchie que si la place Pi possède au moins k jetons. Le franchissement consiste à retirer k jetons de la place Pi .
Si un arc (Tj ,Pi) a un poids k : le franchissement de la transition rajoute k jetons à la place Pi .
Lorsque le poids n'est pas signalé, il est égal à un par défaut.[19]

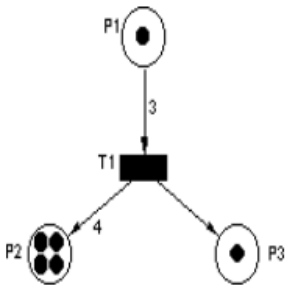

FIGURE 1.29 – Représentation d'un Rdp après franchissement.

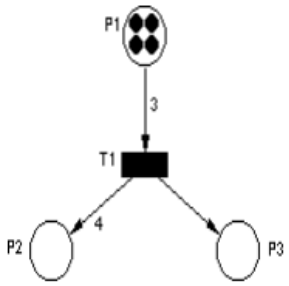

FIGURE 1.30 – Représentation d'un Rdp avant franchissement.

1.4 Nets Within Nets :

Nets Within Nets sont des réseaux Petri de haut niveau qui permettent aux réseaux d'être intégrés dans des structures dynamiques.[20]
A la différence des réseaux ordinaires, où des jetons sont des éléments passifs, les jetons dans les nets whithin nets sont des éléments actifs, c'est-'a-dire les jetons dans ces réseaux peuvent être considérés comme des objets.
En place/réseaux de transition, ces objets représentent généralement des ressources ou indiquent l'état de contrôle. Des objets plus complexes sont modélisés à l'aide de jetons tapés dans des réseaux de Petri colorés. La modélisation orientée objet signifie toutefois que le logiciel est conçu comme une interaction d'objets discrets, intégrant à la fois la structure de données et le comportement. Cette démarche est décrite comme un paradigme de réseaux interne .[20]
De façon générale, Il y a deux types de sémantique des jetons : la sémantique de la valeur et la sémantique de la référence.
Dans la sémantique des valeurs, on peut considérer les jetons comme des représentations directes de réseaux.[20]
Ceci permet d'avoir les réseaux imbriques qui sont structurés hiérarchiquement, parce que les réseaux ne peuvent être situés qu'en un seul lieu. Au niveau de la sémantique de référence on peut obtenir des structures d'imbrication de réseau arbitraires parce que les jetons représentent des références à des réseaux. Celles-ci peuvent être hiérarchisées, acycliques ou même cycliques.
Dans les sections ci-dessous, nous traiterons des réseaux de référence pour trois raisons. En premier lieu, ils sont soutenus par l'outil Renew [13] principes de base des réseaux à l'intérieur des nets whithin nets, deuxièmement, ils montrent les principes de base des nets within nets, et troisièmement, ils permettent des structures d'imbrication acycliques. Lorsque le nombre de références pour les réseaux est réduit à une référence unique pour un réseau, la sémantique des valeurs est égale à la sémantique des références.[20]

1.4.1 sémantique des références :[20]

Les réseaux de référence sont des réseaux de Petri de haut niveau orientés objet, dans lesquels les jetons peuvent redevenir des réseaux. Pour ces réseaux au sein des réseaux, on suppose une sémantique de référence, les jetons d'un réseau peuvent faire référence à d'autres réseaux.
Dans un simple ajustement d'un réseau unique, le réseau externe est appelé réseau système tandis qu'un jeton dans le réseau système se réfère à un réseau objet. Toutefois, les réseaux d'objets eux-mêmes peuvent à nouveau contenir des jetons qui représentent des réseaux, ce qui permet d'obtenir un système de filets imbriqués.
De cette fonctionnalité est que le système modelé est modulable et extensible. De plus, les transitions dans les réseaux peuvent autoriser et déclencher des transitions dans d'autres réseaux.
Comparativement aux éléments nets des réseaux Petri., les réseaux de référence offrent plusieurs éléments additionnels qui augmentent la puissance de modélisation ainsi que la commodité de la modélisation.
Ces éléments supplémentaires incluent certains types d'arcs, des emplacements virtuels et un relevé. Plusieurs types d'inscriptions ont été ajoutés aux éléments du réseau afin de permettre leur utilisation. Les lieux peuvent être tapés et les transitions peuvent être complétées par des expressions, des actions, des gardes, des canaux synchrones et des inscriptions créatrices. Dans ce travail on ne s'intéresse pas par

ces éléments supplémentaires.

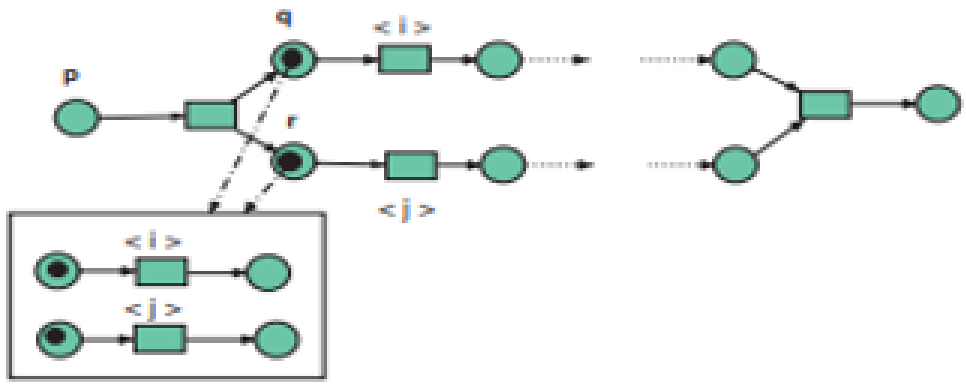

FIGURE 1.31 – Sémantique de référence.

1.4.2 la sémantique des valeurs :[20]

La figure 1.32 présente un exemple pour la sémantique des valeurs.

Dans la sémantique des valeurs, chaque instance de réseau objet a son propre état (marquage), Lorsqu'une interaction se produit avec une transition, où plusieurs réseaux d'objets marqués sont impliqués du côté entrée, une sorte d'unification de leur état actuel (marquage) doit être construite. Cela correspond à la collecte de résultats partiels (de calculs simultanés) à un état cohérent, unifiant ces états partiels.

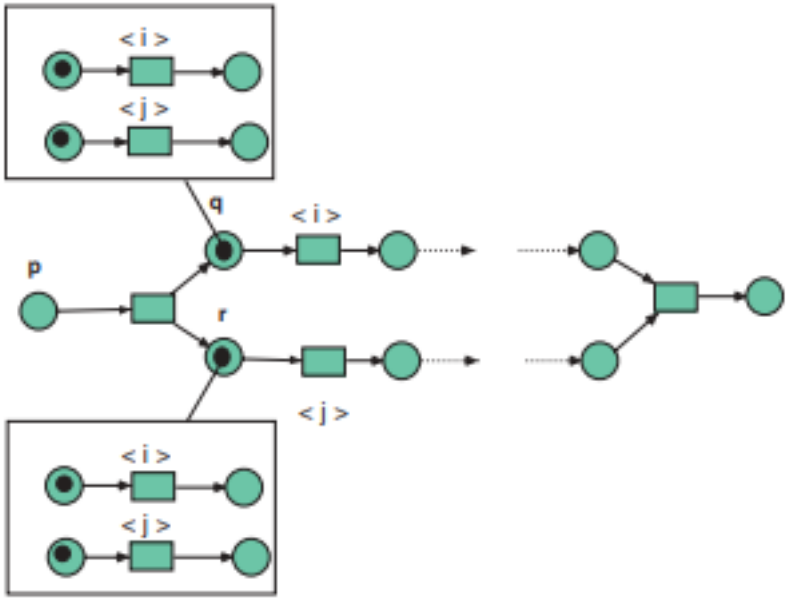

FIGURE 1.32 – Sémontiques de valeur.

1.5 Travaux connexes :

Dans cette partie, nous expliquerons certains travaux connexes sur la génération des méthodes et outils formelles à partir des diagrammes d'états-transitions UML :

1.5.1 Vérification de la correction du diagramme d'états-transitions UML de la clinique ambulatoire basée sur le langage de modélisation commun et SMV [9].

Dans cet article, les auteurs ont utilisé CML (Common Modeling Language) pour modéliser le diagramme d'états-transitions UML dans le langage formel de SMV et exprimer des propriétés dans la logique du temps telles que CTL (Computationnel Tree Logic).

Cette approche est utile pour aider les gens à utiliser des vérificateurs de modèles et elle est également bénéfique pour simplifier les tâches difficiles liées à l'utilisation de la vérification de modèles, telles que la modélisation formelle et la formalisation des propriétés d'un système. Cependant, cette approche ne s'attaque qu'à la vérification des diagrammes d'états-transitions UML simples.

1.5.2 Génération d'expressions LOTOS à partir de diagrammes UML [10] :

Dans ce travail de recherche les auteurs ont exposé une approche pour étudier et mettre en œuvre des techniques qui facilitent l'utilisation commune du langage orienté objets UML et du langage formel LOTOS, en particulier pour la modélisation et la vérification des comportements dynamiques des systèmes complexes.

Cette approche permet aux concepteurs à modéliser les comportements de leurs systèmes rapidement et graphiquement en utilisant les diagrammes d'états-transitions et les diagrammes de communication UML, et traduire leurs modèles UML vers des spécifications formelles en LOTOS en utilisant les techniques de la transformation de graphe et l'outils AToM3. Pour faciliter la transformation des diagrammes d'états-transitions complexe vers LOTOS, les auteurs ont utilisé des modèles FSM (Flat State Machine) comme une étape intermédiaire dans leur processus de transformation.

1.5.3 Modélisation Multiparadigme : Une Approche Basée sur la Transformation de Graphes [11] :

L'objectif principal de cette étude est la "Modélisation Multi-paradigmes" ainsi que la conception d'une approche basée sur la Transformation de Graphes destinée à faciliter l'utilisation des techniques d'analyse formelle au cours du cycle de développement des systèmes complexes.

Parmi les contributions de ce travail, la formalisation du langage UML par le biais des réseaux de Petri colorés. Un outil développé basé sur le Meta-modélisation et les grammaires de Graphes pour convertir un digramme États-Transitions UML et un diagramme de collaboration UML en un modèle réseaux de Petri colorés pour tester les propriétés comportementales des systèmes. Dans ce travail les auteurs ont utilisé aussi des modèles FSM comme une étape intermédiaire dans leur processus de transformation des diagrammes d'états-transitions en modèle réseaux de Petri colorés.

1.5.4 Transformation des diagrammes d'états-transitions vers Maude [12] :

Ce travail de recherche consiste à proposer une grammaire de graphe vise à convertir les diagrammes d'états-transitions UML vers une notation formelle du langage Maude en utilisant l'outils de transformation de graphe AToM3. Les diagrammes d'états-transitions UML représentent en vérité des automates d'état fini, d'un point de vue mathématique, ils représentent des graphiques orientés. Par contre le langage Maude est un langage de description et de programmation fondé sur la logique de réécriture. Le processus de transformations s'est composé en deux étapes. La premiere étape consiste à convertir les diagrammes d'états-transitions UML composites (simultanés/séquentiels) en diagrammes d'états-transitions plats à l'aide d'algorithmes Saldhana. Tandis que la seconde étape consiste à traduire les diagramme d'états-transitionsplat en Maude.

Discussion :

Les travaux mentionnés en dessus, ont exploré des méthodes pour convertir ces diagrammes d'états-transitions UML en outils formels. Cependant, la plupart de ces méthodes ont utilisé des versions simplifiées des diagrammes d'états-transitions ou ont transformé des diagrammes d'états-transitions complexes et concurrents en versions à plat (FSM :"Flat state machine" comme exemple). Il est important de noter que cette transformation conduit à une perte de la sémantique des diagrammes d'états-transitions.

Ce travail de recherche vise à surmonter les limitations associées à la génération des outils formelles à partir des diagrammes d'états-transitions complexes en préservant la sémantique essentielle des modèles. Pour cela, nous proposons une méthode basée sur la transformation de graphes, exploitant l'outil AToMPM, pour générer des nets within nets à partir de diagrammes d'états-transitions UML complexes et concurrents tout en préservant leur sémantique d'origine. Les nets within nets ont utilisé pour vérifier diverses propriétés des diagrammes d'états-transitions UML, telles que la vivacité, l'interblocage, et bien d'autres.

Conclusion :

En conclusion, les diagrammes d'états-transitions UML et Nets Within Nets sont tous des outils importants pour modéliser les systèmes informatiques. Les diagrammes d'états-transitions sont particulièrement utiles pour la modélisation du comportement des objets et des systèmes réactifs, considérant que les Nets Within Nets sont utiles pour la modélisation de systèmes complexes en les divisant en sous-systèmes modélisables séparément. En utilisant ces outils de modélisation, les développeurs peuvent créer des modèles précis et faciles à comprendre pour des systèmes informatiques. Cela facilite la communication entre les membres de l'équipe et aide à déceler l'éventuelles erreurs de conception. Au final, la modélisation des systèmes informatiques est essentielle au succès d'un projet de développement logiciel, et les diagrammes d'états-transitions UML et les nets whithin nets sont des outils précieux pour aider les développeurs à atteindre ce but.

PARTIE 2 : Contribution

Chapitre 2

Méta-modélisation

Introduction :

La méta modélisation est une approche de modélisation qui consiste en la création d'un modèle de langage de modélisation. Cette approche est utilisée dans divers domaines, dont l'ingénierie des logiciels, pour créer des langages de modélisation spécifiques au domaine.

L'utilisation de la méta-modélisation permet de décrire les systèmes plus précisément et de façon plus détaillée, en tenant compte des interactions et des comportements complexes qui peuvent se produire.

En somme, la méta-modélisation de nets whithin nets et de diagrammes d'états-transitions UML est une approche puissante pour la modélisation de systèmes complexes. Elle permet de créer des langages de modélisation spécifiques à un domaine, de décrire le comportement des systèmes de manière détaillée, et faciliter la communication et la compréhension parmi les divers membres d'une équipe de développement.

Dans ce chapitre, nous présentons sommairement le principe et les concepts de méta-modélisation, ainsi que les différents types de transformation. Nous présentons également l'outil de transformation de graphe AToMPM et les deux méta-modèles : de diagramme d'états-transitions UML et les nets within nets.

2.1 AToMPM :

De nombreux outils de transformation de graphes ont été développés pour assurer d'une manière efficace et facile les transformations de modèles à l'aide des transformations de graphes, parmi lesquels nous pouvons citer : AToMPM.

Dans le cadre de ce mémoire, AToMPM est sélectionné parmi les autres pour sa capacité. AToMPM est conçu pour supporter plusieurs paradigmes de modélisation, tels que la modélisation graphique, la modélisation textuelle et la modélisation des machines d'état, AToMPM offre un plus large éventail de fonctionnalités, telles que la capacité de définir et d'appliquer des contraintes sur les modèles, la prise en charge des transformations de modèles et la génération de code à partir des modèles. En termes de convivialité, AToMPM fournit une interface utilisateur graphique (GUI) pour la définition et la manipulation de méta-modèles, ainsi que la génération de code à partir de modèles, tandis qu'AToMPM fournit une interface graphique plus interactive et animée pour la création et l'exploration de modèles graphiques.

Ce type de modélisation, permet aux utilisateurs d'utiliser des formalismes différents dans une même transformation qui est le cas dans notre transformation.

2.2 Présentation :

AToMPM (≪ A Tool for Multi-Paradigm Modelling ≫) est un établi de modélisation (meta), qui permet aux développeurs de langues de créer des langages visuels spécifiques au domaine, et aux experts du domaine d'utiliser ces langages. Un langage est défini par sa syntaxe abstraite dans un metamodel,

sa syntaxe concrète, qui définit la manière dont chaque élément syntaxique abstrait est visualisé,et sa définition sémantique, soit opérationnelle (un simulateur) soit translationnelle (par correspondance avec un champ sémantique connu). AToMPM permet de transformer un modèle en sémantique.
Dans cette section, nous allons vous donner un aperçu de l'interface utilisateur d'AToMPM. Nous vous montrerons comment créer un compte et vous présenterons les différentes parties de l'interface, notamment les barres d'outils, la toile et la console.

Le premier élément que l'utilisateur rencontre lorsqu'il accède à l'interface utilisateur est une page de connexion.(figure 2.1[9]
)

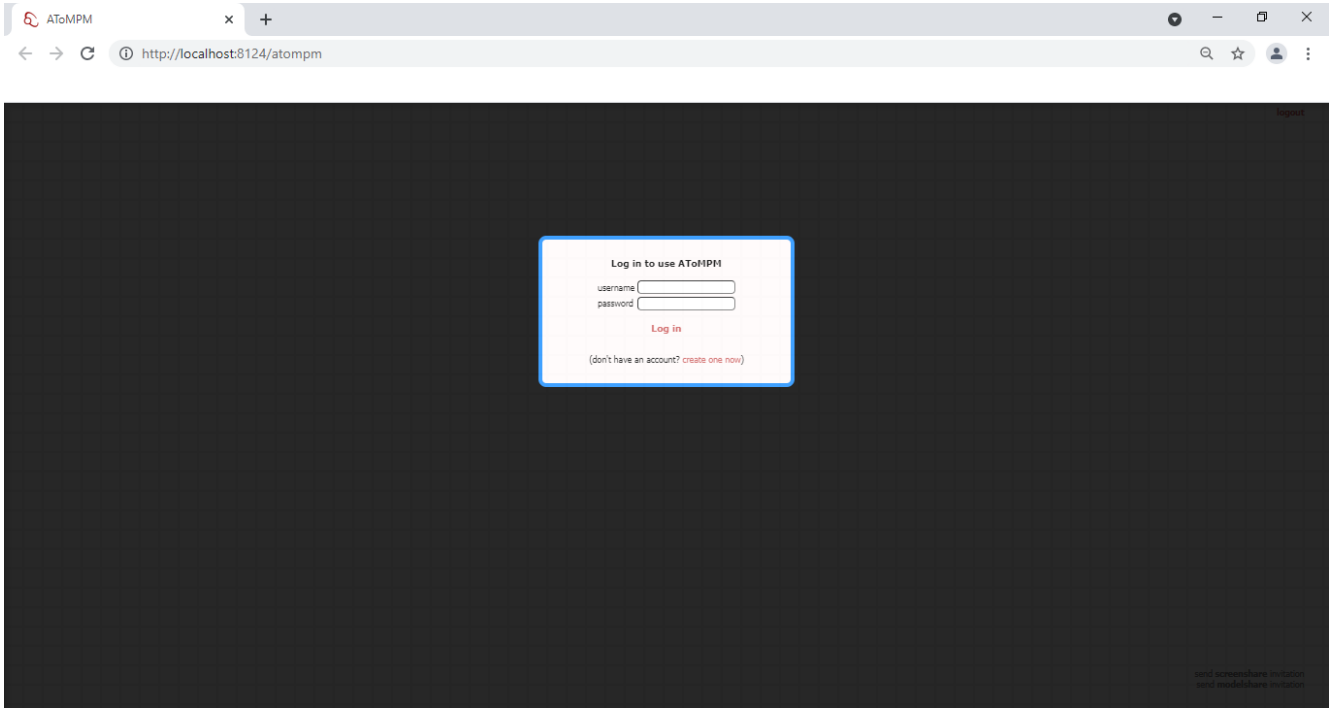

FIGURE 2.1 – L'interface utilisateur d'une page de connexion.

2.3 La méta-modélisation avec AToMPM :

AToMPM propose deux formalismes pour définir des langages de modélisation :le formalisme "graphe de classes"et le formalisme "automate d'états finis"

Les deux formalismes proposés par AToMPM ont des avantages et des inconvénients en fonction du domaine d'application et des besoins spécifiques de l'utilisateur. Le formalisme "graphe de classes" est plus adapté pour définir des langages de modélisation qui représentent des structures statiques, telles que des schémas de bases de données ou des diagrammes de classes UML. Le formalisme "automate d'états finis" est plus adapté pour définir des langages de modélisation qui représentent des systèmes dynamiques, tels que des protocoles de communication ou des machines à états.

En somme, la méta-modélisation avec AToMPM est un moyen puissant pour les développeurs de créer des langages de modélisation personnalisés pour leur domaine d'application, offrant une interface graphique intuitive pour la création et la modification de ces langages et permettant la génération automatique de code pour manipuler les modèles créés.[11]

Afin de clarifier le concept de méta-modélisation, nous allons tenir compte des définitions suivantes :

2.4 Le modèle :

Un modèle est une abstraction et une simplification d'un système qu'il représente. Cette simplification doit tenir compte des besoins et des objectifs à la base de l'étude et de la construction de ce système. Généralement,Un modèle doit pouvoir être utilisé pour répondre à des questions que l'on se pose sur lui. La notion de modèle dans l'IDM fait explicitement référence à la notion de formalisme ou de langage de modélisation bien défini. Un langage de modélisation est défini par une syntaxe abstraite, une syntaxe concrète et une sémantique. La syntaxe abstraite définit les concepts de base du langage. La syntaxe concrète définit le type de notation qui sera utilisé pour chaque concept abstrait qui peut être graphique, textuelle ou mixte. Enfin, la sémantique définit comment les concepts du langage doivent être interprétés par les concepteurs mais surtout par les machines.
En effet, pour qu'un modèle soit productif, il faut qu'il soit capable d'être manipulé par une machine. Le langage dans lequel ce modèle est exprimé doit donc être visiblement défini. De façon naturelle, la définition d'un langage de modélisation se présentait sous la forme d'un modèle, appelé méta-modèle.[11]

2.5 Méta-modèle :

Un méta-modèle est un modèle qui définit le langage de description d'un modèle [14] OMG. Soulignant le fait qu'un méta-modèle représente (modélise) les entités d'un langage, leurs relations ainsi que leurs contraintes, c'est-'a-dire une spécification de la syntaxe du langage
En outre, il devrait être capable de décrire la syntaxe concrète et la sémantique de ce langage .
Le Méta-modèle à son tour est exprimé dans un langage de méta-modélisation spécifié par le Méta-Méta-modèle. Le langage utilisé au niveau du méta-méta-modèle doit être suffisamment puissant pour spécifier sa propre syntaxe abstraite et ce niveau d'abstraction demeure largement suffisant (méta-circulaire). Chaque élément du modèle est une instance d'un élément du méta- modèle[11]. On dit qu'un modèle se conforme à un méta-modèle et qu'il représente un système existant ou imaginaire. La relation entre un méta-méta-modèle et un méta-modèle s'apparente à la relation entre un méta-modèle et un modèle. Cette relation est illustrée dans la Figure 2.2 :

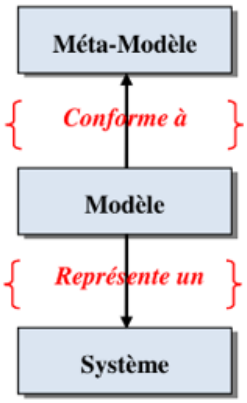

FIGURE 2.2 – Relation entre système,modèle et méta-modèle.[11]

2.6 La transformation des modèles :

Le concept de transformation d'un modèle est au cœur de l'approche IDM. Ce concept concerne l'automatisme de l'opération de transformation.

au cours du cycle d'élaboration qui peut avoir une sémantique différente selon des utilisations : raffinement, optimisation, génération de code, etc.

La transformation du modèle est une opération qui consiste à en générer un ou plus. les modèles cibles en fonction de leur méta-modèle issu d'un ou de plusieurs modèles sources selon leur métaphore.

D'autre part, les méta-modèles source et cible.peut être identique dans certaines situations, nous parlons donc de transformation endogène, sinon cela transformation est qualifiée d'exogène.

La transformation du modèle consiste à appliquer un ensemble de règles de transformations exprimant la correspondance entre les entités du modèle source et celles du modèle cible.Les règles de transformation peuvent être exprimées à partir d'une fonçons déclarative, impérative ou hybride.Dans la spécification déclarative, les règles décrivent ce qu'un certain nombre d'éléments du modèle source devraient être développés.En contraste avec la spécification déclarative, la spécification impérative permet de décrire comment le résultat devrait être atteint par l'imposition d'une série d'actions que la machine doit effectuer.

En dernier lieu, la spécification hybride combine à la fois la spécification déclarative et la spécification impérative.[11]

Nous ferons connaissance avec des concepts clés associés à la transformation est représenté à la figure 2.3.

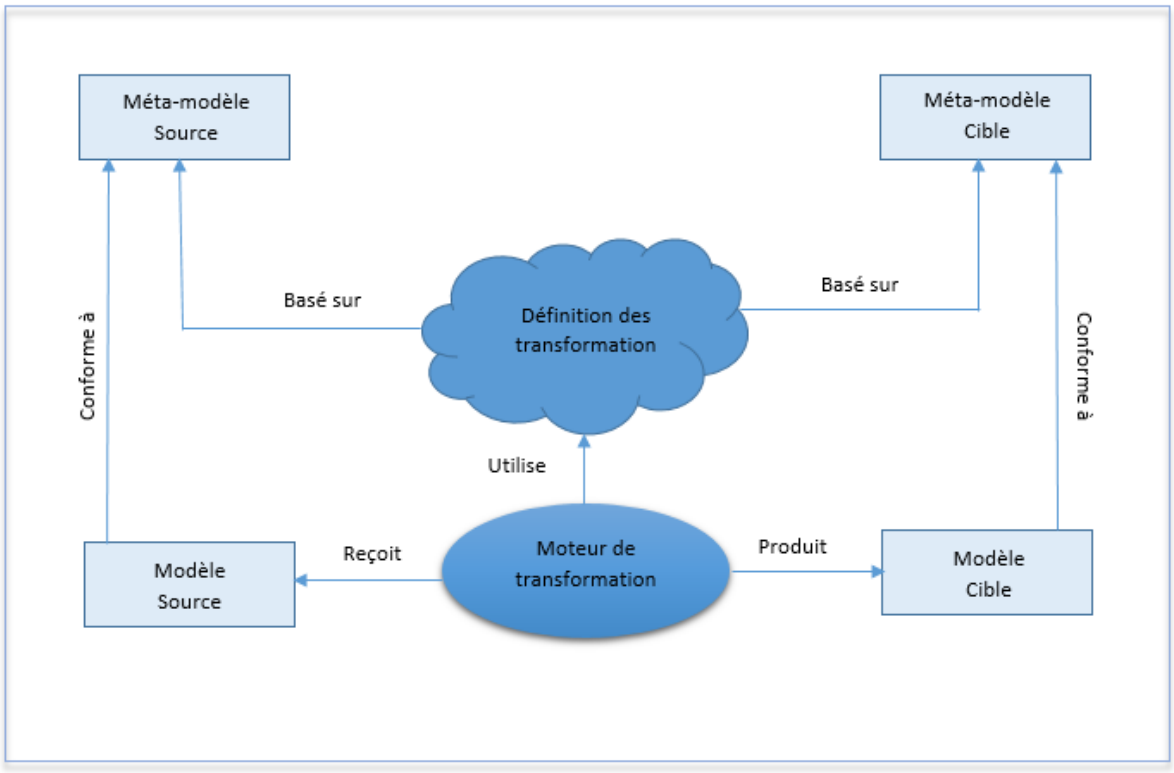

FIGURE 2.3 – Concepts et principe de base de la transformation de modèles[11].

2.7 Les type des transformations :

il existe trois types de transformations, nous expliquerons le concept de chaque type :

Les transformations verticales :

La source et la cible d'une transformation verticale sont définis à divers niveaux d'abstraction.Une transformation qui baisse le niveau d'abstraction est appelée un raffinement.Une transformation qui relève le niveau de l'abstraction est appelée une abstraction.[11]

Les transformations horizontales :

Une transformation horizontale changer la représentation. source tout en conservant le même niveau d'abstraction.Le changement peut être l'ajout, la modification, la suppression ou la restructuration d'informations.[11]

Les transformations obliques :

Une transformation oblique consiste en une transformation horizontale et une verticale.Ce type de traitement est particulièrement utilisé par la compilateurs, qui effectuent des optimisations du code source avant de générer le code exécutable.[11]

De manière orthogonale à cette catégorisation, selon Czarnek [15], il existe deux grandes classes de transformation de modèles : les transformations de type Modèle vers code qui sont aujourd'hui relativement matures et les transformations de type modèle vers modèle qui sont moins maîtrisées.

2.7.1 Transformation modèles vers modèles :

Depuis quelques années, ce genre de transformation connaît un taux important de travaux de chercheurs qui mettent au jour cette approche. Cette émergence est due aussi et surtout à l'apparition du MDA qui est la piste du MDE adoptée par le groupe OMG.

Généralement, le changement entre les modèles est de restructurer les informations sous une autre forme plus exploitable et plus formelle.De plus, il donne l'occasion de. vérifier et d'enlever l'ambigüité réside derrière la semi-formalité.Ce qui permet hautement réduire le taux d'erreur et veiller à la sécurité.Par contre, ce passage nous le permet passer d'un niveau d'abstraction à un autre, ce qui contribue au recul du niveau de perte d'information.[10]

Un certain nombre de méthodes ont été proposées pour la transformation de type modèle vers modèle : les approches par manipulation directe, les approches relationnelles, les approches basées sur les transformations de graphes, les approches dirigées par la structure et les approches hybrides.Nous expliquons ces méthodes dans la suivante :

A. Approches par manipulation directe :

Ces approches de transformation reposent sur une description interne des modèles source et cible, ainsi qu'un ensemble d'API à manipuler. Ceci est généralement mis en œuvre en tant que Frameworks de structuration orientée objet qui fournissent un minimum de concepts sous la forme de classes abstraites par exemple. L'utilisation de cette technique force le développeur à s'inquiéter de la mise en oeuvre ainsi que du séquençage des règles en utilisant un langage de programmation.[10]

B. Approches relationnelles :

Ces approches sont fondées sur des logiques mathématiques et ont recours à des relations mathématiques. Permettre de représenter les rapports entre les éléments composant le modèle source et ceux du modèle cible en adoptant une série de contraintes. En particulier, la programmation logique est appropriée pour ce type d'approche, dans laquelle les transformations construites sont généralement bidirectionnelles.[10]

C. Approches basées sur les transformations de graphes :

Le principe de ces approches apporte bien davantage au principe des approches relationnelles de l'utilisation des règles de transformation déclarative. Toutefois, les règles ne sont plus décrites pour les éléments simples mais pour les fragments de modèle (correspondance de modèle) : filtrage de modèle. Les patrons ou fragments de modèles sont exprimés soit dans leur syntaxe concrète, soit dans leur syntaxe abstraite. Un exemple d'outils qui supportent ce type de transformation est AToMPM, l'outil que nous utiliserons pour exprimer et mettre en œuvre nos règles de transformation.[10]

D. Approches basées sur la structure :

Dans ces approches, il y a deux étapes de transformation du modèle. En premier lieu, il s'agit de créer la structure hiérarchique du modèle cible. Le second consiste en un ajustement des attributs et des références de ce dernier. Le cadre de transformation du modèle en modèle fourni par Optimal constitue un exemple de ce type d'approche.[10]

E. Approches hybrides :

L'approche hybride consiste à combiner diverses techniques. En particulier, nous pouvons trouver des méthodes utilisant à la fois des règles obligatoires et déclaratives. Un exemple de langues reposant sur cette approche est la langue ATL. (ATLAS Transformation Language).[10]

2.7.2 Transformation modèles vers code :

Cette approche peut être considérée comme un cas spécial de transformation d'un modèle à l'autre. La transformation des modèles en code (ou texte) a pour but de produire du code dans un langage de programmation particulier (Java, C++, XML, HTML, etc.) à partir d'un modèle source particulier. En général, il existe deux approches pour transformer un modèle en un code : approche basée sur les visiteurs (Visitor-based approach) et approche basée sur les motifs (Template-based approach).[10]

A. Approches basées sur des visiteurs :

Elles consistent à fournir des mécanismes permettant de passer en revue la représentation interne des modèles sources, ajout d'éléments (mécanismes visiteurs) en même temps sur les modèles parcourus en réduisant la différence sémantique entre le modèle et le langage de programmation cible, et fournit finalement un code écrit dans un flux de sortie (fichier, écran, etc.). Jamna en est un exemple. Qui est un cadre axé sur les objets offrant une gamme de classes pour représenter les modèles UML, une API pour manipuler les gabarits et les mécanismes de visiteur (parfois appelé auteur de code) pour générer le code.

B. Approches basées sur les patrons :

Un modèle est généralement composé d'un texte cible contenant des splices de méta-code qui donnent accès à l'information dans le modèle source, sélectionnez text (code) et développez itérativement. Voici quelques exemples de technologies liées à cette approche : FPL, XFramer et ANGIE.

2.8 Phase de Méta-modélisation :

AToMPM (A Tool for Multi-Paradigm Modeling) [11], le successeur d'AToM3 . AToMPM est un cadre open-source pour la conception d'environnements DSML, la réalisation de transformations de modèles, la manipulation et la gestion de modèles. d'effectuer des transformations de modèles, de manipuler et de gérer des modèles. Il fonctionne entièrement sur le web, ce qui le rend indépendant de tout système d'exploitation, plate-forme ou appareil sur lequel il s'exécute. AToMPM suit la philosophie suivante : modéliser tout explicitement, au bon niveau d'abstraction, en utilisant le(s) formalisme(s) et le(s) processus le(s) plus approprié(s). formalisme(s) et processus les plus appropriés, en étant complètement modélisé par lui-même (c.-à-d., bootstrapped).

2.8.1 Méta-modele de diagramme d'états-transitions :

Dans AToMPM, la syntaxe abstraite d'un langage spécifique à un domaine est définie à l'aide d'un métamodèle qui décrit de manière formelle et abstraite la structure syntaxique du langage. Le métamodèle définit les concepts de base, les relations entre eux et les règles qui régissent leur utilisation. Ainsi, la syntaxe abstraite dans AToMPM se réfère à la structure conceptuelle sous-jacente du langage qui permet de décrire et de modéliser les concepts et les relations spécifiques à un domaine. Cette structure est formalisée dans le métamodèle et sert de fondement pour la définition de la syntaxe concrète et de la sémantique du langage.

Selon ce méta-modèle, un diagramme d'états-transitions peut contenir des états(Etat simple,Etat composite)et des transitions (événements,condition et actions). Un état a un nom (chaîne de caractères pour l'identification), des actions d'entrée et des actions de sortie. Les états composites, contenant des sous-états, qui peuvent être séquentiels ou concurrents. Les transitions, qui sont désignées par des arcs standards, définissent un changement d'un état à un état successeur.[10] Une transition peut contenir un événement avec une liste d'actions .L'illustration 2.4 présente le méta-modèle créé dans AToMPM pour le diagramme d'état-transitions.(C'est la syntaxe abstraite).

Le méta-modèle du diagramme d'états-transitions est composé de six classes et de douze associations, comme suit :

Classe "Stat-chart" :

Il correspond un diagramme d'états-transitions et possède un attribut clé name-S-C de type string.

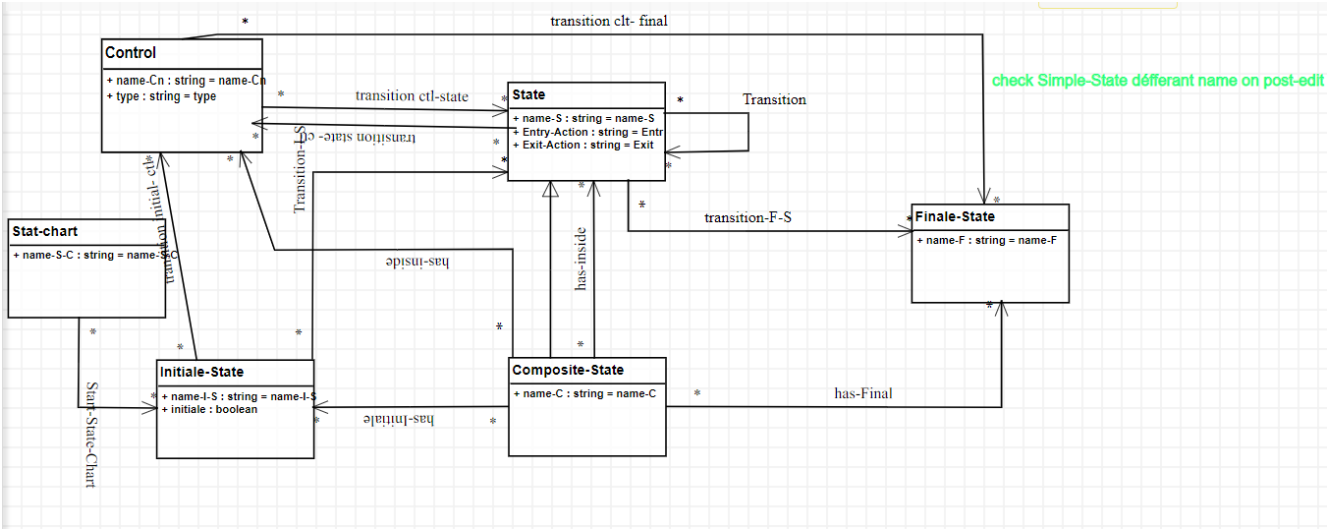

FIGURE 2.4 – Méta-modèle du diagramme d'états-transitions.

Classe "State" :

Cette classe décrit les états. Elle possède trois attributs : un attribut clé name-S de type String indiquant le nom de l'état ; et deux attributs Entry-Action et Exit-Action de type String indiquant respectivement les actions exécutées à l'entrée et à la sortie de l'état.

Classe "Composite-State" :

Cette classe représente les états composites. Elle hérite de Simple State tous ses attributs, les multiplicités et les associations.

Classe "Initiale-State" :

Cette classe représente l'état initial d'un diagramme d'états transitions, l'état initial d'un état composite séquentiel ainsi que les états initiaux d'un état composite concurrent.Elle possède une seul attribut clé name-I-S de type String indiquant le nom de l'état.

Classe "Finale-State" :

Cette classe représente l'état final d'un diagramme d'états transitions.Elle possède une seul attribut clé name-F de type String indiquant le nom de l'état.

Classe "Controle" :

Cette classe représente point de choix,fork,join..etc d'un diagramme d'états transitions.Elle possède deux attributs clé name-Cn de type String et type de type string pour connaître le type de cas.

Association "Transition-I-S" :

Elle permet de relier l'état initial du diagramme d'états-transitions avec un état simple ou composite.

Association "has-Final" , "transition-F-S" et "transition-clt-final :

Cette association permet de relier un état simple ou composite ou controle avec un état final.

Association "Start-State-Chart" :

Cette association relie un diagramme d'états-transitions avec son état initial.

Association "Transition" :

Cette association représente la transition d'un état source à un état destination. Il contient troix attributs : event de type string ,condition de type string action de type String.

Association "has-inside" :

Cette association exprime la notion de l'hiérarchie des états, c'est-à-dire les états qui sont à l'intérieur d'un état composite. cette association relie un état composite et un autre état (qui peut être aussi composite).

Association "transition initial-ctl" :

Cette association relie un état initiale avec point de choix ou fork ou join ou ...etc.

Association "transition state-ctl" et "transition ctl-state" :

Cette association relie les point de choix ou les fork ou les join ou ...etc avec un état simple.

Association "has-Initial" :

Cette association exprime la notion de hiérarchie entre un état composite et son état initial.

Association "has-Final" :

Cette association exprime la notion de hiérarchie entre un état composite et son état final.

ensuite la syntaxe concrète pour le diagramme d'état-transitions dans AToMPM présente dans la figure 2.5
Dans AToMPM, la syntaxe concrète d'un langage spécifique à un domaine est définie visuellement à l'aide d'une notation graphique dans l'environnement de modélisation visuelle. Cette notation graphique utilise des boîtes, des flèches et des icônes pour représenter les éléments de syntaxe abstraits du langage, ainsi que des règles de mise en page et de style pour assurer une présentation cohérente des modèles créés avec le langage. En somme, la syntaxe concrète dans AToMPM se réfère à la représentation visuelle des éléments syntaxiques abstraits d'un langage spécifique à un domaine dans l'environnement de modélisation visuelle de AToMPM.

Il existe deux classes principales dans AToMPM : Icon et Link. La première est utilisée pour contenir les éléments visuels qui constituent la visualisation d'une instance de classe, tandis que la seconde permet de définir la visualisation d'une instance d'association sous forme d'une flèche.

La syntaxe concrète du diagramme d'états-transition est composée de six instances(Icon), dans chaque instances la représentation de chaque classe dans la syntaxe abstraite et des associations, comme suit :(figure 2.5)

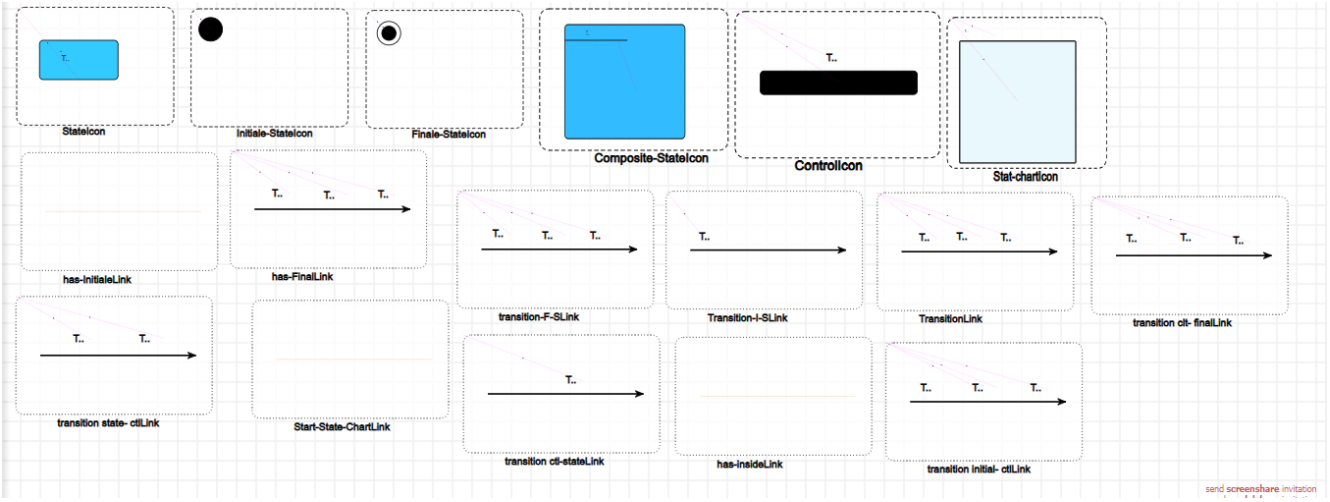

FIGURE 2.5 – La syntaxe concrète du diagramme d'états-transitions.

Instance (Icon) "ControlIcon" :

Cette instance est constitué certaine type de cntrole : état historique, point de choix, fork, join, point de jonction et point de connexion.par example dans le mappers de point de choix créer le code suivant :
/* specify code that evaluates to an associative array of the form ...,csattr :val,... */
var style =
"stroke" : "000000",
"stroke-dasharray" : "",
"fill" : "333333",
"fill-opacity" : 1,
"font-size" : "20px",
"stroke-width" : 3,
"arrow-start" : "none",
"arrow-end" : "none" ;
if (getAttr("PointChoix"))
style["fill"]= "fffff" ;

('style' :style)

Instance (Icon) "StateIcon" :

Ce corps est constitué d'une représentation graphique de l'état ,où il a la forme d'un rectangle à l'intérieur de ce rectangle se trouve le nom de l'état simple (ce nom est le même que celui trouvé à la classe à la syntaxe abstraite) de type string.
Nous écrirons ce code dans "mapper" et "paser" , comme le montre la figure 2.6 :

Méme chose pour les autre instance (Icon) chaque instance est constitué des représentation graphiques.

Instance (Link) "hase-FinaleLink" :

Ce corps est constitué une représentation graphique de lien entre les instances (Icon),où il a la forme d'un flèche ou une ligne sans destination(elle doit être de couleur claire ou bien foncée) par example l'instance (Link) has-insideLink et has-FinaleLink, les deux sans représenter dans la figure 2.7 .

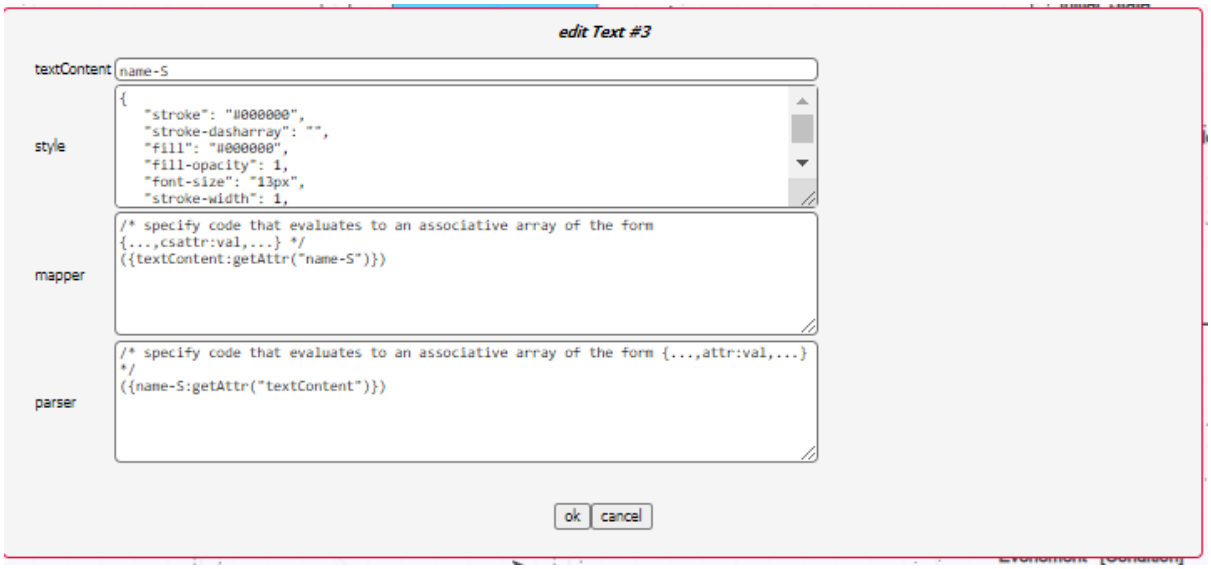

FIGURE 2.6 – Le code "mapper" et "parers" de "T.." (etat simple).

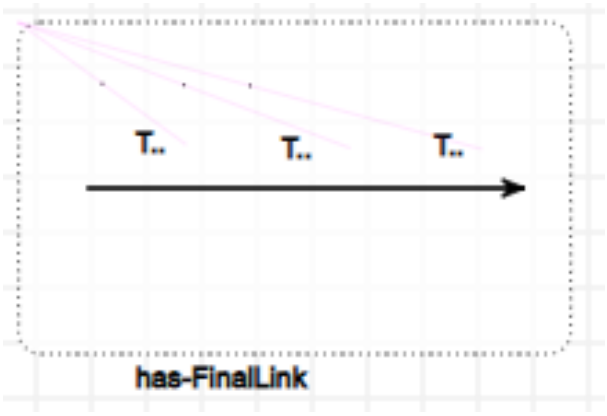

FIGURE 2.7 – L'instance(Linke) de lien hase-FinaleLink.

2.8.2 Méta-modele de nets within nets :

Selon ce méta-modèle, une nets within nets peut contenir des place(placeON(Object Net),placeSN(Sytem Net)et des transitions (transitionON,transitionSN).Dans chaque place il y a une jeton (jetonSN ou joutinON) et les jetonSN au sein des référance (RdpON) .L'illustration 2.8 présente le méta-modèle créé dans AToMPM pour Nets within nets.(C'est la syntaxe abstraite).

Le méta-modèle de nets within nets est composé de huit classes et de neuf associations, comme suit :

Classe "Rdp-SN" :

Cette classe présentent le cadre à l'intérieur duquel nous dessinons un nets within nets. Elle possède une seul attribut clé name-S de type String indiquant le nom de cadre (de nets within nets).

Classe "Place-SN" :

Cette classe représente la place system net d'un nets within nets.Elle possède une seul attribut clé name-P-SN de type String indiquant le nom de la place.

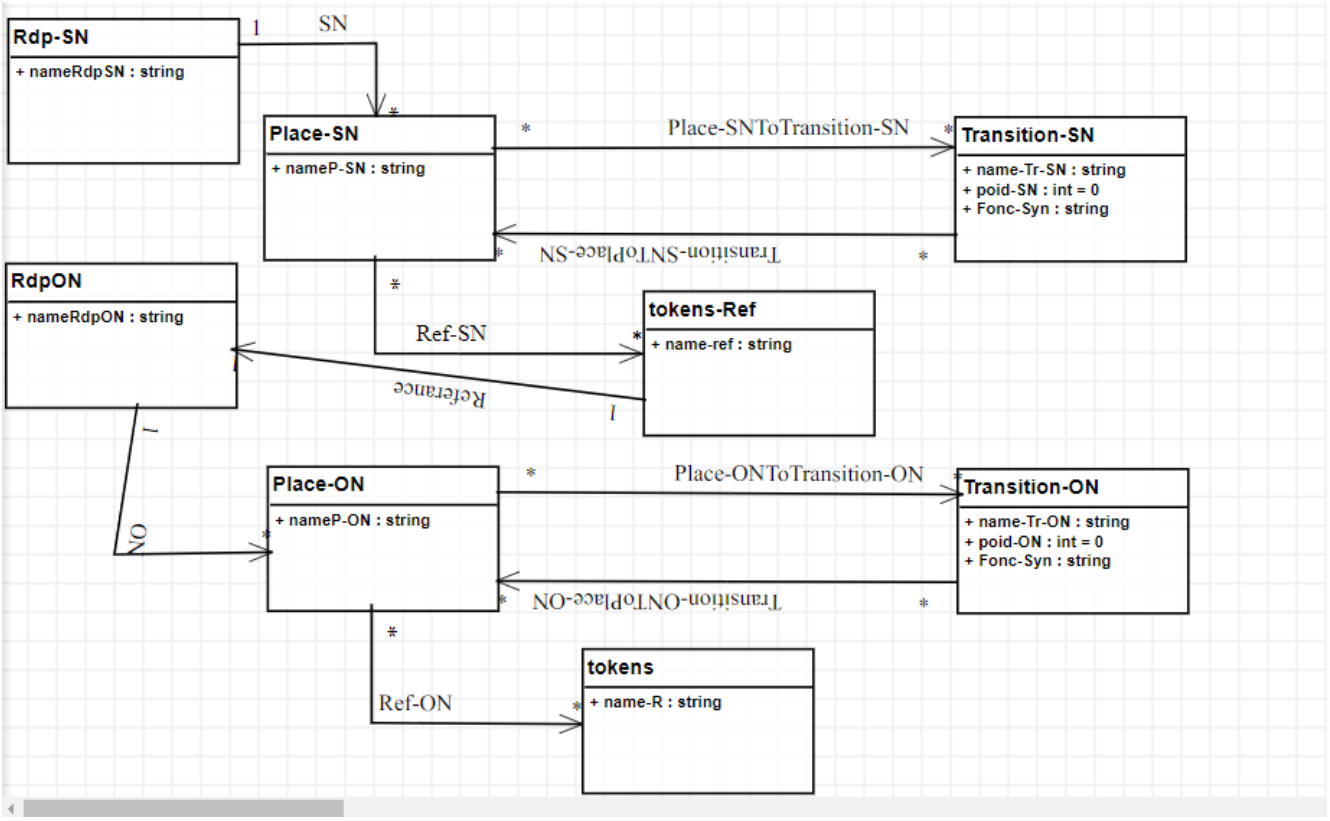

FIGURE 2.8 – Le méta-modéle de nets within nets".

Classe "Transition-SN" :

cette classe présentent la transtion entre deux placeSN. Elle possède trois attribut le clé c'est le name-Tr-SN de type string indiquant le nom de la transition,le poid-SN c'est le poid des jeton dans la transition de type int et Fonc-syn c'est la fonction syncronisation entre les transition de type string.

Classe "tokens-Ref" :

Cette classe représente le jeton system net ce qui est à l'intérieur de place-SN d'un nets within nets.Elle possède une seul attribut clé name-ref de type String indiquant le nom de jeton.

Classe "Rdp-ON" :

Cette classe présentent la référance d'un jeton ce qui est à l'intérieur de la place-SN . Elle possède une seul attribut clé nameRdpON de type String indiquant le nom de référance .

Classe "Place-ON" :

Cette classe représente la place object net d'un nets within nets.Elle possède une seul attribut clé name-P-ON de type String indiquant le nom de la place.cette place ce qui est à l'intérieur de "Rdp-ON".

Classe "Transition-ON" :

cette classe présentent la transtion entre deux placeON. Elle possède trois attribut le clé c'est le name-Tr-ON de type string indiquant le nom de la transition,le poid-0N c'est le poid des jeton dans la transition de type int et Fonc-syn c'est la fonction syncronisation entre les transition de type string.

Classe "tokens" :

Cette classe représente le jeton object net ce qui est à l'intérieur de place-ON d'un nets within nets.Elle possède une seul attribut clé name-R de type String indiquant le nom de jeton.

association "SN"

Elle permet de relier le cadre du nets within nets avec un place-SN,une cadre contient plusieurs plase-SN.

association "Place-SNToTransition-SN"

cette association pour relier une place-SN avec transition-SN.

association "Transition-SNToPlace-SN"

cette association pour relier une transition-SN avec place-SN.

association "Ref-SN"

cette association pour relier une place-SN avec les jeton .

association "Referance"

Cette association pour relier un jeton contient dans la place-SN avec la référence de chaque jeton.

association "Place-ONToTransition-ON"

cette association pour relier une place-ON avec transition-ON.

association "Transition-ONToPlace-ON"

cette association pour relier une transition-ON avec place-ON.

association "Ref-ON"

Cette association pour relier un jeton contient dans la place-SN avec la référence de chaque jeton.

association "ON"

Elle permet de relier la référance avec un ou plusieur place-ON,une référance contient plusieurs plase-ON et transion-0N. ensuite la syntaxe concrète pour Nets Within Nets dans AToMPM présente dans la figure 2.9

Instance (Icon) "Place-SN" :

Ce corps est constitué d'une représentation graphique de place-SN ,où il a la forme d'un cercle à l'intérieur de ce cercle se trouve le nom de la place (ce nom est le même que celui trouvé à la classe à la syntaxe abstraite) de type string.
Le reste des représentations graphiques se fera de la même manière, chaque classe a son propre représentation.

La syntaxe concrète du Nets Within Nets est composée de huit instances(Icon), dans chaque instances la représentation de chaque classe dans la syntaxe abstraite et des associations, ceci est montré dans la figure 2.9.
Nous expliquerons les type de liens trouvé dans la syntaxe concrète du diagramme de Nets Within Nets.

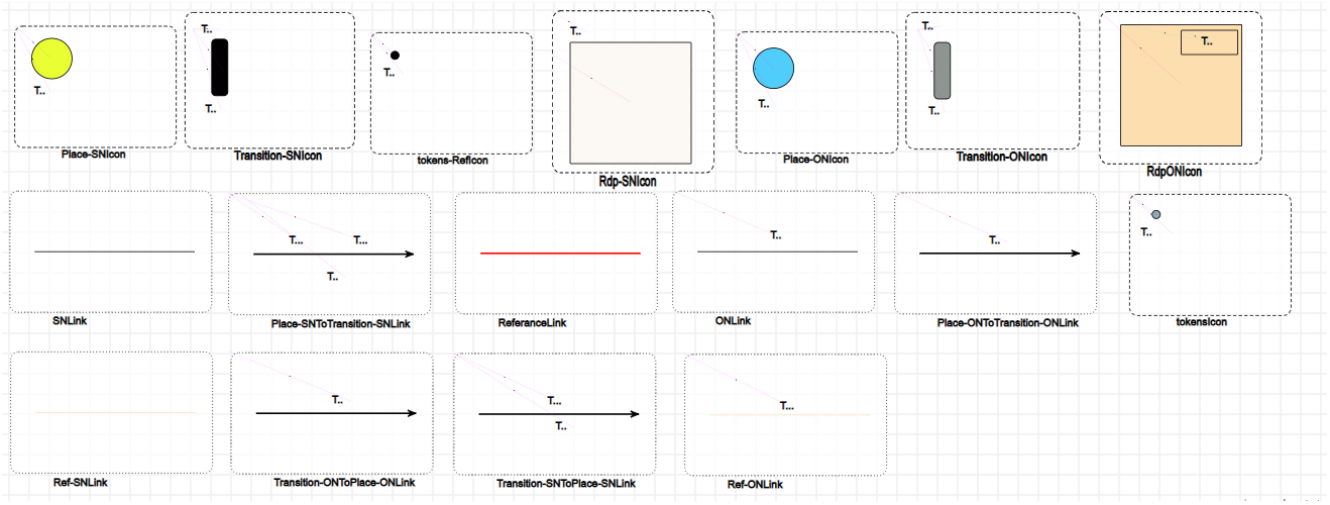

FIGURE 2.9 – La syntaxe concrète du diagramme de Nets Within Nets.

Par example la figure 2.10 il contient quatre type de lien :

1. Le lien "ONLink" : Ce lien se présente sous la forme d'une ligne noire reliant le cadre de référace et la ou les place-ON.

2.Le lien "ReferanceLink" : Ce lien se présente sous la forme d'une ligne rouge claire reliant le jeton-SN a sa référance.

3.Le lien "Place-SNToTansition-SN" : Ce lien se présente sous la forme d'une flèche avec une direction de place-SN a transition-SN et un chaine de caractére spécifique écrit au-dessus de la flèche.

4.Le lien "Ref-ONLink" : Ce lien est presque invisible et est destiné à relier la place-ON à jeton-ON de manière ordonnée.

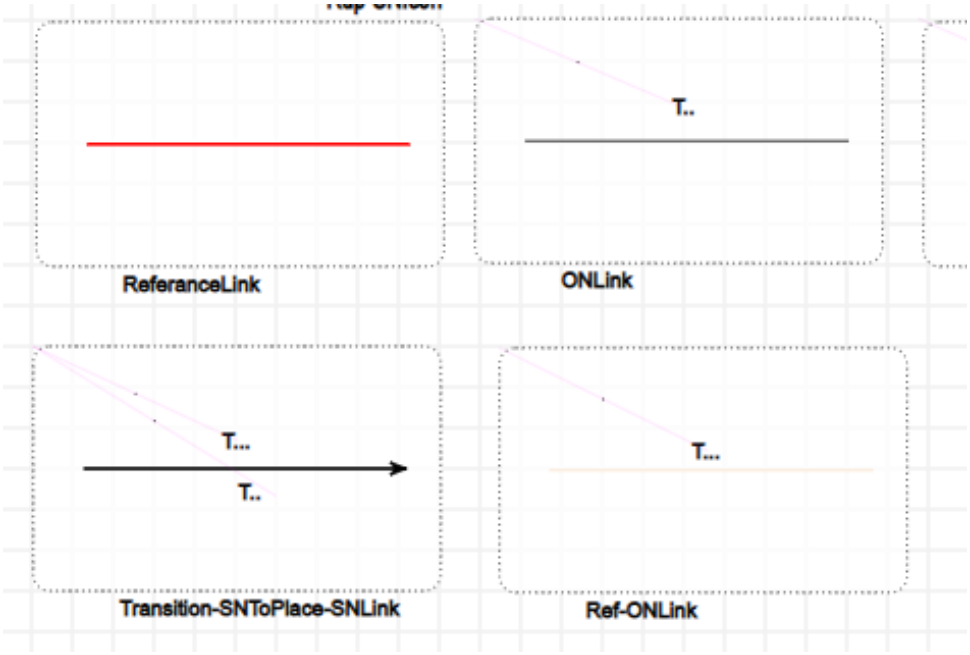

FIGURE 2.10 – Quelques instance(Link).

Conclusion :

Dans ce chapitre, nous avons commencé dans un premier temps par présenter sommairement le méta-modélisation. Nous avons focalisé plus particulièrement sur la transformation de modèles, l'une des techniques prometteuses d'IDM. Nous avons présenté une énumération de quelques types de transformation de modèles, suivie d'une classification de certaines approches de transformation. Dans la deuxième section de ce chapitre, nous avons mis l'accent sur la phase de méta-modélisation, en abordant les points suivants :

1.Méta-modele de diagramme d'états-transitions .
2.Méta-modèle des nets within nets.

En outre, nous avons présenté d'une façon assez exhaustive AToMPM, l'outil de transformation de graphe qui est utiliser pour générer des nets within nets à partir des diagramme d'etatst-ransitions UML.

Chapitre 3

Transformation des graphes :

Introduction :

La transformation d'un diagramme d'états-transitions UML en un modèle Nets Within Nets est une étape cruciale à cause de la différence dans la syntaxe et la sémantique des deux formalismes. Cependant, cette transformation doit être effectuée en suivant des règles bien définies afin de garantir la précision et la cohérence du modèle Nets Within Nets par rapport au diagramme d'états-transitions original.

Dans cette optique, ce chapitre présente les règles de transformation de diagramme d'états-transitions vers les Nets Within Nets en utilisant AToMPM, et en mettant l'accent sur les éléments clés tels que les états, les transitions et les actions.
Nos règles de transformation se divisent en trois catégories distinctes :
La première catégorie concerne le transformation des objets dans le diagramme d'états-transitions tels que les états simples, les états composites, etc. en d'autres objets dans les nets within nets incluant placeOn, placeSn, etc.
La deuxième catégorie englobe la création des arcs dans les nets within nets. La dernière catégorie englobe la suppression des objets du diagramme d'états-transitions UML.

Nous avons conclu cette grammaire de graphe comme suite :
es états peuvent être convertis en places dans le nouveau modèle. Cela se fait généralement en attribuant chaque état possible dans le diagramme comme un composant ou un élément distinct dans le nouveau modèle, représenté par des places. De cette manière, vous pouvez convertir les différents états en places dans la nouvelle structure en fonction des besoins et des exigences de la transformation.
un "état composite" peut être converti en "place" en subdivisant l'état composite en sous-états indépendants, puis en représentant chaque sous-état en tant que "place" dans le nouveau modèle. Cela signifie que l'état composite contient généralement des sous-états ou des subdivisions, et chaque subdivision peut être représentée comme une "place" distincte à l'intérieur du modèle global "nets within nets".
un "arc" peut être converti en "transition" en reliant les différentes états avec des "arcs" dans le diagramme d'origine, puis en transformant ces connexions en "transitions" dans le nouveau modèle. Chaque lien entre deux états par un "arc" est représenté dans le modèle final par une "transition" qui reflète le passage d'un état à un autre. Cette démarche permet de préserver les relations et les transitions entre les états d'origine dans le nouveau modèle.

3.1 La première catégorie

3.1.1 Les régles 1-4 :

La première règle permet de transformer d'un état simple dans le diagramme d'états-transitions UML en une PlaceON dans le réseau de Petri nets within nets (figure 3.1). La deuxième règle est utilisée pour transformer un état final en une transitionON (figure 3.2). La troisième règle permet la transformation de l'état initial vers un jeton initial (figure 3.4). Enfin, la quatrième règle est utilisée pour transformer un état composite en une PlaceSN (figure 3.3).

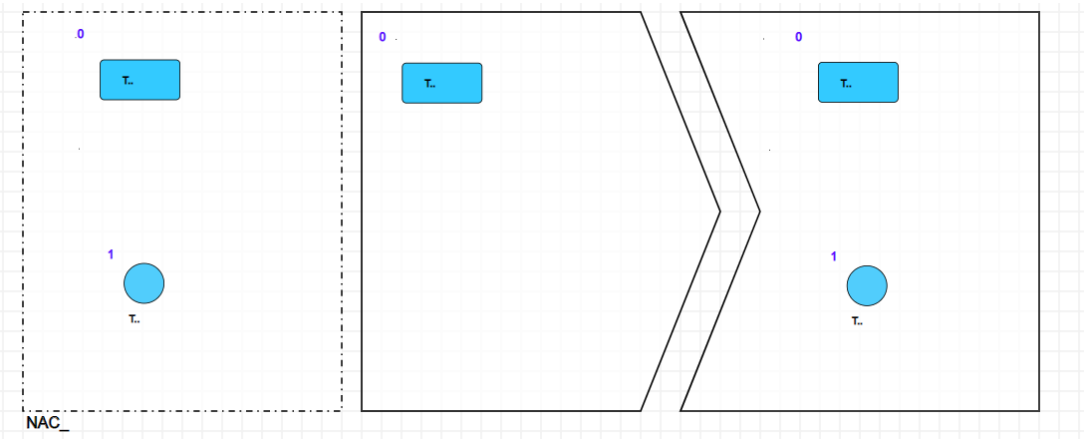

FIGURE 3.1 – Représentation graphique de la première régle 'Etat2placeON'.

Le code NAC,LHS et RSH de la règle "Eta2placeON" :

Dans la condition de NAC :

result = (getAttr('name-S','0') == getAttr('nameP-ON','1'))

dans la condition de LHS :

result = True

et dans l'action de RHS :

setAttr('nameP-ON',getAttr('name-S','0'),'1')

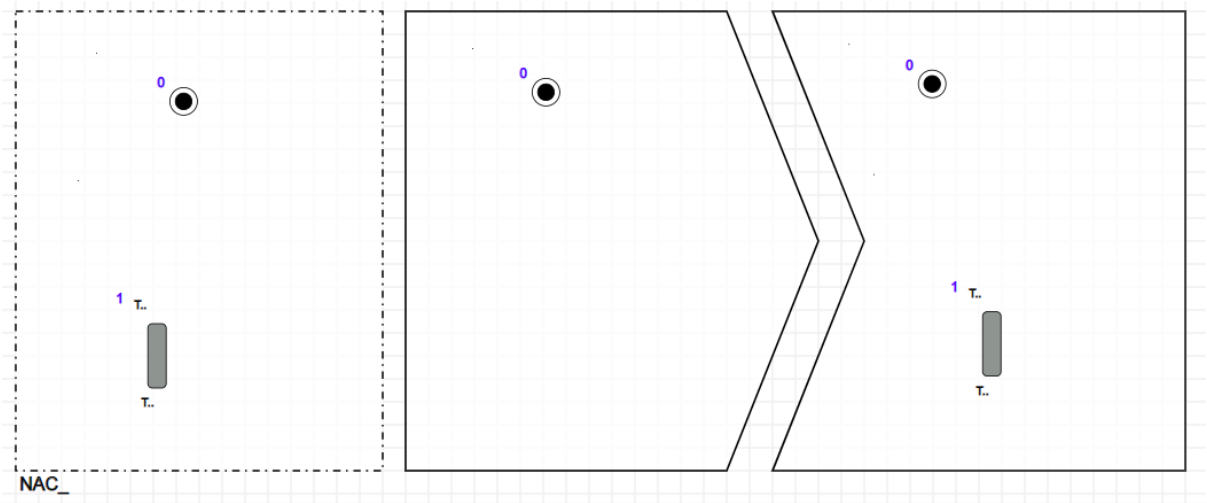

FIGURE 3.2 – Représentation graphique de la douxième régle 'etatFinale2TransionON'.

Le code NAC,LHS et RSH de la règle "etatFinale2TransitionON" :

Dans la condition de NAC :

result = (getAttr('name-F','0') == getAttr('name-Tr-ON','1'))

dans la condition de LHS :

result = True

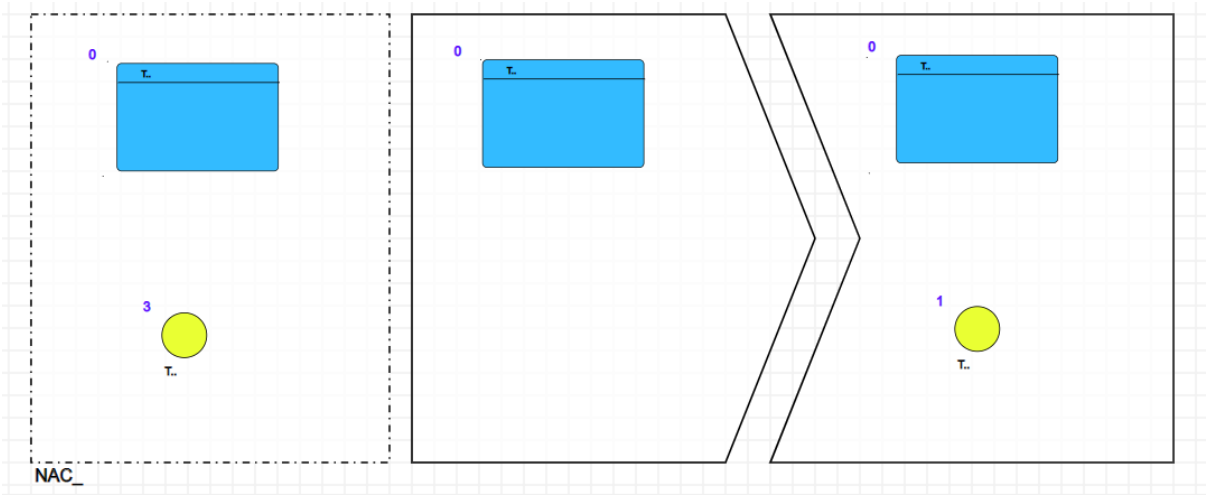

FIGURE 3.3 – Représentation graphique de la troisième régle 'etatCompsite2placeSN'.

et dans l'action de RHS :

setAttr('name-Tr-ON',getAttr('name-F','0'),'1')

Le code NAC,LHS et RSH de la règle "etatComposite2PlaceSN" :

Dans la condition de NAC :

result = (getAttr('name-C','0') == getAttr('nameP-SN','1'))

dans la condition de LHS :

result = True

et dans l'action de RHS :

setAttr('nameP-SN',getAttr('name-C','0'),'1')

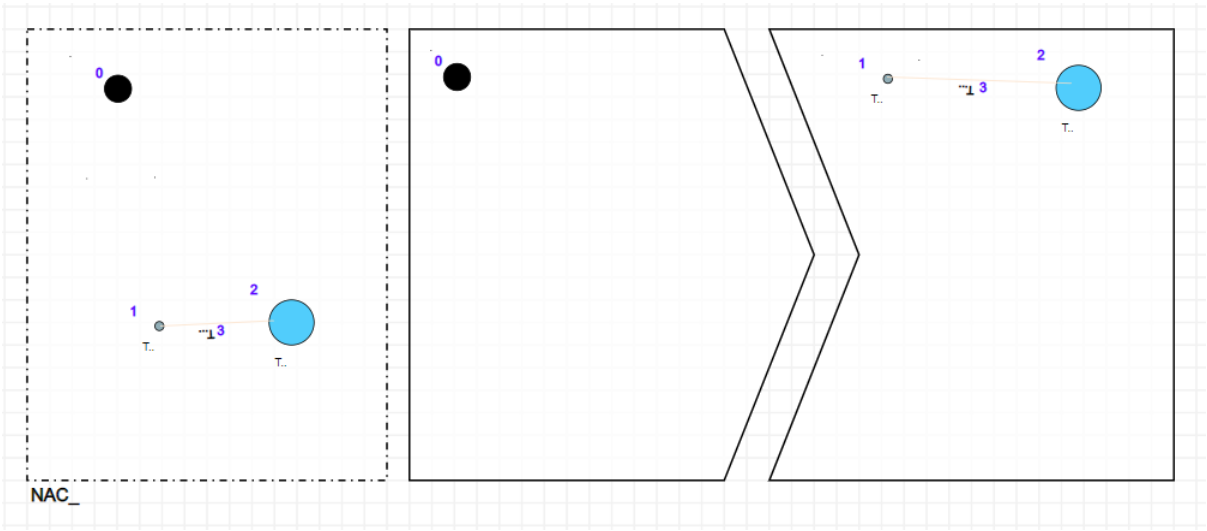

FIGURE 3.4 – Représentation graphique de la quatrième régle 'etatInitial2jetonOn'.

3.2 La deuxième catégorie

3.2.1 Les régles "règles 5-11" :

La règle "5" consiste à créer une nouvelle transitionON entre deux places placeON. Tout d'abord, elle collecte les événements, les conditions et les actions associés aux transitions entre les états du diagramme d'états-transitions UML. Ensuite, ces événements, conditions et actions sont intégrés dans les noms des transitionON sous la forme suivante : évènement[condition] / action dans les Nets within nets. Finalement, on supprime l'arc entre les deux états du diagramme d'états-transitions (figure 3.5).

De la même manière de la règle 5, la règle "6" consiste à créer une nouvelle transitionON entre deux places placeON dans les nets within nets s'il existe une transition entre un état simple et un état final correspondants dans le diagramme d'états-transitions UML (figure 3.6).

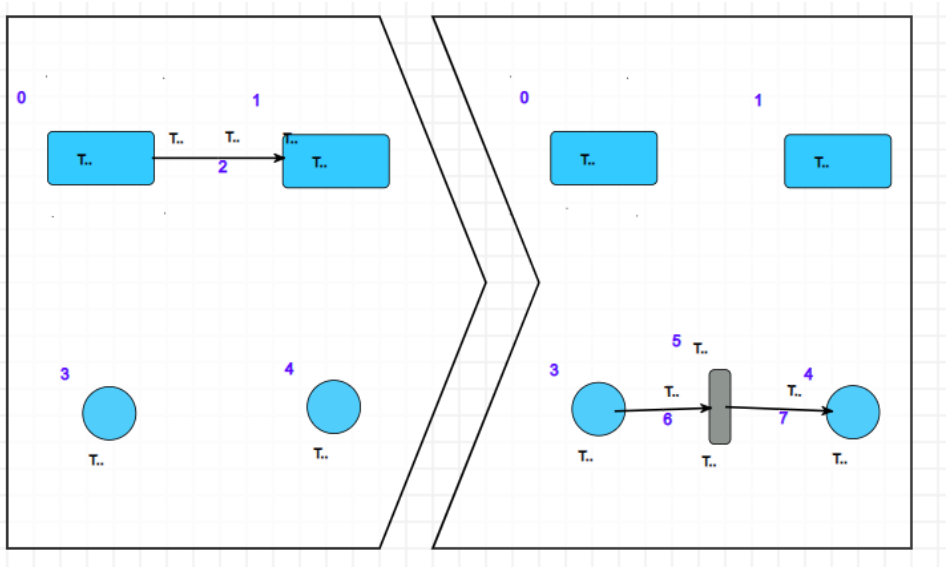

FIGURE 3.5 – Représentation graphique de la règle 'arc2transitionON'.

Le code LHS de cette règle :

result = ((getAttr('name-S','0') == getAttr('nameP-ON','3')) and (getAttr('name-S','1') == getAttr('nameP-ON','4')))

Le code RHS :

setAttr('name-Tr-ON', ((getAttr('Evenement', '2'))+"["+(getAttr('Condition', '2'))+"] /"+(getAttr('Action', '2'))), '5')

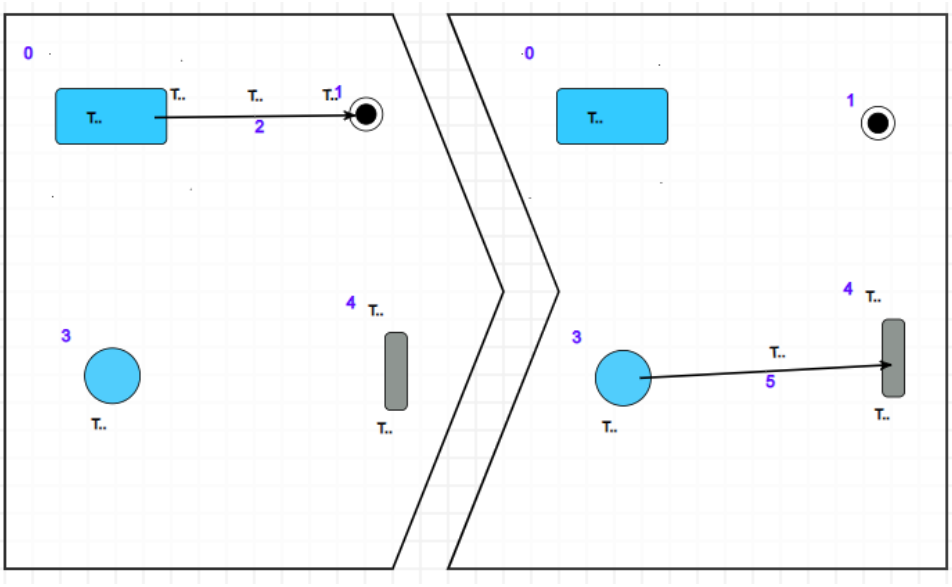

FIGURE 3.6 – Représentation graphique de la règle 'etatFin2transitionFinON'.

Le code LHS de cette règle :

result = (getAttr('name-S','0') == getAttr('nameP-ON','3'))

Le code RHS :

setAttr('name-Tr-ON', ((getAttr('Evenement', '2'))+"["+(getAttr('Condition', '2'))+"]
/"+(getAttr('Action', '2'))), '4')

De la même manière de la règle 5, la règle "7" consiste à créer une nouvelle transitionON entre deux places placeON dans les nets within nets s'il existe une transition entre un état simple et un point de choix correspondants dans le diagramme d'états-transitions UML (figure 3.7).

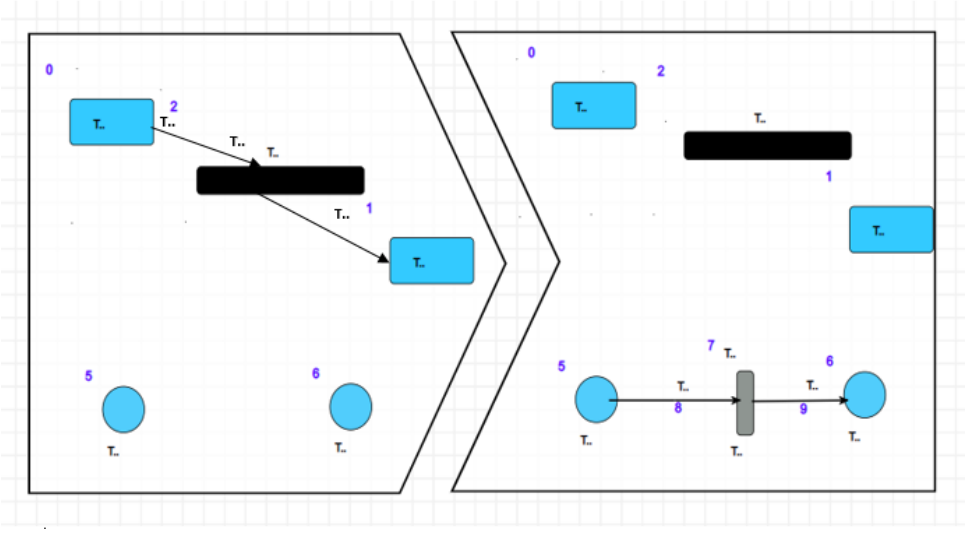

FIGURE 3.7 – Représentation graphique de la règle 'PointDeChoix2transitionOn'.

De la même manière de la règle 5, la règle "8" consiste à créer une nouvelle transitionON entre deux places placeON dans les nets within nets s'il existe une transition entre un état initial et un état simple correspondants dans le diagramme d'états-transitions UML (figure 3.8).

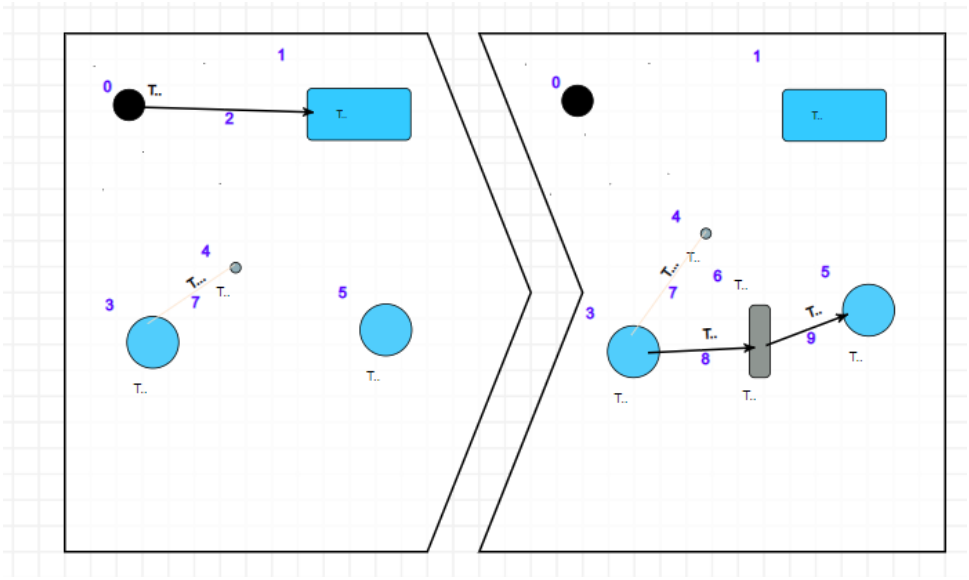

FIGURE 3.8 – Représentation graphique de la règle 'arcInitial2transition'.

La règle "9" consiste à génère une nouvelle transition " transitionON" entre deux places "placeON" dans les nets within nets qui correspondent deux états de diagramme d'états-transitions UML. Elle recueilli les événements, les conditions et les actions associés aux arcs entre deux états du diagramme d'états-transition UML. Ces événements, conditions et actions sont intégrés dans les noms des transitions "transitionON" sous la forme suivante : événement[condition] / action dans les Nets within nets. Les deux places "placeON" doivent avoir la même référence. Finalement, elle supprime tous les arcs entre les deux états (voir la figure 3.9).

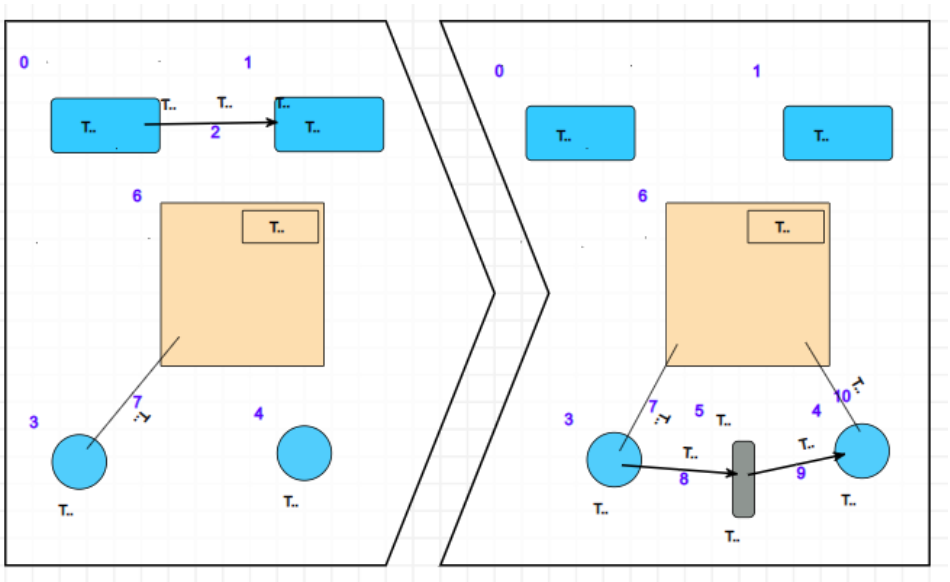

FIGURE 3.9 – Représentation graphique de la règle 'arc2tans'.

La règle "10" est présentée dans la figure 3.10.

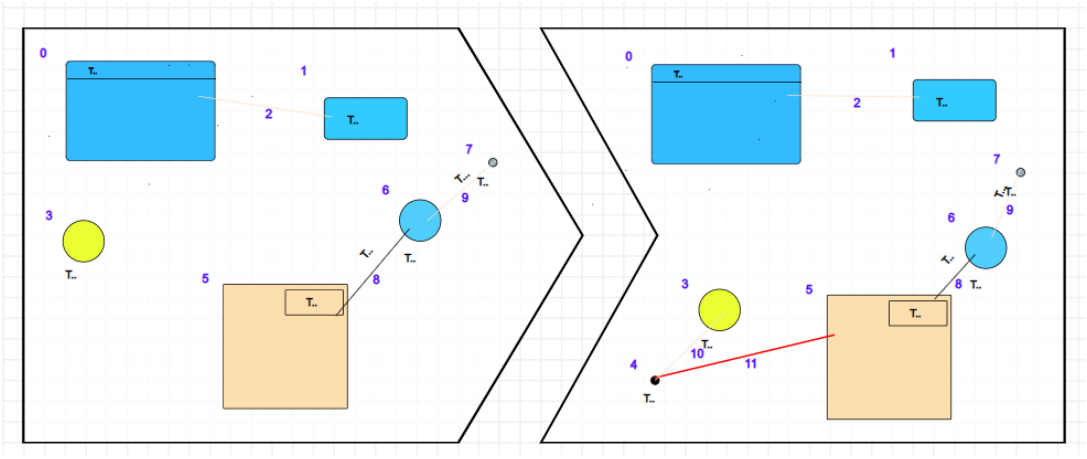

FIGURE 3.10 – Représentation graphique de la règle 'rule10'.

3.3 La troixième catégorie

3.3.1 Les régles "règles 11-15" :

Ces règles permet de supprimer les éléments de diagramme d'états transition (etat initiale, etat finale, etat, etat composite et controle)(figure 3.11).

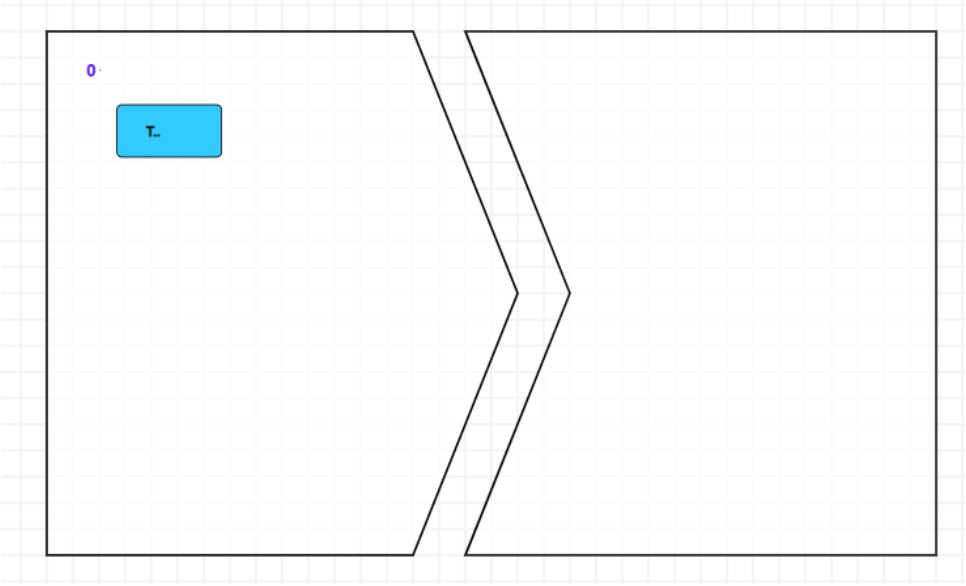

FIGURE 3.11 – Représentation graphique d'une règle par example 'DelateEtat'.

Le code LHS de cette règle :

result = True

Le code RHS :

pass

Conclusion :

En conclusion, ce chapitre a présenté une approche puissante pour transformer les diagrammes d'états-transitions UML en nets within nets grâce à l'outil AToMPM. Cette approche offre une solution automatisée et efficace pour la vérification des propriétés dynamiques des systèmes complexes en combinant les notations graphiques conviviales (UML) avec les notations formelles analytiques (nets within nets).

Les règles de transformation des graphes utilisées dans AToMPM ont été essentielles pour réaliser cette conversion avec précision et cohérence. Grâce à la méta-modélisation et à la définition de règles appropriées, nous avons pu capturer les éléments essentiels des diagrammes d'états-transitions et les représenter de manière équivalente dans le formalisme des nets within nets.

L'outil AToMPM s'est avéré être une plateforme polyvalente et flexible, offrant des capacités de transformation de graphes puissantes et adaptées à nos besoins. Sa facilité d'utilisation et sa capacité à gérer des modèles complexes ont grandement contribué à la réussite de notre approche de transformation.

Cependant, ce travail n'est qu'un premier pas vers l'utilisation de l'outil AToMPM pour la transformation de modèles. Dans le futur, il serait intéressant d'explorer davantage les fonctionnalités avancées d'AToMPM pour gérer des modèles plus vastes et complexes. De plus, il serait pertinent de développer des règles de transformation supplémentaires pour prendre en charge d'autres types de diagrammes UML et ainsi étendre l'applicabilité de cette approche a un éventail plus large de systèmes.

En définitive, cette recherche ouvre de nouvelles perspectives pour l'analyse et la vérification des systèmes complexes en combinant de manière synergique les avantages des notations graphiques et formelles. L'outil AToMPM s'avère être un atout prometteur pour la transformation de modèles, et ses développements futurs peuvent contribuer de manière significative à l'amélioration de la vérification et de l'analyse des systèmes complexes.

Chapitre 4

Étude de cas :

Introduction :

Dans ce chapitre, nous illustrons notre approche et son outil de support que nous avons implémenté. Ces éléments ont été présentés dans le chapitre précédent et seront démontrés à travers deux études de cas, chacune étant associée à une ressource différente [16,17].
Dans chaque étude de cas nous allons passer par les deux étapes suivantes :

1. La modélisation du comportement de chaque système en utilisant les diagrammes d'états transitions UML.

2. Application des règles de grammaires de graphes, développes dans le chapitre 3, sur les deux exemples en utilisant notre outil de transformation de graphe

L'objectif de ces études est de montrer en particulier la capacité de notre approche et de son outil à générer efficacement des nets within nets à partir des diagrammes d'états-transitions UML.

4.1 Étude de cas 1 :protocole de demande d'horaire

Dans cette section, nous examinons l'interaction d'un protocole en utilisant la méthode des statecharts propositionnels [16]. Cette méthode se divise en au moins trois perspectives, comprenant une vue globale où le protocole est défini de manière abstraite, ainsi que des perspectives individuelles correspondant à chaque rôle participant. Dans le cas du protocole de demande de planification, une vue globale possible est illustrée dans la Figure 4.1. Le protocole est représenté par un état de type "ou" qui comporte deux sous états de type "ou", l'un pour regrouper tous les états différents représentant l'exécution du protocole (dans notre exemple, l'état de demande de planification en attente) et l'autre pour englober les états de conclusion possibles (l'état de demande de planification conclue).
Dans le premier sous-état, nous avons un unique état de base pour traiter la demande, où le fournisseur est attendu pour fournir un résultat "res". Les états de conclusion possibles incluent la fin du protocole en raison d'un délai dépasse (timeout) ou de la réception d'un message du demandeur au fournisseur (résultat reçu).

Pour plus de détail nous expliquons la vue globale du protocole de demande de calendrier, représentée dans la figure 4.1. Un état (schedule request(requester, provider)) comporte deux états composites distincts, à savoir (schedule request concluded) et (schedule request pending), dans le cas du protocole de demande de calendrier. Les messages échanges entre ces deux états composites sont les suivants : (informer), signifiant que j'accepte votre calendrier ; (refuser), signifiant que je rejette votre calendrier car il ne me convient pas ; (failure), qui peut être vide, ou (timeout), lorsque le temps réel pour la réponse est écoulé.
Les mises en œuvre des rôles variés du protocole abstrait sont ensuite spécifiées. Les états demeurent inchangés, cependant, les expressions de transition et les actions d'état sont adaptées pour refléter les actions et les événements surveillés spécifiques à chaque rôle.

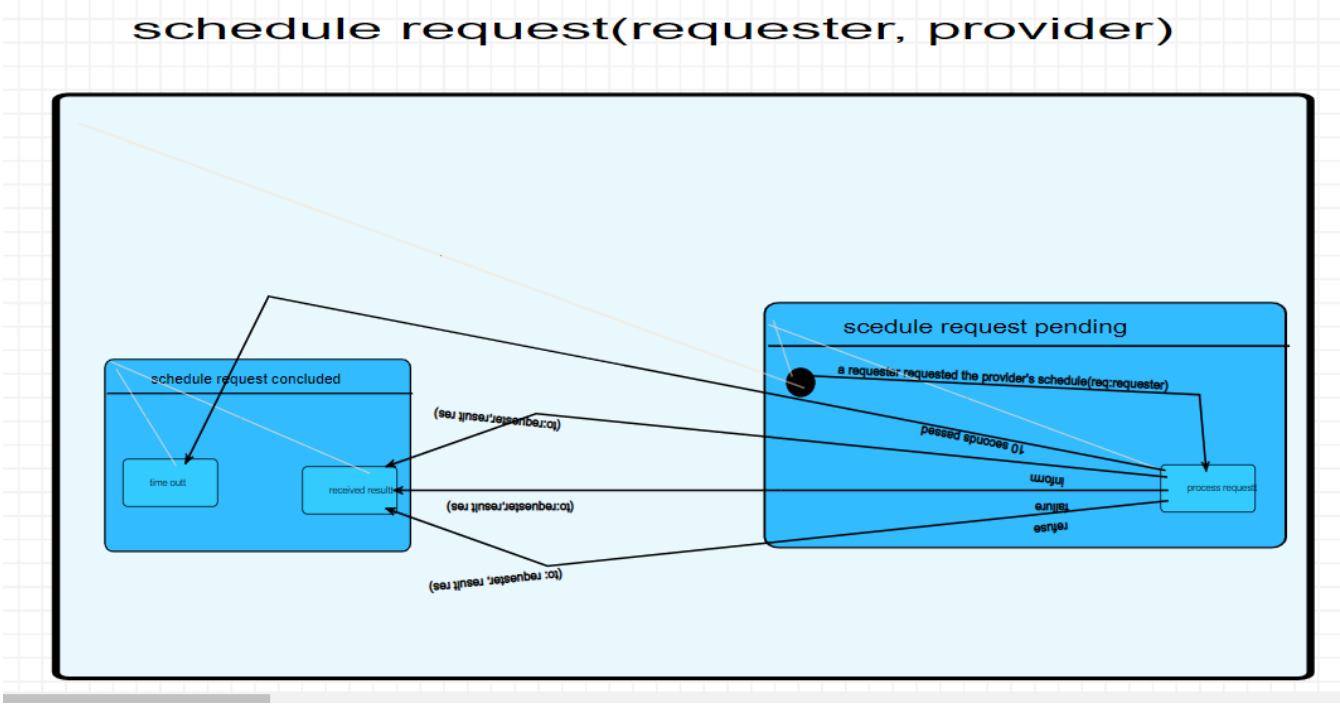

FIGURE 4.1 – Représentation d'un diagramme d'états propositionnels d'un protocole de demande de planification.

Les mises en œuvre des rôles variés du protocole abstrait sont ensuite spécifiées. Les états demeurent inchangés, cependant, les expressions de transition et les actions d'état sont adaptées pour refléter les actions et les événements surveillés spécifiques à chaque rôle.

Le lecteur peut consulter les implémentations potentielles du protocole abstrait dans la Figure 4.2 pour le rôle du demandeur et dans la Figure 4.3 pour le fournisseur. Il est à noter comment les différents messages entrants et sortants sont modélisés ainsi que l'utilisation du langage logique dans les expressions.

Après l'excution de nos règles de transformation de graphes sur les diagrammes d'états-transitions UML presentés dans les figures 4.1, 4.2 et 4.3, nous avons obtenu les nets within nets présentés dans les figures suivantes :

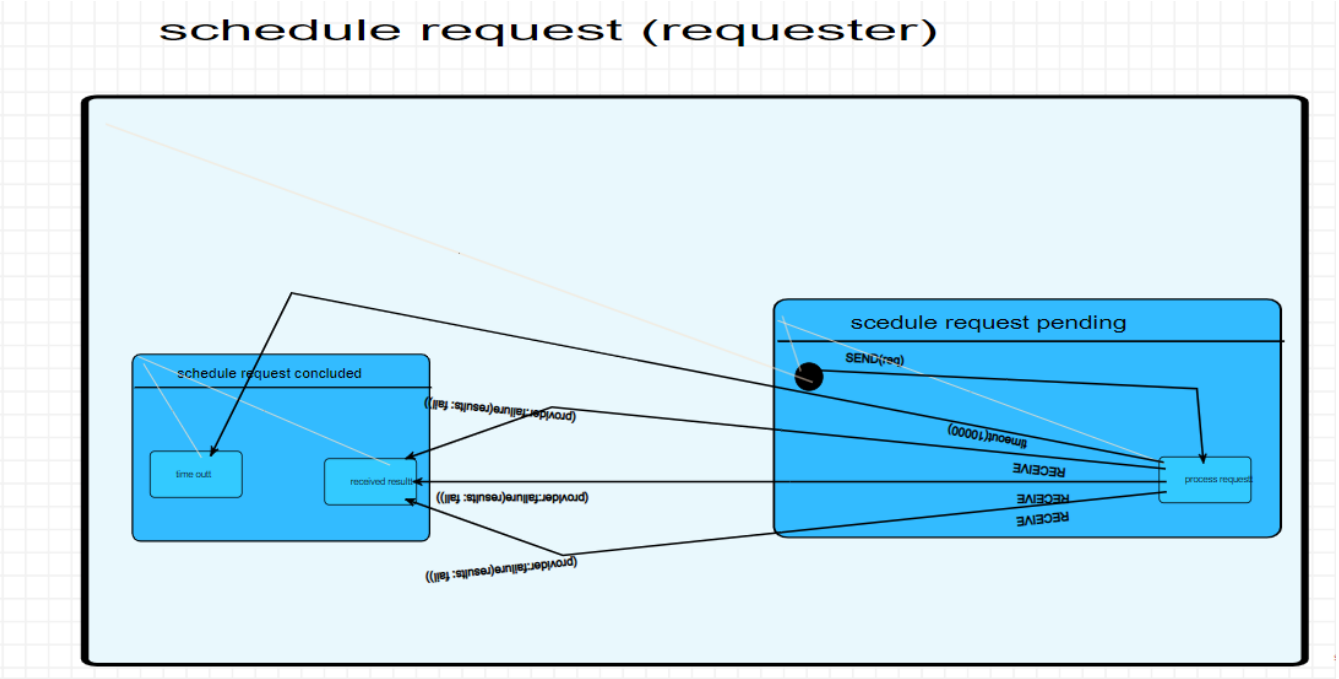

FIGURE 4.2 – Représentation d'un diagramme d'états propositionnels pour un agent demandeur.

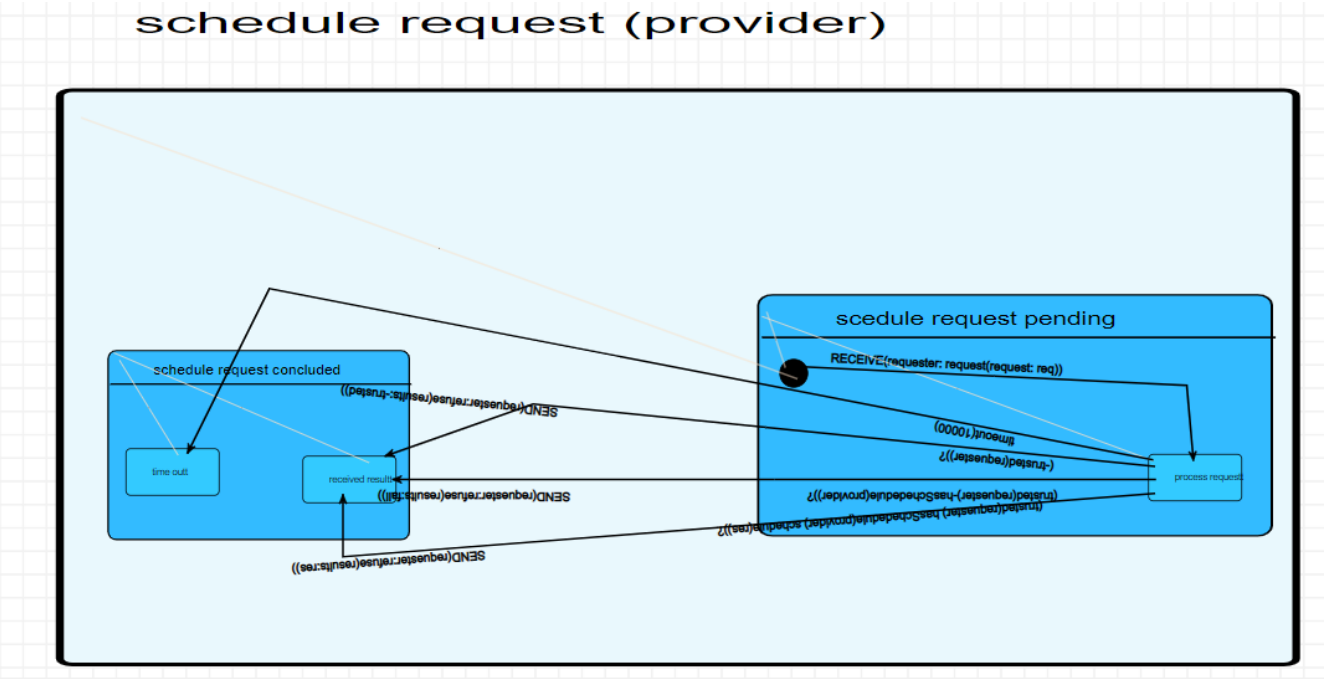

FIGURE 4.3 – Représentation d'un diagramme d'états propositionnels pour un agent fournisseur.

La première figure, la 4.4, représente la transformation du protocole de demande de planification. La deuxième figure, la 4.5, illustre la transformation des éléments propositionnels pour un agent demandeur. La troisième figure, la 4.6, présente la transformation des éléments propositionnels pour un agent fournisseur. La deuxième figure 4.5 c'est la transformation de propositionnels pour un agent demandeur.

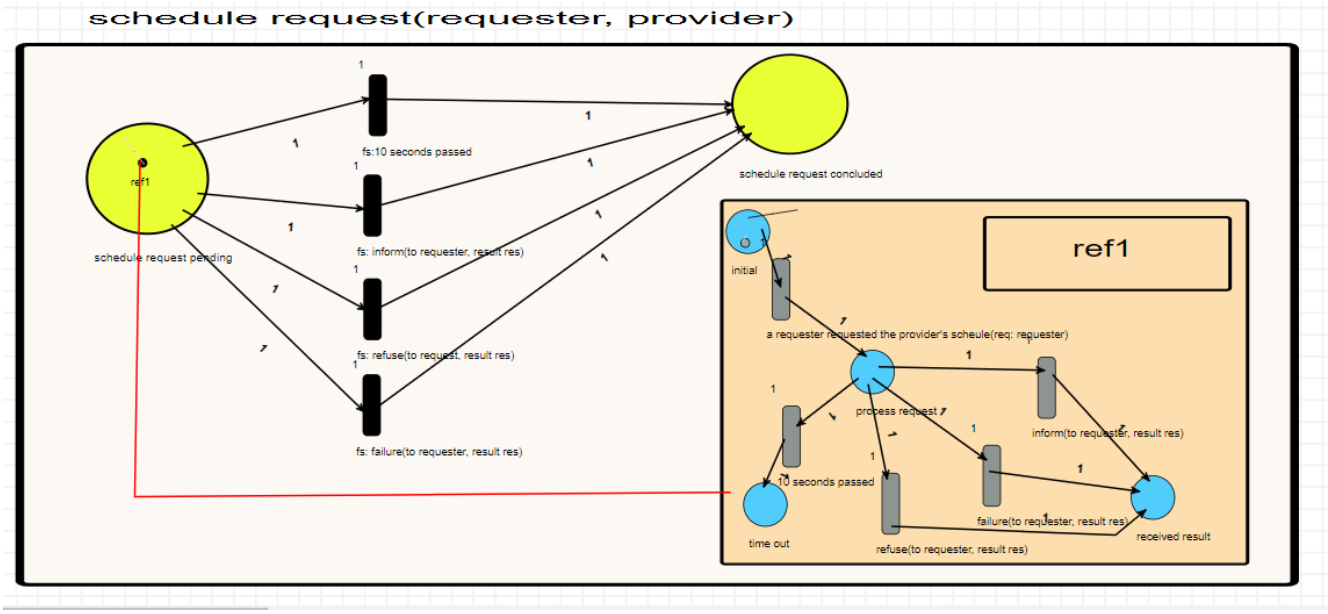

FIGURE 4.4 – Représentation de nets within nets de propositionnels d'un protocole de demande de planification.

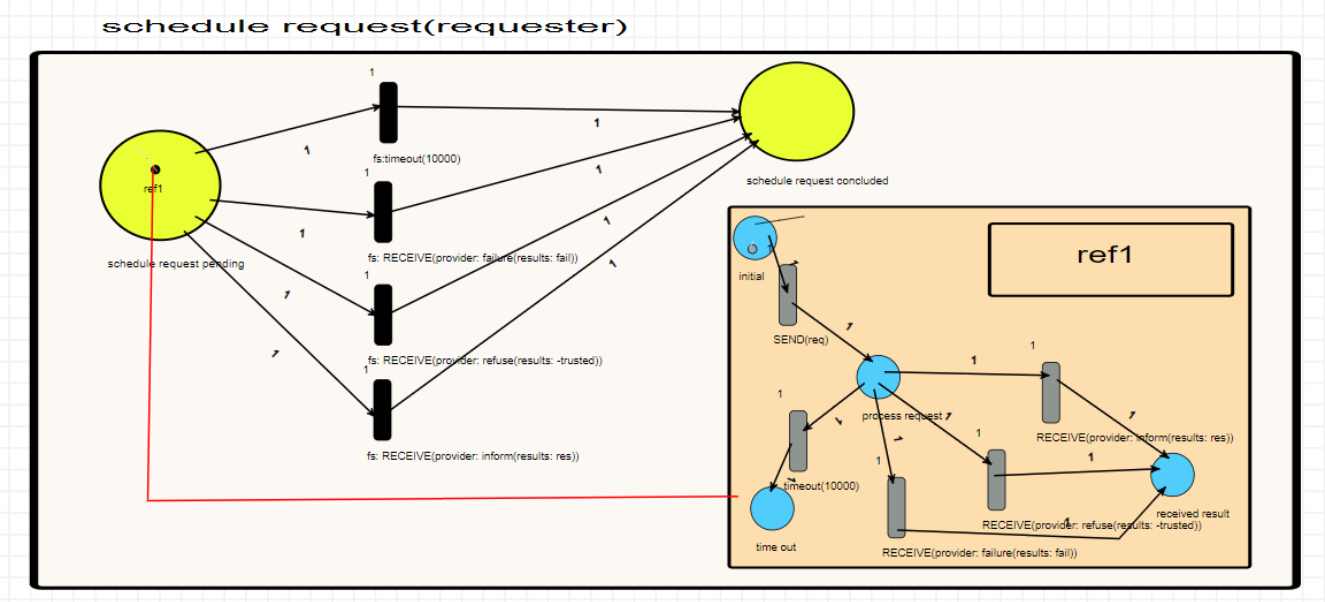

FIGURE 4.5 – Représentation de nets within nets propositionnels pour un agent demandeur.

La troisième figure 4.6 c'est la transformation de propositionnels pour un agent fournisseur.

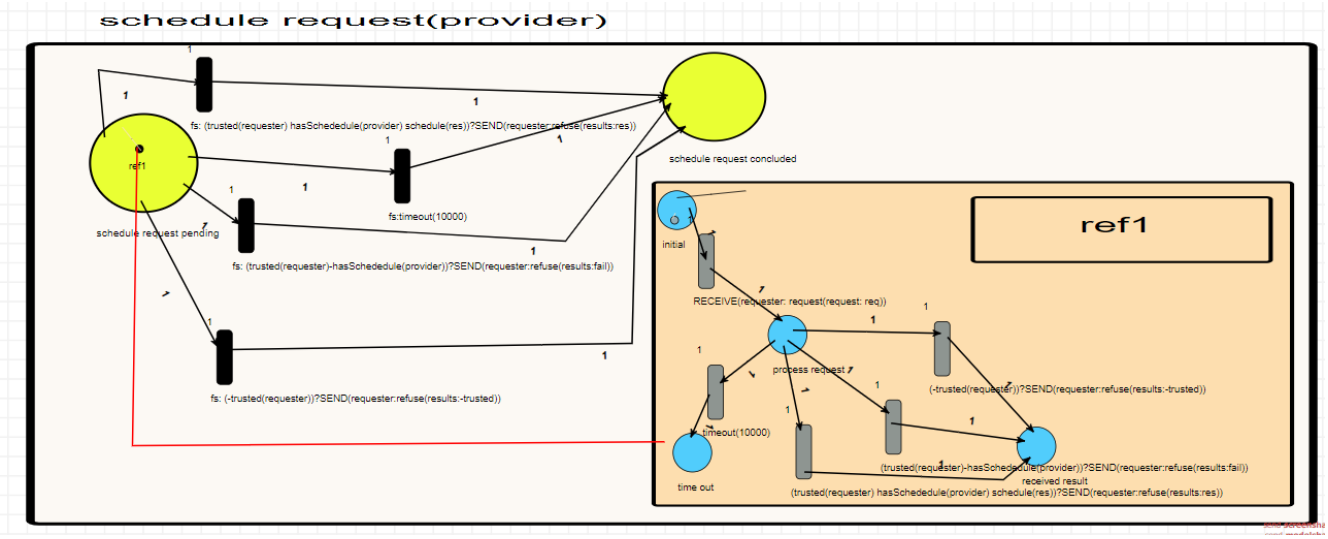

FIGURE 4.6 – Représentation de nets within nets propositionnels pour un agent fournisseur.

4.2 Étude de cas 2 :Protocole d'enchères en anglais

La figure 4.7 illustre la seconde étude de cas qui est un protocole d'enchères anglais [18] modélisé à l'aide d'un diagramme d'états-transitions UML [17]. Dans ce protocole, le commissaire-priseur accepte l'offre faite par l'un des enchérisseurs les plus élevés, éventuellement de manière arbitraire. C'est seulement à travers le processus d'acceptation d'une offre par le commissaire-priseur qu'un chemin d'action partiel vers un nouvel état d'enchère est créé. Dans la figure 4.7, le commissaire-priseur annonce également à tous les enchérisseurs le prix qui a été offert et l'identité de l'enchérisseur le plus élevé.

Deuxièmement, le commissaire-priseur a la possibilité d'abandonner et de déclarer l'article invendu à n'importe quel stade de l'enchère une fois qu'elle a commencé. Cela garantit que l'enchère peut toujours se terminer même s'il n'y a pas un seul enchérisseur le plus élevé. Il y a deux variables d'état associées aux versions du Protocole d'Enchères Anglaises montrées dans la figure 4.7, désignées sous les noms de "cp" (prix actuel) et "hb" (meilleur enchérisseur) respectivement. Les valeurs leur sont attribuées par une assignation explicite dans le cadre des événements du chemin d'action.

Les mises en œuvre des rôles variés du protocole abstrait illustré dans la figure 4.7 sont ensuite spécifiées en un fil séparé pour le commissaire-priseur (figure 4.9) et pour chaque enchérisseur (figure 4.11).

Après l'exécution de notre grammaire de graphe nous obtenons les nets witin nets illustrés dans les figures 4.8, 4.10, 4.12 .

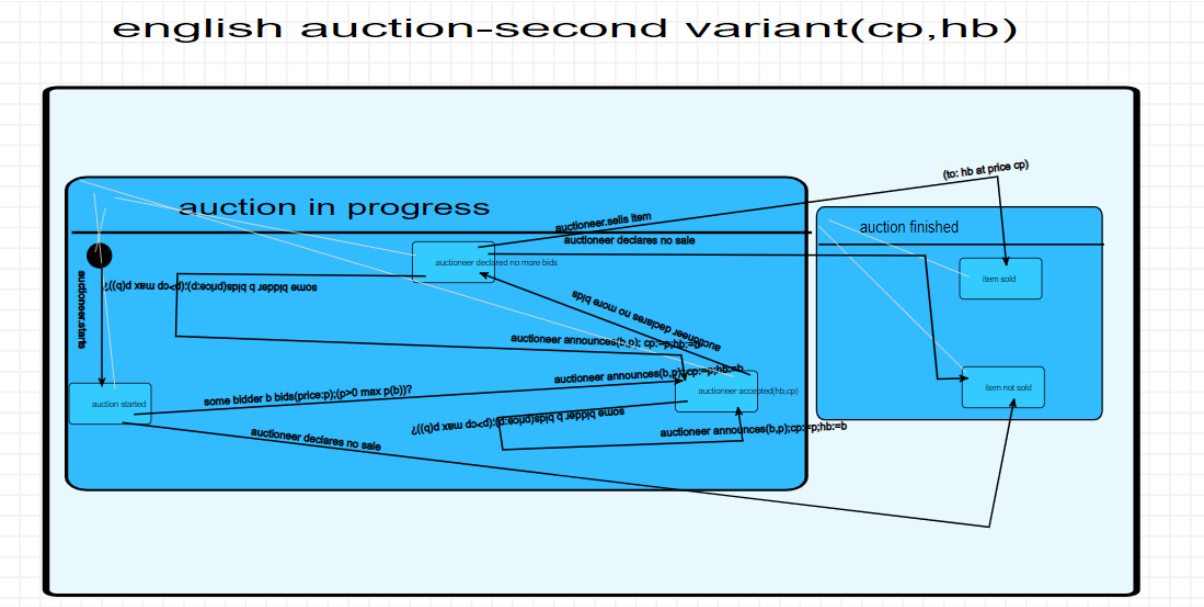

FIGURE 4.7 – Une version plus détaillée du Protocole d'enchères anglais illustré.

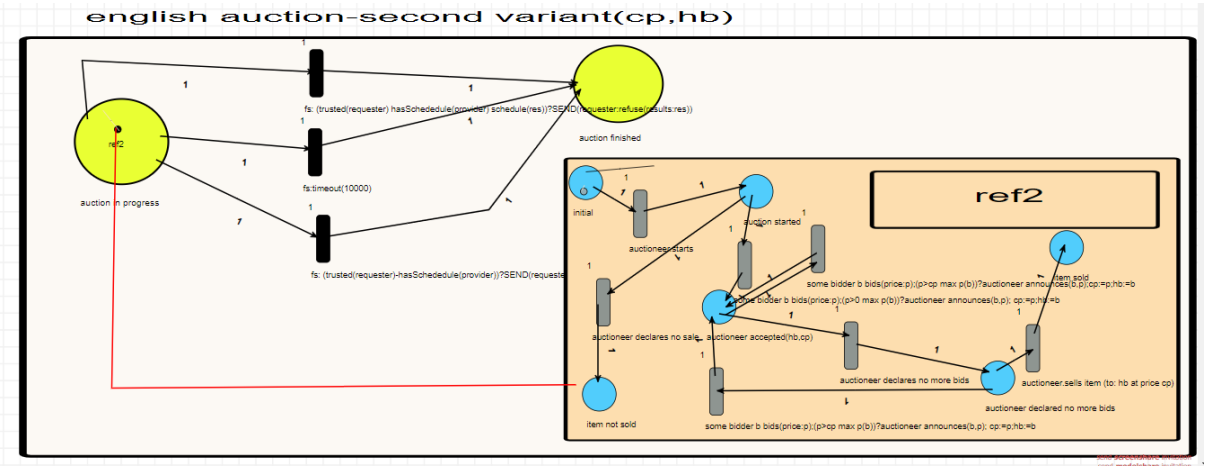

FIGURE 4.8 – Représentation de nets within nets de version plus détaillée du Protocole d'enchères anglais illustré.

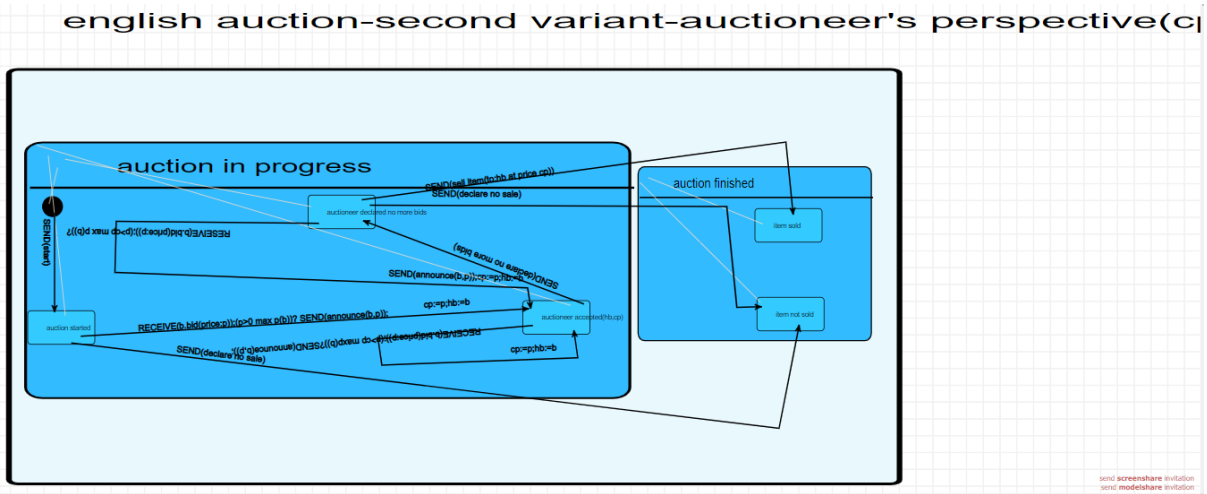

FIGURE 4.9 – Le protocole illustré dans la figure 4.7 du point de vue de l'enchérisseur.

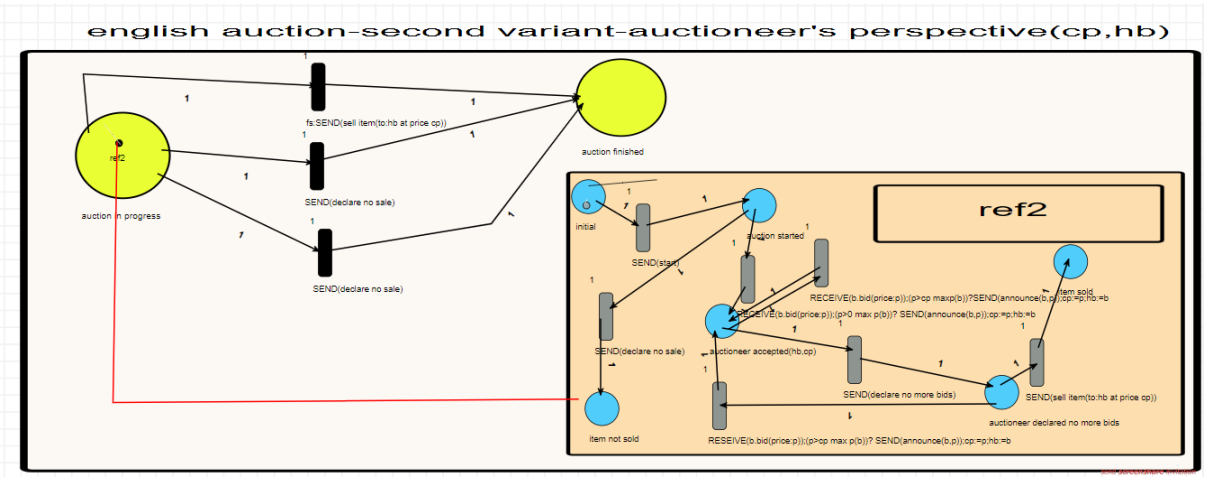

FIGURE 4.10 – Représentation de nets within nets protocole illustré dans la figure 4.7 du point de vue de l'enchérisseur.

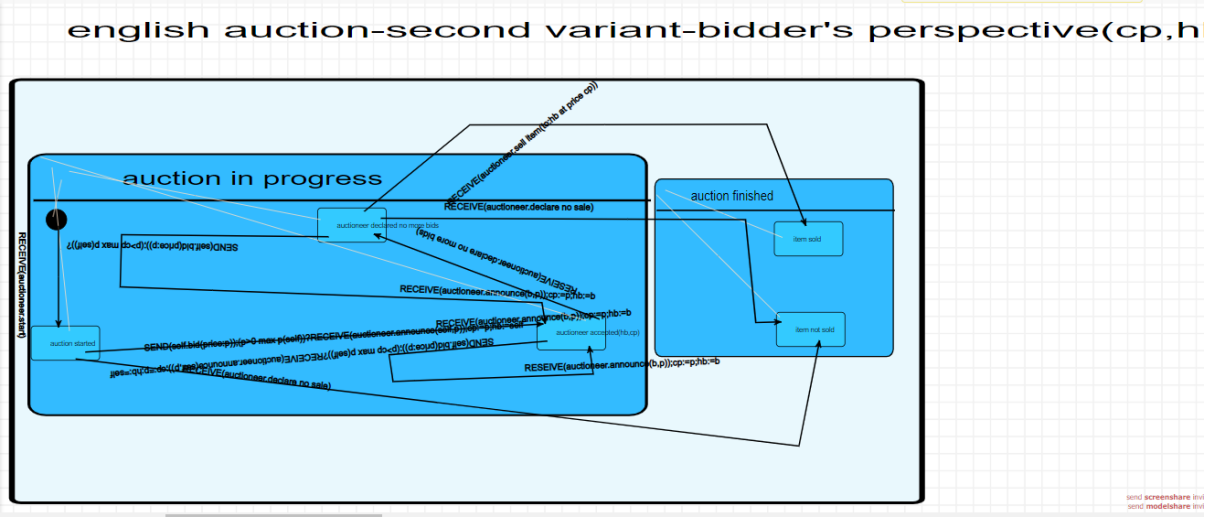

FIGURE 4.11 – Le protocole montré dans la figure 4.7 du point de vue d'un enchérisseur.

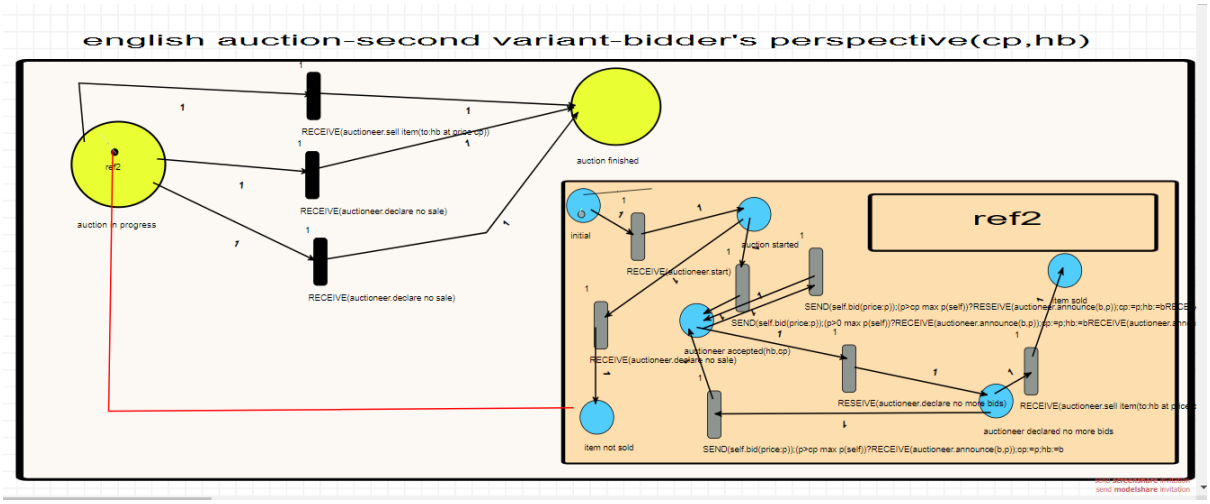

FIGURE 4.12 – Représentation de nets within nets du protocole montré dans la figure 4.7 du point de vue d'un enchérisseur.

Conclution :

Dans ce chapitre, nous avons appliqué notre approche de transformation des diagrammes d'états-transition UML en formalisme "Nets Within Nets" sur deux études de cas distinctes. Ces deux études, portant respectivement sur le protocole de demande de calendrier et le protocole d'enchères anglais, ont été modélisées à l'aide des diagrammes d'états-transitions UML. Ensuite, les Nets Within Nets équivalents ont été automatiquement générés grâce à notre grammaire de graphe.

Conclusion générale :

En conclusion de ce mémoire sur la transformation des diagrammes d'états-transitions UML vers nets within nets en utilisant l'outils de transformation de graphe AToMPM, nous pouvons affirmer que cette approche présente des avantages significatifs pour la modélisation et la vérification des systèmes complexes.

Tout d'abord, la transformation des diagrammes d'états-transitions UML vers les RDP nets within nets offre une représentation graphique plus expressive et facilite également l'analyse formelle et la vérification des propriétés du système. Les RDP sont des modèles mathématiques bien établis et puissants, capables de représenter les systèmes concurrents et non déterministes de manière claire et concise. Cela permet une meilleure compréhension du comportement du système et facilite la détection des erreurs potentielles.

Ensuite, AToMPM, en tant qu'outils utilisé pour cette transformation, se révèle être un outil efficace et convivial. Grâce à ses fonctionnalités avancées, il permet aux modélisateurs de travailler de manière intuitive et efficace. Cela se traduit par un gain de temps considérable lors de la modélisation des systèmes complexes, tout en réduisant les risques d'erreurs humaines.

Cependant, malgré ces avantages, il est essentiel de reconnaitre certaines limitations de cette approche. En particulier, la transformation peut être complexe pour certains systèmes très volumineux ou hautement concurrents. Des efforts supplémentaires de recherche et de développement sont nécessaires pour optimiser le processus de transformation et rendre l'outil encore plus performant et robuste.

En conclusion, la transformation des diagrammes d'états-transitions vers les nets within nets dans AToMPM est une approche prometteuse pour la modélisation et la vérification des systèmes complexes. Elle offre une représentation formelle, expressive, et facilite l'analyse formelle des systèmes. Cependant, il reste encore des défis à relever pour améliorer et étendre cette approche. Avec davantage de recherche et d'investissement, cette méthode pourrait devenir un outil essentiel pour les ingénieurs et les chercheurs travaillant sur la conception et la vérification des systèmes critiques.

Perspectives

Pour parvenir à développer une approche totalement automatique de vérification des diagrammes d'états-transitions UML complexes, nous proposons les améliorations suivantes :

• Amélioration des métamodèles :

- En ce qui concerne le diagramme d'états-transitions UML, nous envisageons de prendre en compte plusieurs niveaux hiérarchiques dans les états composites.
- Pour les réseaux de Petri imbriqués, notre objectif est de prendre en compte un réseau à plusieurs niveaux hiérarchiques, ainsi que tous les éléments additionnels par rapport aux réseaux de Petri ordinaires.

• Compléter l'approche en intégrant la transformation des réseaux de Petri nets within nets vers la spécification de l'outil de vérification formelle Renew.

Cette démarche vise à rendre la vérification des diagrammes d'états-transitions UML complexes entièrement automatisée et plus efficace.

Table des abréviations :

API	Application Programming Interface
ATL	ATLAS Transformation Language
AToMPM	A Tool for Multi-Paradigm Modeling
AToM3	A Tool for Multi-formalism Meta-Modelling
CML	Common Modeling Language
CTL	Computationnel Tree Logic
DSML	Domain-Specific Modeling Language
FPL	Frame-Processing-Language
FSM	Flat State Machine
GUI	Graphical User Interface
IDM	Ingénierie Dirigée par les Modèles
HTML	HyperText Markup Language
LHS	Left Hand Side
LOTOS	Language Of Temporal Ordering Specification
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MOF	Meta Object Facility
NAC	Negative Application Condition
ON	Object Net
OMT	Object Modeling Technique
OMG	Object Management Group
OOSE	Object Oriented Software Engineering
RdP	Model Driven Architecture
Réseaux de Petri	Model Driven Engineering
RHS	Right Hand Side
SMV	Symbolic Model Checker
SN	Sytem Net
UML	Unified Modeling Langage
TCP	Transmission Control Protocol
XML	Extensible Markup Language

Bibliographie

[1] David Harel, Eran Gery, *Executable Object Modeling with Statecharts*. Prentice Hall 1997.

[2] Craig Larman, *UML 2 et les design patterns*. Pearson Education France, 2005.

[3] Benoit Charoux, Aomar Osmani, Yann Thiery-Mieg, *UML 2*. Éditions Eyrolles,2005.

[4] Michael R Blaha, James Rumbaugh, *Modélisation et conception orientées objet avec UML 2*. ,2005.

[5] David Harel and Hillel Kugler, *The Rhapsody Semantics of Statecharts*. ,2005.

[6] Laurent Audibert, *UML 2 : De l'apprentissage à la pratique*. Éditions Eyrolles,2009.

[7] Michelle L. Crane and Juergen Dingel, *UML vs. Classical vs. Rhapsody Statecharts : Not All Models are Created Equal*. ,2005.

[8] Object Management Group, *Unified Modeling Language*, ,2009.

[9] ATOMPM Documentation — ATOMPM 0.10.0 documentation. *(s. d.).*, ,https ://atompm.readthedocs.io/en/latest/ .07/07/2023.

[10] Salim DJAABOUB. Génération d'expressions LOTOS à partir de diagrammes UML. *Thèse de doctorat. Université Constantine 2 Abdelhamid Mehri*, ,Faculté des Nouvelles Technologies de l'Information et de la Communication. Département d'Informatique Fondamentale et ses Applications. 17, 46-48, 74. 2017

[11] El-hillali Kerkouche. Modélisation Multi-Paradigme : Une Approche Basée sur la Transformation de Graphes. 11 *Thèse de doctorat. Université de Mentouri constantine*, ,Faculté des sciences de l'ingénieur. Département informatique. 2011.

[12] BOUDIA MALIKA. Transformation des diagrammes d'états-transitions vers Maude. *Mémoire de Magistère*, ,Université de m'sila. Faculté des mathématiques et de l'informatique. Département d'informatique. 2007.

[13] Olaf Kummer, Frank Wienberg, and Michael Duvigneau. *Renew – The Reference Net Workshop*, ,http ://www.renew.de, October 2004. Release 2.0.1.07/07/2023.

[14] OMG08. *Meta Object Facility (MOF) 2.0 http ://www.omg.org/spec/QVT/1.0/PDF/*, ,Query/View/Transformation Specification. Version 1.0., April 2008.

[15] Krzysztof Czarnecki and Simon Helsen, *"Classification of Model Transformation Approaches"*, ,OOPSLA'03 Workshop on Generative Techniques in the Context of ModelDriven Architecture, 2003.

[16] Nikolaos I. Spanoudakis *Engineering Multi-agent Systems with Statecharts Theory and Practice*, ,SN Computer Science (2021) 2 :317. https ://doi.org/10.1007/s42979-021-00706-5.07/07/2023.

[17] H. R. Dunn-Davies and R. J. *S. Paurobally. Propositional Statecharts for Agent Interaction Protocols*, ,Cunningham, . Electronic Notes in Theoretical Computer Science 134 (2005) 55–75. doi :10.1016/j.entcs.2005.02.020. 07/07/2023.

[18] David, E., Azoulay-Schwartz R. and Kraus, *S., An English Auction Protocol for Multi-Attribute Items, Revised Papers from the Workshop on Agent Mediated Electronic Commerce IV*, ,Designing Mechanisms and Systems (2002).

[19] Hachi Yacine, Benhacine Djilani, *Conception et Réalisation d'un outil de Génération automatique de Spécification Maude a partir des réseaux de Petri*, ,theme de master université echahid hamma lakhdar el oued faculté des sciences exactes Département Mathématique et Informatique(2016).9-18

[20] R udiger Valk, *Universit at Hamburg, Vogt-K olln-Str.30, D-22527 Hamburg*, , Germany valk@informatik.uni-hamburg.de 4-8. 07/07/2023.