République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Djilali BOUNAÂMA Khemis Miliana

Faculté des Sciences et de la Technologie

Département de Mathématiques et d'Informatique

Mémoire Présenté

Pour l'obtention de diplôme de

Master en Informatiques

Spécialité : « Ingénierie De Logiciel »

Titre :

Une Approche De Transformation Des Diagrammes D'activités UML Vers Les Réseaux De Pétri Temporellement Temporisés.

Réalisé par : Saidi Baya

Bouanane Razika

devant le jury composer de:

Mr Hannich.F ……………….Président

Dr.Hachichi.Hiba ………….Encadreur

Mr. Bahloul.D ……………...Examinateur1

Mme. Boudali.F………………. Examinateur2

Année Universitaire 2017/2018

Dédicaces

Ma mère, qui a œuvré pour ma réussite, de par son amour, son soutien, tous les sacrifices consentis et ses précieux conseils, pour toute son assistance et sa présence dans ma vie, reçois à travers ce travail aussi modeste soit-il, l'expression de mes sentiments et de mon éternelle gratitude.

Mon père, qui peut être fier et trouver ici le résultat de longues années de sacrifices et de privations pour m'aider à avancer dans la vie. Puisse Dieu faire en sorte que ce travail porte son fruit ; Merci pour les valeurs nobles, l'éducation et le soutient permanent venu de toi.

A ma chère sœur Amina pour leurs encouragements permanents, et son soutien moral.

A mes chers frères, Hakim et Billel, pour leur appui et leur encouragement,

A mon fiancé : Amine, sans toi je ne serai pas arrivé ici

A toute ma famille pour leur soutien tout au long de mon parcours universitaire,

A tous mes ami(e)s et collègues.

A toute la promotion d'informatique 2016-2017.

Que ce travail soit l'accomplissement de vos vœux tant allégués, et le fuit de votre soutien infaillible,

Merci d'être toujours là pour moi.

Razika

Dédicaces

A mes très chères personnes au monde, mes parents, qui sans eux je Ne serai arrivées là,

A mes frères et mes sœurs,

A ma fille adorée leila

A tous les membres de ma famille

A tous mes amis

A tous mes collègues et Enseignants de la faculté ST de l'université Djilali Bounaama Khemis miliana.

A tous ceux que j'aime et qui me sont chères.

Je dédie ce modeste travail.

Baya

Remerciements

Nous remercions DIEU le tout puissant qui nous ‘a donné la force, la volonté et le courage pour accomplir ce modeste travail.

Nous adressons également nos remerciements à nos parents
Pour le soutien inconditionnel qui nous ‘ont apporté au cours des années de nos études supérieurs. Leur appui nous ‘a été très précieux et nous leur en témoigne aujourd'hui notre plus grande reconnaissance
Nous tenons également à remercier très chaleureusement notre encadreur HIBA HACHICHI. Pour son soutien, sa disponibilité, sa patience, et son aide qui nous ‘ont permis de mener à bien ce travail.

Nos remerciements aux membres de jury M. Boudali,
M. Bahloul et M. Haniche qui ont accepté de juger notre travail

Nous ne saurions oublier dans ces remerciements MONSIEUR LE Recteur de l'Université de Djilali Bounaama, qui nous ‘a donné la chance d'accéder au master2.

Nous adressons également nos remerciements à nos amies et nos collègues de la cellule site web et centre systèmes et réseaux
qui nous ‘ont soutenu durant cette année.

Enfin merci à ceux et celles qui de près ou de loin nous ‘ont permis d'arriver à bout dans ce long périple.

.

Sommaire

Résumé : VI

Introduction générale 7

Introduction générale .... - 1 -

I. L'Ingénierie Dirigée par les Modèles........ - 4 -

I.1. Introduction .... - 4 -

I.2. Concepts fondamentaux de l'ingénierie dirigée par les modèles .... - 4 -

I.2.1. Système........ - 4 -

✓ Les déférentes vues d'un système .... - 5 -

I.2.2. Modèle : .... - 6 -

I.2.4. Méta-modèles et méta-modélisation .... - 8 -

I.3 Transformation de modèles........ - 8 -

I.3.1 Types de transformations .... - 9 -

I.3.2 Classification des approches de transformation de modèles........ - 10 -

a. Transformation Modèle vers Modèle (M2M : Model-to-Model) .... - 10 -

b. Transformation Modèle vers Code (M2T : Model-to-Text) .... - 11 -

I.4. Manipulation des modèles .... - 12 -

I.5. L'architecture dirigée par les modèles (ADM) .... - 14 -

I.5.1 Définition .... - 14 -

I.5.2 Typologie des modèles dans l'ADM .... - 15 -

I.5.3. Cycle de développement de l'approche ADM .... - 16 -

I.5.4 L'architecture à quatre niveaux .... - 17 -

I.6. Transformation de graphes........ - 18 -

I.6.1 Notion de graphe .... - 18 -

I.6.1.1 Graphe non-orienté .... - 18 -

I.6.1.2 Digraphe........ - 19 -

I.6.2 Grammaire de graphe .... - 20 -

I.6.2.1 Règle de transformation .... - 21 -

I.6.2.2 Système de transformation de graphe .... - 21 -

I.7 Outils de transformation de graphes........ - 23 -

I.8. Conclusion........ - 24 -

II. Modélisation semi-formelle avec UML2.0........ - 26 -

I.2.1. Introduction : .... - 26 -

II.2. La modélisation........ - 26 -

II-2-1-modélisation semi-formelle : - 27 -
II.3. Historique d'UML : - 27 -
II.3.1. UML - 28 -
II.3.1.1. Les diagrammes UML - 29 -
a. Diagrammes structurels ou diagrammes statiques (*Structure Diagram*) - 29 -
b. Diagrammes comportementaux ou diagrammes dynamiques (*BehaviorDiagram*) - 30 -
II.4. Les diagrammes d'activité - 32 -
II.4.1. Définition - 32 -
II.4.2. Intérêts des diagrammes d'activité - 33 -
II.4.3. Composition d'un diagramme d'activité - 34 -
II.4.3.1. Les nœuds - 35 -
II.4.3.1.a. Nœud d'activité (*Activity Node*) - 35 -
II.4.3.1.b. Partitions d'activité (*activity partition)* - 41 -
II.4.3.1.c. Région interruptible d'activité (*interruptible activity region)* - 42 -
II.4.3.1.d. Région d'expansion (*expansion region*) - 42 -
II.4.3.1.e. Une pré-condition ou post-condition locale *(precondition or postcondition)* - 43 -
II.4.3.1.f. Ensemble de paramètres (*parameter set)* - 43 -
II.4.3.2. Les arcs (edges) - 43 -
II.4.3.2.2. Flux de contrôle (*Control Flow*) - 44 -
II.4.3.2.3. Flux d'objet (Object Flow): - 44 -
II.4.3.2.4. Handler d'exception (*Exception Handler*): - 44 -
II.5. Conclusion - 45 -
III. Méthodes formelles et modèles RPTT - 47 -
III.1. Introduction : - 47 -
III.2. Les méthodes formelles : - 47 -
III.3 Classification des méthodes formelles - 48 -
III.3.1. L'approche axiomatique - 48 -
III.3.2. L'approche basée sur les états - 48 -
III.3.3. L'approche hybride : - 49 -
III.4. Les avantages des méthodes formelles : - 49 -
III.5. Les langages formels : - 51 -
III.6. Classification des langages formels : - 51 -
III.7. Technique de vérification formelle : - 53 -
III.7.1. Le model checking : - 53 -

III.7.2. Techniques basées sur la simulation : .... - 54 -
III.7.3. Techniques basées sur le test .... - 54 -
III.7.4. Techniques basées sur la preuve de théorème : .... - 54 -
III.8. Les Réseaux de Pétri : .... - 55 -
III.8.1 Présentation des réseaux de Petri : .... - 55 -
III.8.2 Définition Formelle .... - 56 -
III.8.3 Marquage d'un Réseau de Pétri .... - 56 -
III.8.4 Évolution d'un Réseau de Petri .... - 57 -
III.8.4.1 Transition validée .... - 57 -
III.8.4.2 Règle de Franchissement .... - 57 -
III.9 Modélisation Avec les Réseaux de Petri .... - 58 -
III.9.1 Parallélisme .... - 58 -
III.9.2 Synchronisation .... - 59 -
III.9.2.1 Exemple 1 : Problème du producteur/consommateurs .... - 59 -
III.9.3 Calcul De Flux De Données .... - 59 -
III.9.4 Principales Propriétés des Réseaux de Petri .... - 60 -
a. Accessibilité .... - 60 -
b.Bornitude et RDP Sauf .... - 61 -
c. Vivacité .... - 62 -
c.1 Transition Vivante .... - 62 -
c.2 RDP Vivant .... - 62 -
d. Blocage .... - 63 -
e.Réinitialisable et État d'accueil .... - 63 -
f. Couverture .... - 64 -
g. Persistance .... - 64 -
III.10 Les Réseaux de Pétri de Haut Niveau .... - 65 -
III.10.1 Réseau de pétri Coloré .... - 65 -
III.11. Réseaux de Pétri temporellement temporisés (RPTT) .... - 66 -
III.11.1. Principe des réseaux de pétri temporellement temporisé .... - 66 -
III.11.2 La dépendance causale des transitions .... - 67 -
III.11.3 Définition formelle des réseaux de Petri temporellement temporisés .... - 68 -
III.11.4 Des sous-classes du RPTT .... - 68 -
III.12Conclusion : .... - 69 -
IV. Contribution .... - 71 -

IV.1. Introduction : ... - 71 -
IV.2 ATOM^3 ... - 71 -
IV.2.1 Formalisme Diagrammes de Classes dans *AToM*3 ... - 72 -
IV.2.1.1 Contraintes ... - 73 -
IV.2.1.2 Action ... - 74 -
IV.2.1.3 Attributs ... - 74 -
IV.2.2 Transformation de Graphes ... - 75 -
IV.3 Présentation de l'approche de transformation: ... - 77 -
IV.3.1. Méta-modèle des diagrammes d'activité : ... - 77 -
IV.3.1.1. Les classe : ... - 79 -
IV.2.1.2. Les associations ... - 80 -
IV.3.2. Méta-modèle des RPTT : ... - 81 -
IV.3.3. Définition des Règles de Transformation ... - 82 -
IV.3.3.1. Grammaire de Graphes ... - 82 -
IV.4 Etude de cas : ... - 94 -
IV.4.1. Exemple1 : le processus de réalisation d'un mémoire de fin d'étude ... - 94 -
IV.4.2. Exemple2 : le comportement d'un distributeur automatique d'argent (CCP) ... - 96 -
IV.5. Conclusion : ... - 98 -
Conclusion générale ... - 101 -
Références bibliographiques ... 103

Liste des figures

Figure I. 1 : modélisation de l'architecture d'un système [2]. ---------- - 6 -
Figure I. 2 : Relation entre système, modèle et méta-modèle[9] ---------- - 8 -
Figure I. 3 : concepts de base de la transformation de modèles[11]. ---------- - 9 -
Figure I. 4 : Les approches de transformations de modèles[13]. ---------- - 12 -
Figure I. 5 : Le cycle de développement en Y[33]. ---------- - 16 -
Figure I. 6 : Architecture à quatre niveaux [34] ---------- - 17 -
Figure I. 7 : Graphe non-orienté (A)[35]. ---------- - 19 -
Figure I. 8 : Graphe (A) et sous-graphe (B)[35]. ---------- - 19 -
Figure I. 9 : Digraphe [35]. ---------- - 20 -
Figure I. 10 : Graphe orienté étiqueté [35]. ---------- - 20 -
Figure I. 11 : Les étapes d'application d'une règle de transformation [34]. ---------- - 22 -
Figure I. 12 : Système de réécriture de graphe [35]. ---------- - 23 -

Figure II. 1: Evolution d'UML [34]. .... - 28 -
Figure II. 2 : La hiérarchie des diagrammes UML 2.0 [46]. .... - 31 -
Figure II. 3 : Exemple de diagramme d'activité[48]. .... - 33 -
Figure II. 4: Exemple d'activité avec paramètre d'entrée [50]. .... - 35 -
Figure II. 5: Notation nœuds d'activité [51]. .... - 36 -
Figure II. 6 : Arbre de spécialisation du nœud d'objet [51]. .... - 36 -
Figure II. 7 : Notation nœud d'objet [51]. .... - 37 -
Figure II. 8 : Deux représentations équivalentes pour représenter un flux d'objet [51]. .... - 37 -
Figure II. 9: Arbre de spécialisation des nœuds de contrôle[51]. .... - 38 -
Figure II. 10: Représentation graphique des nœuds de contrôles [51]. .... - 38 -
Figure II. 11: Notation nœud de décision[51]. .... - 39 -
Figure II. 12 : Exemple d'une action avec des pré-conditions et post-conditions [51]. .... - 41 -
Figure II. 13 : Exemple illustratif (partitions d'activités)[51] .... - 42 -
Figure II. 14: Région d'expansion [51]. .... - 43 -
Figure II. 15 : Arc d'activité [51]. .... - 43 -
Figure II. 16: Notation flux de contrôle [51]. .... - 44 -
Figure II. 17 : Notation flux d'objet [51]. .... - 44 -
Figure II. 18 : Notation d'un handler d'exception[51]. .... - 45 -

Figure III. 1: Classification des méthodes formelles[51]. .... - 49 -
Figure III. 2 : classifications existantes des langages formels [79]. .... - 52 -
Figure III. 3: L'approche du model checking [81]. .... - 53 -
Figure III. 4:Exemple d'un Réseau de Petri[93]. .... - 56 -
Figure III. 5 : Exemple de Réseau de Petri [94] .... - 57 -
Figure III. 6: Exemple de règle de franchissement de transition [93] .... - 58 -
Figure III. 7: Parallélisme dans les Réseaux de Pétri [93] .... - 59 -
Figure III. 8 : Problème du producteur et consommateurs [95] .... - 59 -
Figure III. 9: Exemple d'un calcul de flux de données par un Rdp [93]. .... - 60 -
Figure III. 10 : Exemple d'un réseau de Petri non borné [96] .... - 61 -

Figure III. 11 : Exemple de vivacité des Réseau de Petri [96] .... - 63 -
Figure III. 12 : Exemple d'un réseau de Pétri réinitialisable [93] .... - 64 -
Figure III. 13 : Exemple d'un réseau de Pétri persistant [93] .... - 64 -
Figure III. 14 : Exemple de partage de ressources dans un réseau de Petri [95] .... - 66 -
Figure III. 15 : Le passage d'un état à un autre pour le jeton [97] .... - 67 -

Figure IV. 1 : interface d'ATOM3 .... - 72 -
Figure IV. 2: Éditeur des propriétés .... - 73 -
Figure IV. 3 : Éditeur de contraintes .... - 74 -
Figure IV. 4: Éditeur des attributs. .... - 75 -
Figure IV. 5 : Éditeur de grammaire .... - 76 -
Figure IV. 6 : Éditeur de règle .... - 77 -
Figure IV. 7 : le diagramme d'activité .... - 78 -
Figure IV. 8 : outil de modélisation du diagramme d'activité. .... - 81 -
Figure IV. 9 : Méta-modèle du RPTT .... - 81 -
Figure IV. 10 : outil de modélisation du RPTT .... - 82 -
Figure IV. 11 : Action initial .... - 83 -
Figure IV. 12 : Action Finale .... - 83 -
Figure IV. 13 : Rule_Action .... - 84 -
Figure IV. 14 : Rule_joinNode .... - 84 -
Figure IV. 15 : Rule_forkNode. .... - 85 -
Figure IV. 16 : Rule_decesionNode. .... - 85 -
Figure IV. 17 : Rule_PinNode. .... - 86 -
Figure IV. 18 : Rule_FlowFinal .... - 86 -
Figure IV. 19 : Rule_ActiviryFinal. .... - 86 -
Figure IV. 20 : Rule_inittAction. .... - 87 -
Figure IV. 21 : Rule_inittAction. .... - 87 -
Figure IV. 22 : Rule_inittoFork .... - 88 -
Figure IV. 23 : Rule_ActiontiAction. .... - 88 -
Figure IV. 24 : RuleActiontoFork. .... - 89 -
Figure IV. 25 : Rule_ActiontoMerge .... - 89 -
Figure IV. 26: Rule_ActiontoJoin. .... - 89 -
Figure IV. 27: Rule_ ForktoAction .... - 90 -
Figure IV. 28 : Rule_ DecisiontoDecision. .... - 90 -
Figure IV. 29 : Rule_ ActiontoSelf. .... - 91 -
Figure IV. 30 : Rule_ ActiontoPin. .... - 91 -
Figure IV. 31 : Rule_ ActiontoActivityFinal .... - 91 -
Figure IV. 32 : Rule_ MergetoActivityFinal. .... - 92 -
Figure IV. 33 : Rule_ DelJoin. .... - 92 -
Figure IV. 34 : Rule_ DelFork. .... - 93 -
Figure IV. 35 : Rule_ DelDecision. .... - 93 -
Figure IV. 36 : Rule_ DelAction. .... - 93 -
Figure IV. 37 : Rule_ DelFlowFinal. .... - 93 -
Figure IV. 38 : Rule_ DelFlowFinal .... - 94 -
Figure IV. 39 : modèle source de processus de réalisation d'un mémoire de fin d'étude .... - 95 -

Figure IV. 40 : modèle cible de processus de réalisation d'un mémoire de fin d'étude .... - 96 -
Figure IV. 41: modèle source CCP .... - 97 -
Figure IV. 42 : modèle cible CCP .... - 98 -

الملخص

أدى التطور التكنولوجي في مجال الشبكات والاتصالات إلى تطوير تطبيقات تناسب التطورات الجديدة ، مثل بروتوكولات الاتصال والأنظمة الموزعة. مع الاخذ بالاعتبار الجانب الزمني.

في الواقع ، يجب أن تتضمن النمذجة آليات إدارة الوقت والأحداث التي تؤخذ في الاعتبار فكرة التطورات المتزامنة للإجراءات. ونتيجة لذلك نحن بحاجة الى نماذج تتحمل اجراءات هيكلية ومؤقتة.

علما، RPTT يملك اجراءات يمكن ان تستمر مع مرور الوقت ولها الفرصة للتعبير عن هذا النوع من السلوك.

العمل المقدم في هذا المشروع هو مساهمة في مجال الهندسة النموذجية المدفوعة IDM.هدفها الرئيسي هو تطبيق تقنيات تحويل النماذج وتحويل الرسم البياني بشكل ادق لتكون قادرة على تطبيق تقنيات ادوات التحليل والتحقق من خلال عملية تطوير الأنظمة المعقدة.

ولذلك فإننا نقترح إنهج واداة لنمذجة وتحويل diagramme d'activité الى نموذج RPTT لتنفيذ ذلك يتطلب استخدام اداة ATOM ولغةPYTHON.

الكلمات المفتاحية: هندسة البرامج بواسطة النماذج، الطرق التحليلية، تحويل الرسم البياني، القواعد النحوية البيانية ،التصميم المتعدد النماذج، نماذجRPTT.

Abstract:

The technological evolution in the field of networks and telecommunications led us to develop applications that are well suited to these new advances, such as communication protocols and distributed systems. The latter apprehend the temporal aspect.

Indeed, the modeling must involve time management mechanisms and events where the notion of simultaneous evolutions of actions is taken into account. Therefore, we need models that support the expression of non-atomic actions structurally and temporally, that is to say divisible actions and not necessarily of zero duration.

However, temporally timed petri net in which actions can last over time have provided the opportunity to express this type of behavior.

The work presented in this project is a contribution in the field of Model Driven Engineering (MDI). Its main objective is the application of model transformation techniques, and more precisely graph transformations, to be able to apply analysis and verification tools during the development process of complex systems.

Therefore, we propose an approach and a tool for the modeling and the transformation of the activity diagrams into temporally timed models of Petri nets.

The implementation of this approach requires the use of the AToM3 tool and the PYTHON language.

Keywords: Model-Driven Engineering, Meta-modeling, Graph Transformation, Graph Grammars, Formal Methods, RPTT Temporarily Timed Petri Net, Activity Diagrams

Résumé :

L’évolution technologique dans le domaine des réseaux et télécommunications nous poussait à faire développer des applications bien appropriées à ces nouveaux progrès, telles que les protocoles de communication et les systèmes repartis. Ces derniers appréhendent l’aspect temporel. En effet, la modélisation doit faire intervenir des mécanismes de gestion du temps et d’évènements où la notion d’évolutions simultanées d’actions est prise en compte. Par conséquent, nous avons besoin d’un modèle sémantique du parallélisme qui supporte l’expression d’actions non atomiques structurellement et temporellement, c’est-à-dire d’actions divisibles et non nécessairement de durée nulle. Cependant, les réseaux de Pétri temporellement temporisés(RPTT) dans lesquels les actions peuvent durer dans le temps ont apporté la possibilité d’exprimer ce type de comportements.

Le travail présenté dans ce projet est une contribution dans le domaine de l'ingénierie dirigée par les modèles (MDI). Son objectif principal est l'application de techniques de transformation de modèles, et plus précisément de transformations de graphes, pour pouvoir appliquer des outils d'analyse et de vérification au cours du processus de développement de systèmes complexes.

Par conséquent, nous proposons une approche et un outil pour la modélisation et la transformation des diagrammes d'activité en modèles de réseaux de pétri à temporisation temporelle.

La mise en œuvre de cette approche nécessite l'utilisation de l'outil AToM3 et du langage PYTHON.

Mots-clés: Ingénierie dirigée par les modèles, méta-modélisation, transformation de graphes, grammaires de graphes, méthodes formelles, réseaux de pétri temporisés, RPTT, diagrammes d'activité.

Introduction générale

Introduction générale

De nos jours les systèmes sont souvent critiques et très complexes dans leur structure ainsi que leur composition. C'est pour cette raison que ces systèmes doivent être modélisés et vérifiés formellement avant leur mise en œuvre. Un des langages les plus utilisés dans la modélisation semi-formelle est UML (Unified Modelling Language), qui est un langage visuel (graphique) spécialement développé pour modéliser proprement un système orienté-objet (OO). Les diverses vues offertes par UML permettent de visualiser plusieurs aspects d'un même système. Ceci permet de mieux gérer la complexité du système. Cependant, UML étant un langage à caractère plutôt visuel, il souffre d'un manque de Sémantique formelle. En effet, les notations semi-formelles et visuelles d'UML peuvent provoquer des inconsistances au niveau des modèles développés, il est donc impossible d'établir des preuves sur la concordance entre la solution envisagée et celle qui a été élaborée d'une part .et d'autre part, la modélisation doit faire intervenir des mécanismes de gestion du temps et d'évènements où la notion d'évolutions simultanées d'actions est prise en compte. Par conséquent, nous avons besoin des modèles qui supportent l'expression d'actions non atomiques structurellement et temporellement, c'est-à-dire d'actions divisibles et non nécessairement de durée nulle.

Cependant, les RPTT (réseaux de pétri temporellement temporisés) dans lesquels les actions peuvent durer dans le temps ont apporté la possibilité d'exprimer ce type de comportements.

C'est pourquoi, nous proposons une approche ainsi qu'un outil qui permet la transformation du modèle UML et plus particulièrement des diagrammes d'activités vers les RPTT en se basant sur la transformation de graphes.

Nous tenterons d'établir les concepts nécessaires a une transformation automatique des diagrammes d'activités conforme à la spécification définie par l'OMG vers les réseaux de pétri temporellement temporisés.

Organisation du mémoire

Ce mémoire est divisé en quatre chapitres :

Dans le premier chapitre, nous proposons un tour d'horizon sur les travaux relatifs à l'IDM il s'agit de mettre l'accent sur l'étude des langages de modélisation des processus en général, et des processus IDM en particulier. Quant à l'IDM, c'est un domaine particulièrement vaste et

évolutif, c'est la raison pour laquelle nous avons opté d'en donner une vision limitée en mettant l'accent sur la notion de transformation de modèles avec ses langages et outils associés. Le chapitre qui suit traitera la Modélisation semi-formelle avec UML2.0, il éclaircit historique d'UML, et décrit ses diagrammes et leurs concepts, en effet ; nous présentons les diagrammes d'activités et leurs principaux éléments qui constituent le modèle de base dans notre travail, Le troisième chapitre est une vue générale sur les méthodes formelles et leurs classifications. Leur intégration à l'IDM sera ensuite étudiée. La fin de ce chapitre sera consacrée aux réseaux de pétri, en mettant l'accent sur les réseaux de pétri temporellement temporisés.

Dans le quatrième chapitre, nous présentons une approche et un outil de modélisation qui est l'ATOM3 pour la spécification et la transformation des diagrammes d'activités d'UML 2.0 vers les RPTT. Tel que nous présentons notre grammaire de graphe, après nous illustrons l'application de notre approche et outil sur deux études de cas qui consistent en un processus de réalisation d'un mémoire, et un distributeur automatique d'argent, nous terminons par une conclusion générale qui résume les contributions de ce mémoire, et discute des limites et des perspectives de notre travail.

Chapitre I :

L'Ingénierie Dirigée par les Modèles

I. L'Ingénierie Dirigée par les Modèles

I.1. Introduction

L'Ingénierie Dirigée par les Modèles (IDM, ou MDE : Model Driven Engineering) est utilisée principalement dans le domaine des logiciels, elle a permis plusieurs améliorations significatives dans le processus de développement de systèmes complexes en se concentrant sur des préoccupations plus abstraites autour des modèles utilisés sur la programmation classique (le code). L'IDM offre un cadre méthodologique et technologique qui permet d'unifier différentes façons de faire dans un processus homogène. Il est ainsi possible d'utiliser la technologie la mieux adaptée à chacune des étapes du développement du logiciel, tout en ayant un processus global de développement qui soit unifié dans un paradigme unique.

L'IDM exploite les modèles en les détaillant en fonction du besoin, jusqu'à obtention d'un ou plusieurs squelettes de codes générés de manière systématique. L'intérêt principal de cette approche réside dans le fait qu'elle permet d'élever la productivité du développement en séparant la « logique métier » des plateformes utilisées.

Dans ce chapitre, nous allons présenter les concepts de base de l'Ingénierie Dirigée par les Modèles (IDM). Nous nous intéresserons plus à l'Architecture Dirigée par les Modèles (ADM) tel que nous introduisons les modèles et le formalisme de modèle, les méta-modèles, Nous présentons par la suite la manipulation des modèles et enfin nous expliquons la transformation de graphe.

I.2. Concepts fondamentaux de l'ingénierie dirigée par les modèles

Dans cette section, nous nous proposons de définir les concepts de base de l'Ingénierie Dirigée par les Modèles afin d'appréhender de manière optimale cette approche.

I.2.1. Système

Dans le cadre de l'ingénierie dirigée par les modèles, nous parlons souvent du système, quel que soit sa nature (existant ou à réaliser). Un système est défini comme un ensemble organisé d'entités collaborant de manière unitaire, et en interaction permanente, pour assurer une ou plusieurs fonctions. Dans les systèmes, on s'intéresse surtout aux logiciels. Ces derniers doivent respecter certains critères de qualité tels que [1] :

- **La convivialité :** Elle décrit la facilité d'apprentissage, d'utilisation, de préparation des données, d'interprétation des erreurs et de rattrapage en cas d'erreur d'utilisation.
- **La validité :** Un logiciel doit assurer la conformité de ses fonctionnalités avec celles décrites dans le cahier des charges.
- **La réutilisabilité :** Un logiciel doit être apte à être réutilisé, partiellement ou dans son intégralité, pour le développement d'autres logiciels.
- **La fiabilité :** Un logiciel doit être capable de gérer correctement ses propres erreurs de fonctionnement en cours d'exécution.
- **La transparence :** Un logiciel doit avoir la faculté de masquer à l'utilisateur les détails inutiles à l'utilisation de ses fonctionnalités.
- **L'intégrité :** Un logiciel doit protéger son code et ses données et doit pouvoir gérer les autorisations d'accès.
- **La portabilité :** Exprime la possibilité de compiler le code source et/ou d'exécuter le logiciel sur des plateformes différentes.
- **L'extensibilité :** Un logiciel doit se laisser maintenir, c'est-à-dire supporter des éventuelles modifications de ses fonctionnalités ou une extension vers des fonctions qui lui seront demandées sans compromettre son intégrité et sa fiabilité.
- La vérifiabilité, L'autonomie, etc.

✓ **Les déférentes vues d'un système**

Les vues définissent le système, ce sont des formulations du problème selon un certain point de vue et elles peuvent se chevaucher pour compléter une description. Il existe cinq types de vues pour le quel leur somme représente le modèle en entier.

La figure I.1 présente différentes vues, indépendantes et complémentaires, permettant de définir l'architecture d'un système [2], chaque vue est une projection, selon un aspect particulier, dans l'organisation et la structure du système.

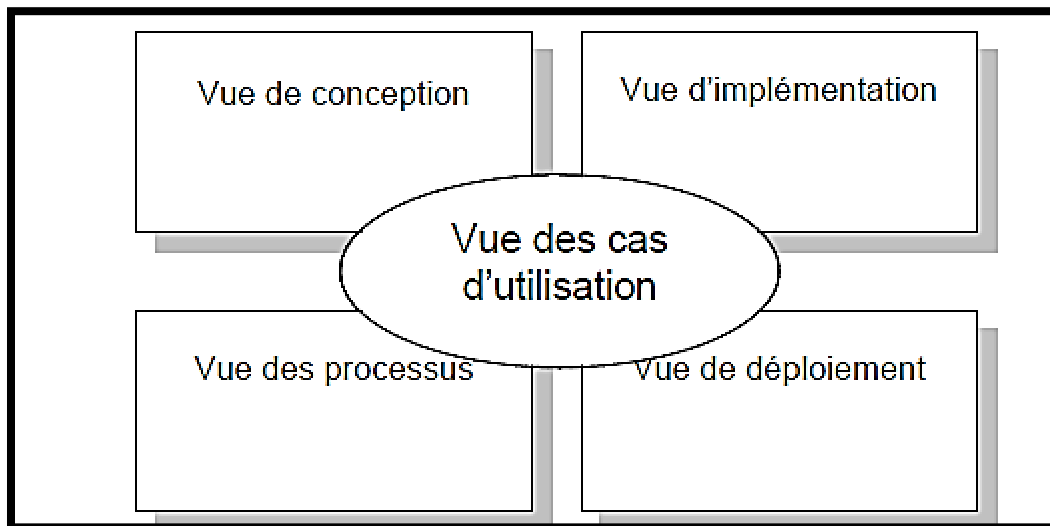


Figure I. 1 : modélisation de l'architecture d'un système [2].

Les différentes vues sont :

- **Vue des cas d'utilisation (qui, quoi) :** Description du système vu par les acteurs du système.
- **Vue logique (de conception)** : elle modélise les éléments et les mécanismes principaux du système. Les éléments UML impliqués sont les classes, les interfaces, etc.
- **Vue d'implémentation :** elle liste les différentes ressources des projets, fichiers binaires, bibliothèques, bases de données, etc. Elle établit le lien entre les composants et elle permet d'établir des dépendances et de ranger les composants en modules.
- **Vue de déploiement :** Dans le cas de système distribué, elle définit les composants présents sur chaque nœud du système. C'est la vue spatiale du projet.
- **Vue des processus** : c'est la vue temporelle et technique, qui manipule les notions de tâches concurrentes, contrôle, synchronisation, processus, threads, etc. [3].

I.2.2. Modèle :

Selon le dictionnaire Hachette : « un modèle est ce qui est proposé à l'imitation ». Cette définition est assez générale. Celles qui suivent proviennent de la littérature technique de l'informatique et donnent plus d'information sur le terme modèle dans l'informatique. « Un modèle est défini comme étant une abstraction d'un système qui se substitue à ce dernier pour le simuler et analyser ses propriétés » [4].

Dans [5], Bézivin et Gerbé ont défini un modèle comme étant « une simplification d'un système créé avec un but spécifique. Il doit être capable de répondre aux demandes à la place du système étudié. Les réponses fournies par le ce dernier doivent être les mêmes que celles fournies par le système, à condition qu'elles restent dans la limite du domaine défini par le but général du système ».

Un modèle doit avoir les propriétés suivantes [6] :

- ***Abstrait :*** Un modèle doit permettre de cacher certains détails inutiles à un moment donné de la conception.
- ***Compréhensible:*** Un modèle doit être facile à assimiler et à manipuler par tous les acteurs entrant dans l'élaboration du système.
- ***Fidèle et précis :*** Un modèle doit représenter fidèlement les propriétés et les caractéristiques du système.
- ***Prédictif :*** Un modèle doit fournir assez d'informations pour permettre de prédire les propriétés du système.
- ***Economique :*** Les coûts de construction et de tests sur le modèle doivent être limités et ne doivent pas dépasser les coûts de construction d'un prototype du système.

La création des modèles est faite en utilisant des langages bien définis. Ces derniers sont connus sous le nom de langages de modélisation.

I.2.3. Langage de modélisation

Un langage de modélisation est défini par : une syntaxe abstraite qui décrit les concepts de base du langage ; une syntaxe concrète qui définit le type de notation (graphique, textuelle ou mixte) ; et une sémantique pour permettre l'interprétation des concepts par les acteurs et les machines. De plus, il est conçu dans un domaine limité et avec des buts spécifiques, de sorte que nous pouvons considérer l'existence de plusieurs langages de modélisation, chacun étant adapté à un domaine spécifique [7]. Le langage conçu pour créer des modèles est souvent défini comme un méta-modèle.

I.2.4. Méta-modèles et méta-modélisation

L'ingénierie dirigée par les modèles préconise l'utilisation d'un mécanisme standard et abstrait pour définir des modèles. Ce mécanisme abstrait est dénoté par le terme *méta-modèle.*

Dans la littérature, nous trouvons plusieurs définitions de méta-modèle.
Selon Jean Marie Favre [8]«un méta-modèle est un modèle d'un langage de modèles».

Les auteurs ont défini un méta-modèle comme étant « un modèle qui définit le langage pour exprimer un modèle »[9].

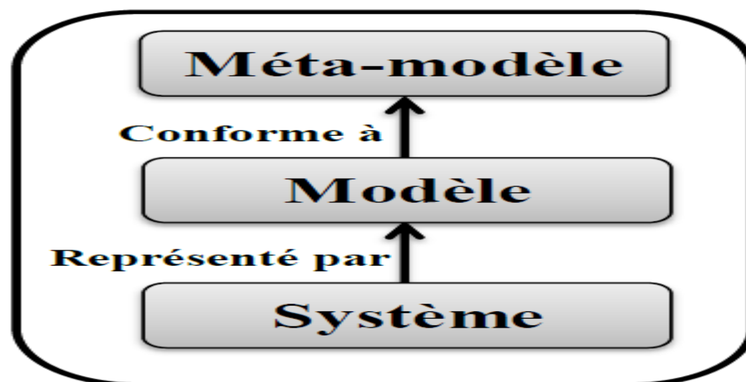


Figure I. 2 : Relation entre système, modèle et méta-modèle[9]

De manière générale, un méta-modèle est le modèle qui sert à exprimer (modéliser) le langage d'expression d'un modèle, avec ses entités, ses relations et ses contraintes. À son tour, le méta-modèle est aussi spécifié dans un langage de méta-modélisation par le méta-méta-modèle. Arrivé à ce niveau d'abstraction méta-circulaire, le langage est assez puissant pour spécifier sa propre syntaxe abstraite.

Enfin, la méta-modélisation est l'activité de construire des méta-modèles. Elle est très utilisée aussi dans le domaine de l'ingénierie des systèmes d'information, et particulièrement dans l'ingénierie des modèles et des méthodes [10].

La relation entre le système, le modèle et le méta-modèle est représentée dans la figure I.2.

I.3 Transformation de modèles

L'ingénierie dirigée par les modèles considère les opérations de transformations de modèles comme le moteur de développement, que ce soit pour l'analyse, l'optimisation ou la génération de code.

La transformation de modèles consiste à générer, à partir d'un ou de plusieurs modèles sources, un ou plusieurs modèles cibles du même système. Les modèles sont dans tous les cas conformes à leurs méta-modèles respectifs. Elle est gouvernée par un ensemble de règles utilisées par le moteur de transformation. Ce moteur prend en entrée le(s) modèle(s) source(s), exécute les règles de transformation et génère le(s) modèle(s) cible(s).

La figure I.3 illustre les principaux concepts impliqués dans le processus de la transformation de modèles.

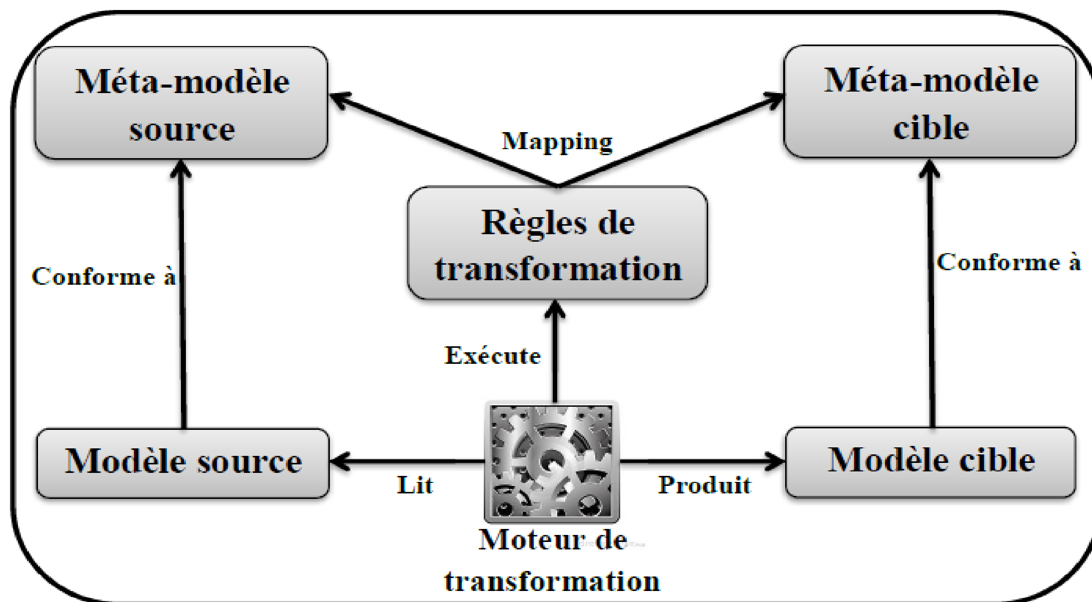


Figure I. 3 : concepts de base de la transformation de modèles[11].

I.3.1 Types de transformations

Un facteur important à prendre en considération dans les transformations de modèles est le niveau d'abstraction. Selon ce facteur, nous pouvons distinguer trois types de transformations [11] :

- ***Transformations horizontales :*** Ces transformations gardent le même niveau d'abstraction en modifiant les représentations du modèle source (ajout, modification, suppression ou restructuration d'informations).
- ***Transformations verticales :*** La source et la cible d'une transformation verticale sont définies à différents niveaux d'abstraction. Un raffinement fait référence à une

transformation qui baisse le niveau d'abstraction. Tandis qu'une abstraction désigne une transformation qui élève le niveau d'abstraction.

- ***Transformations obliques :*** Ces transformations sont généralement utilisées par les compilateurs qui optimisent le code source avant la génération du code exécutable.

Elles sont le résultat de la combinaison des deux premiers types de transformations.

Selon la nature des méta-modèles sources et cibles, nous distinguons deux types de Transformations [12] :

- ***Transformations endogènes :*** La transformation de modèles est qualifiée d'endogène si les modèles sources et cibles sont conformes au même méta-modèle.
- ***Transformations exogènes :*** la transformation de modèles est dite exogène si elle se fait entre deux méta-modèles (source et cible) différents.

I.3.2 Classification des approches de transformation de modèles

Selon la classification proposée par Czarnecki et Helsen [13], les transformations de modèles peuvent être partagées en deux grandes classes : les transformations « Modèle vers Modèle » et les transformations « Modèle vers Code ».

a. Transformation Modèle vers Modèle (M2M : Model-to-Model)

Les transformations de type M2M consistent à transformer un modèle source en un modèle cible, ces modèles peuvent être des instances de différents méta-modèles. Elles offrent des transformations plus modulaires et faciles à maintenir, et permettent la génération de plusieurs modèles intermédiaires avant d'atteindre le modèle final. Ces modèles intermédiaires peuvent être utiles pour étudier les différentes vues du système, son optimisation, la vérification de ses propriétés et sa validation.

Nous pouvons distinguer cinq techniques de transformation M2M :

- ***Approche de manipulation directe :*** Cette approche est basée sur une représentation interne des modèles source et cible, en plus des API pour les manipuler. La combinaison JMI [14] (Java Metadata Interface) et Java sont souvent utilisées dans la mise en oeuvre de cette approche.

- ***Approche relationnelle :*** Cette approche utilise les contraintes pour spécifier les relations entre les éléments du modèle source et ceux du modèle cible en utilisant une logique déclarative basée sur des relations mathématiques. L'un des outils qui supportent cette technique de transformation est medini QVT[15].
- ***Approche basée sur les transformations de graphes :*** Cette approche convient lorsque les modèles sont représentés par des graphes. Elle exprime les transformations sous une forme déclarative. Les règles de transformation sont définies sur des parties de modèle et non pas sur des éléments basiques. Une opération de filtrage de motifs *(Pattern Matching*) est ensuite lancée. Le moteur de transformation compare à chaque fois des fragments du modèle source pour trouver des règles applicables. Ce fragment est ensuite remplacé par son équivalent dans la règle appliquée. Quelques exemples d'outils permettant les transformations de graphes: VIATRA2[16], ATOM3[17], UMLX[18], etc.
- ***Approche dirigée par la structure :*** Elle est divisée en deux étapes, la première se charge de la création d'une structure hiérarchique du modèle cible, la seconde ajuste ses attributs et ses références. Le cadre de transformation fourni par l'outil OptimalJ[19] est un exemple d'une approche basée sur la structure.
- ***Approche hybride :*** Les approches hybrides représentent une combinaison des différentes techniques ou alors celle d'approches utilisant à la fois des règles déclaratives et impératives. Comme ATL[20] est un exemple d'un outil de cette approche.

b. Transformation Modèle vers Code (M2T : Model-to-Text)

Il existe deux techniques de transformations M2T : la première est basée sur le principe du visiteur, tandis que la deuxième est basée sur le principe de templates.

- ***Approche basée sur le visiteur (Visitor-basedApproach):*** Elle consiste à fournir un mécanisme de visiteur pour traverser la représentation interne d'un modèle et créer le code. On peut citer comme exemple le frameworkJamda [21] qui fournit un ensemble de classes pour représenter les modèles UML, une API pour manipuler les modèles, et un mécanisme de visiteur pour générer le code.

- ***Approche basée sur les templates (Template-basedApproach):*** Dans cette approche, la structure d'un template ressemble au code à générer. Au niveau du template , il n'y a pas de séparation syntaxique entre la partie gauche d'une règle de transformation (LHS) et la partie droite de la règle (RHS). le LHS utilise une logique exécutable pour accéder au

modèle source, le RHS combine des patrons non typés et une logique exécutable (la logique : code ou requêtes déclaratives). Parmi les outils qui supportent cette technique, nous pouvons citer : JET [22] et OptimalJ (ce dernier fournit aussi la transformation modèle vers modèle).

La figure I.4 présente les deux classifications ainsi que les différentes techniques de transformation de modèles.

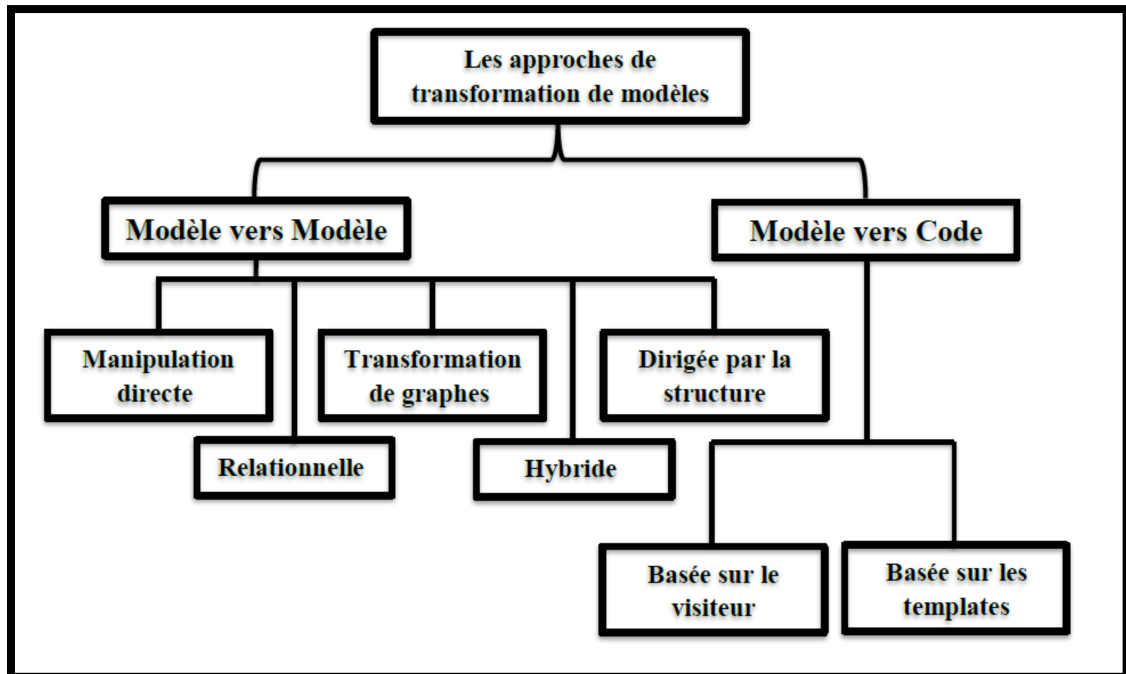


Figure I. 4 : Les approches de transformations de modèles[13].

I.4. Manipulation des modèles

Dans le domaine de l'ingénierie dirigée par les modèles, nous pouvons trouver plusieurs activités liées à la manipulation des modèles. Chacune appartient à un contexte spécifique. Parmi ces activités nous pouvons citer:

- ***La réalisation des modèles :*** Une bonne maîtrise du langage utilisé dans la modélisation et une expertise technique sont nécessaires pour la réalisation des modèles. Plus les systèmes sont complexes, plus leurs modèles deviennent également complexes et surtout de taille importante. Dans ce cas, La qualité de l'outillage devient alors essentielle car ce sont les outils qui permettent de mieux visualiser le modèle, de s'affranchir de certains détails ou encore de vérifier automatiquement la syntaxe.

- ***Le stockage des modèles :*** Cette activité concerne les formats de stockage, l'organisation du stockage et la gestion des métadonnées des modèles.

- ***L'échange des modèles :*** Pour une bonne communication entre eux, les acteurs d'un projet doivent procéder à l'échange de leurs modèles. Ces derniers doivent être compréhensibles par tous, pour éviter tout risque pouvant exposer le système implémenté à des dégâts. L'échange de modèles prend en considération les contraintes du format (sérialisation, transport, etc.), les contraintes de traduction ainsi que celles d'interprétation de la sémantique pour leur interopérabilité.
- ***L'interrogation des modèles :*** C'est l'activité permettant la recherche et la récupération des informations dans les modèles.

- ***L'exécution des modèles :*** Cette activité comprend toutes les tâches réalisables à partir de la simulation du système jusqu'à son exécution en temps-réel, en passant par l'exécution symbolique et la génération du code.

- ***La vérification des modèles :*** La vérification d'un modèle consiste à vérifier ses propriétés propres par rapport à ce que l'on attend de lui. Cette vérification est syntaxique et sémantique. La vérification sémantique est la plus complexe. Il existe différentes techniques pour procéder à la vérification d'un modèle. Nous citons: la technique de preuves qui s'appuie sur les représentations formelles du système basées sur la logique, les automates, les réseaux de Petri, etc. Une autre technique est le « *model-cheking»* qui vise à analyser le comportement du système tout en vérifiant des propriétés telles que la sûreté, l'atteignabilité ou la vivacité. chacune de ces techniques a ses avantages et ses inconvénients. Par exemple, la technique de « *model-cheking»* apporte avec elle un risque d'explosion combinatoire du nombre d'états dans le cas des systèmes complexes.

- ***La validation des modèles :*** La validation affirme ou non que le système implémenté répond aux besoins initiaux. Certaines techniques de tests sont utilisables pour la vérification mais également pour la validation. Les modèles génèrent des scénarios et des vecteurs de test de façon automatique [22].

- ***La gestion de l'évolution des modèles :*** Les modèles évoluent tout au long du cycle de développement du système. Ils peuvent être corrigés ou modifiés en ajoutant d'autres fonctionnalités. Ces modifications sont automatiquement répercutées sur les autres modèles impliqués.

I.5. L'architecture dirigée par les modèles (ADM)

L'ingénierie dirigée par les modèles est un domaine basé sur les modèles et les technologies liées à leurs manipulations. Pour la pratique, il existe plusieurs manières d'utiliser les modèles dans le processus de développement d'un système. L'approche la plus développée et la plus utilisée est L'Architecture Dirigée par les Modèles (MDA : Model Driven Architecture).

I.5.1 Définition

L'architecture dirigée par les modèles est une approche proposée par l'OMG (Object Management Group) [24] ,en 2000 comme une réponse à l'hétérogénéité des technologies utilisées dans le cadre du génie logiciel[25].

L'approche ADM préconise l'utilisation de plusieurs modèles indépendants des détails techniques de l'implémentation. Cette méthode consiste en l'élaboration et la transformation de modèles tout au long du processus de développement d'un système. Ces modèles ont pour objectif de simplifier la gestion de la complexité des systèmes en spécifiant différents niveaux d'abstraction, aussi bien pour la vue globale du système que pour les protocoles et les algorithmes. Ces modèles sont reliés par des liens de traçabilité et peuvent être exprimés de façon textuelle ou graphique.

En résumé, l'ADM est une démarche de développement basée sur les modèles et un ensemble de standards de l'OMG. Cette démarche permet de séparer les spécifications fonctionnelles d'un système des spécifications de son implémentation sur une plateforme donnée. Parmi les standards fondamentaux de l'OMG liés à l'ADM nous pouvons citer :

- *UML (UnifiedModelingLanguage)* [26]: C'est un langage visuel semi-formel pour la modélisation des systèmes. Il permet de schématiser l'architecture, les solutions et les vues avec des diagrammes augmentés de texte.

- *MOF (Meta-Object Facility)[27]*: C'est un standard de méta-modélisation constitué d'un ensemble d'interfaces standards pour définir la syntaxe et la sémantique d'un langage de modélisation, créé principalement pour définir la notation UML.
- *OCL (Object ConstraintLanguage)[28]*: C'est un langage qui, intégré à UML, lui permet de formaliser l'expression des contraintes.
- *XMI (XML MetadataInterchange)[29]*: C'est un standard d'échange de métadonnées.
- *CWM (Common WarehouseMetamodel)[30]*: C'est une interface basée sur UML, MOF et XMI pour faciliter l'échange de métadonnées entre outils, plateformes et bibliothèques de métadonnées dans un environnement hétérogène.
- *MOFM2T (MOF Model-to-Textlanguage[31])*: C'est une spécification utilisée pour exprimer des transformations de modèles en texte.
- *QVT[32]*: C'est un langage standard pour exprimer les transformations de modèles.

I.5.2 Typologie des modèles dans l'ADM

L'approche MDA permet la manipulation des modèles de natures diverses tels que : les modèles d'objets métiers, de processus, de service, de plateforme, etc. Nous pouvons distinguer quatre classes de modèles :

- ***Les modèles d'exigences (CIM : Computation Independent Model) :*** La réalisation de ce modèle est la première étape dans le développement d'un système puisqu'elle va permettre de modéliser toutes les exigences du client et définir les différentes interactions qui impliqueront le système dans ses environnements, interne ou externe.

  De ce fait, le CIM sera considéré comme référence pour vérifier la conformité du système avec les exigences du client.Le CIM ne donne aucun détail sur la façon de la réalisation ou le fonctionnement du système mais exprime clairement des liens de traçabilité avec les modèles futurs.

- ***Les modèles d'analyse et conception abstraite (PIM : Platform Independent Model) :*** Ces modèles doivent être indépendants de toute plateforme d'implémentation mais aussi contenir assez de détails pour permettre la génération automatique du code.

  Ils organisent le futur système en modules et sous-modules et relient le premiermodèle d'exigence CIM avec le code.

- ***Les modèles de description de plateforme (PDM : Platform Description Platform) :*** Ces modèles fournissent l'ensemble de concepts techniques liés à la plateforme d'exécution et à ses services. Ils contiennent toutes les informations nécessaires à sa manipulation.
- ***Les modèles de code (PSM : Platform Specific Model) :*** Les modèles PSM facilitent la génération du code à partir de la combinaison des modèles PIM et PDM. Les PSM expriment, par exemple, les composants, les instructions, les conditions, etc.

I.5.3. Cycle de développement de l'approche ADM

Le cycle de développement de l'approche ADM suit un processus en Y[33] dont les branches représentent les spécifications fonctionnelles du système et les spécifications techniques de la plateforme. L'implémentation est la branche qui rassemble les deux spécifications comme le montre la figure I.5.

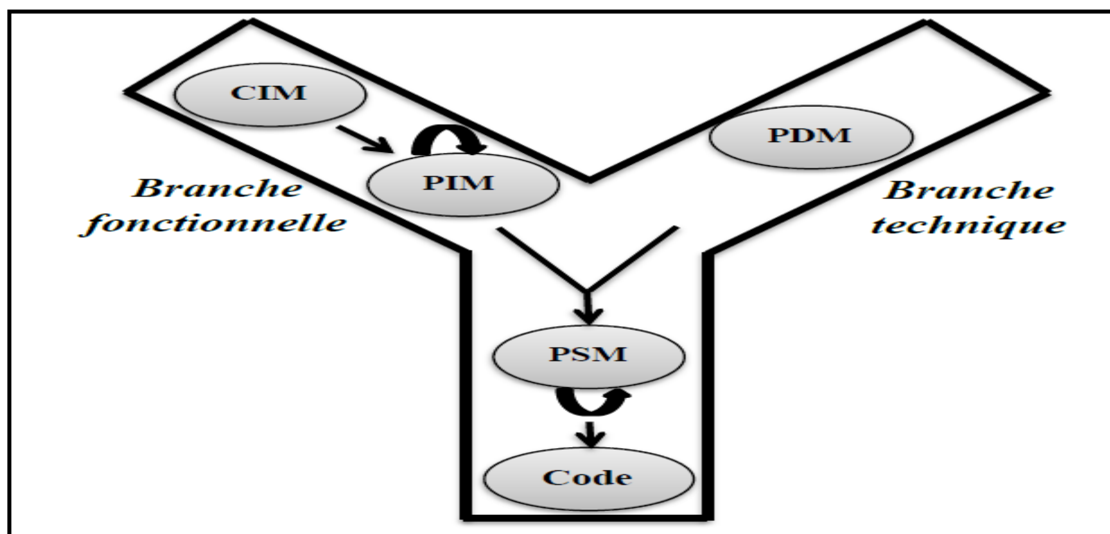


Figure I. 5 : Le cycle de développement en Y[33].

L'architecture dirigée par les modèles considère le processus de développement du système comme une suite successive et stratégique de transformations de modèles. Ces transformations établissent de manière automatique des liens de traçabilité entre les quatre types de modèles discutés précédemment.

I.5.4 L'architecture à quatre niveaux

L'OMG a défini une architecture à quatre niveaux d'abstraction, comme cadre général pour l'intégration des méta-modèles, en se basant sur MOF. Cette architecture est représentée dans la figure I.6.

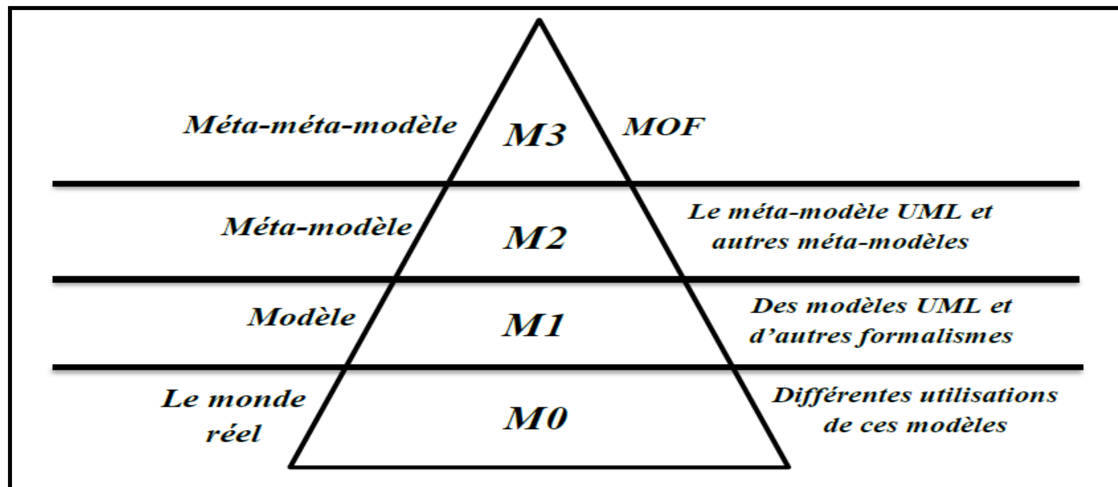


Figure I. 6 : Architecture à quatre niveaux [34]

- ***Le niveau M0 (Le monde réel) :*** Il ne s'agit pas d'un niveau de modélisation proprement dit. Il contient les informations réelles de l'utilisateur, c'est une instance du modèle du niveau M1.
- ***Le niveau M1 (Modèle) :*** Le niveau M1 est composé de modèles d'informations. Il décrit les informations de M0. Ce niveau contient les modèles UML, les PIM et les PSM. Les éléments d'un modèle « M1 » sont des instances des concepts décrits dans un méta-modèle « M2 ».
- ***Le niveau M2 (Méta-modèle):*** Il définit le langage de modélisation et la grammaire de représentation des modèles M1. Par exemple, ce niveau contient le méta-modèle UML qui définit la structure interne des modèles UML ainsi que les profils UML qui étendent le méta-modèle. Les concepts définis par un méta-modèle sont des instances des concepts du MOF.
- ***Le niveau M3 (Méta-méta-modèle):*** Ce niveau est composé d'une seule entité réflexive (auto-descriptive) appelée le MOF. Cette dernière permet de décrire la structure des méta-modèles, d'étendre ou de modifier les méta-modèles existants.

I.6. Transformation de graphes

Les graphes représentent un moyen intuitif, pratique et direct pour la modélisation des systèmes complexes. Les réseaux de pétri, les automates et les diagrammes UML sont des exemples pratiques de graphes de modélisation. Pour spécifier l'évolution des modèles, nous pouvons exploiter les techniques de transformation de graphes. Ces dernières ont évolué pour remédier au problème du manque d'expressivité rencontré par les approches de réécriture classiques telles que les grammaires de Chomsky.

Dans la suite de cette section, nous allons présenter d'abord la notion de graphes ainsi que ces propriétés. Après, nous présenterons les différents principes de la transformation de graphes.

I.6.1 Notion de graphe

Il existe deux types de graphes: les graphes non-orientés et les digraphes.

I.6.1.1 Graphe non-orienté

Un graphe est constitué d'un ensemble de sommets (nœuds) dont certaines paires sont directement reliées par un ou plusieurs liens (arêtes). Plus formellement, un graphe G=(V,E) où :

- V est un ensemble de sommets.
- E est un sous-ensemble de V $\times$V représentant les arêtes.

Soit $v_1, v_2 \epsilon$ V :

- Si (v_1,v_2) ϵ E :Nous disons que les deux sommets v_1 et v_2 sont **adjacents.**
- Si $\exists$ e_1 ϵ E $\cap e_1 = (v_1,v_1)$: Nous disons que v_1 **réflexif** et e_1 est une **boucle.**

L'ordre d'un graphe est le nombre de ses sommets (|V|= Card(V)) et le **degré** d'un sommet est le nombre de ses voisins.

La figure I.7 montre un exemple d'un graphe non-orienté (A). Ce dernier est constitué d'un ensemble de sommets V={1,2,3,4,5} et un ensemble d'arêtes E={(1,1),(1.2),(2,3),(2,4),(3,5),(4,5)}. Dans cet exemple, nous avons |V|=5.

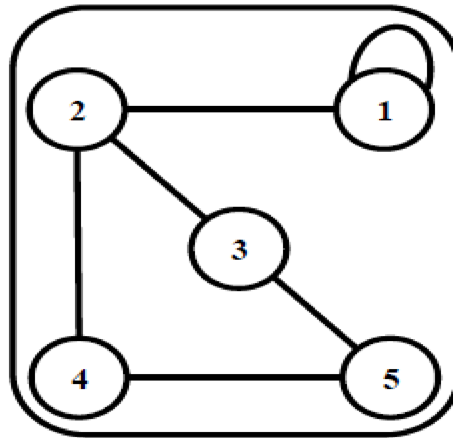


Figure I. 7 : Graphe non-orienté (A)[35].

Un sous-graphe d'un graphe G est un graphe G' composé de certains sommets de G, ainsi que de toutes les arêtes qui relient ces sommets. La figure I.8 présente un exemple où le graphe (B) est un sous-graphe du graphe (A).

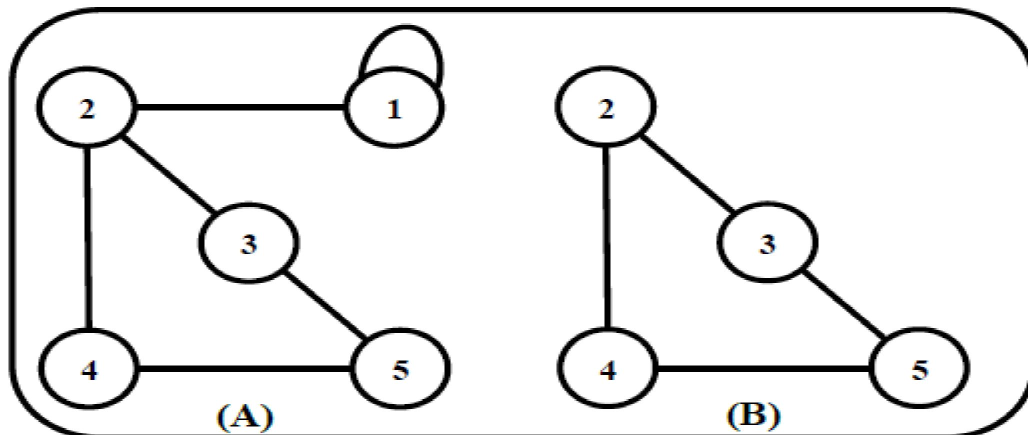


Figure I. 8 : Graphe (A) et sous-graphe (B)[35].

I.6.1.2 Digraphe

Un digraphe (graphe orienté) est un graphe dont les arêtes sont orientées : nous parlons alors de l'origine et de l'extrémité d'une arête. Dans ce cas, une arête est appelée« arc ». La figure I.9 illustre un exemple de graphe orienté.

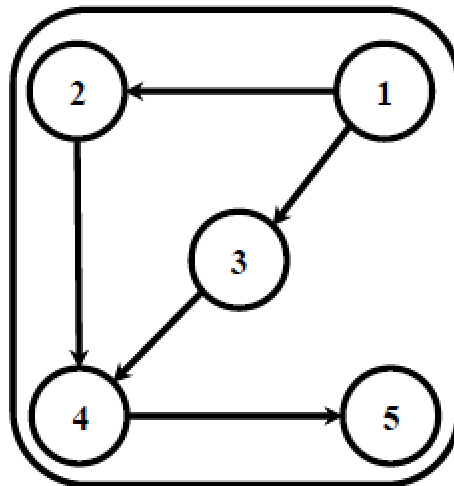


Figure I. 9 : Digraphe [35].

Un graphe étiqueté est un graphe orienté, dont les arcs sont munis d'étiquettes. Si toutes les étiquettes sont des nombres positifs, on parle de graphe pondéré. La figure I.10 illustre un exemple de graphe orienté étiqueté.

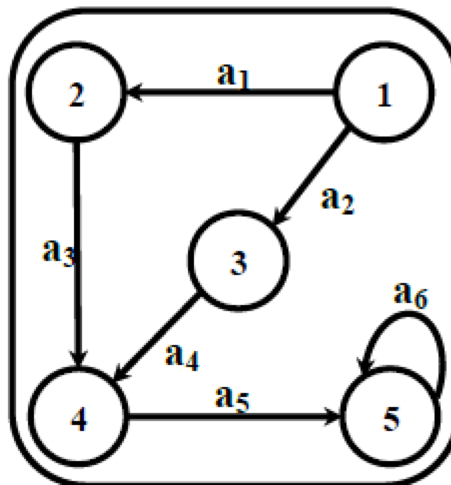


Figure I. 10 : Graphe orienté étiqueté [35].

- ✓ ***Un graphe complet :*** est un graphe simple dont tous les sommets sont adjacents les uns avec les autres.
- ✓ ***Un graphe attribué :*** est un graphe qui peut contenir un ensemble prédéfini d'attributs.

I.6.2 Grammaire de graphe

Une grammaire de graphe est généralement définie par un triplet *GG = (P, S, T)* Où :

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

Une grammaire de graphes distingue les graphes non terminaux, qui sont les résultats intermédiaires sur lesquels *les règles* sont appliquées, des graphes terminaux dont on ne peut plus appliquer de *règles* [36]. Ces derniers sont dans le *langage engendré* par la grammaire de graphe. Pour vérifier si un graphe *G* est dans les langages engendrés par une grammaire de graphe, il doit être analysé. Le processus d'analyse va déterminer une séquence de règles dérivant de *G* [37].

I.6.2.1 Règle de transformation

Une règle de transformation de graphe « *R* » est définie par un 6-uplet Où:

- *R= (LHS, RHS, K, glue, emb, cond)*
- *LHS :* C'est le graphe de partie gauche.
- *RHS :* C'est le graphe de partie droite.
- *K* : C'est un sous graphe de *LHS*.
- *glue :* C'est une occurrence de *K* dans *RHS* qui relie le sous graphe avec le graphe de partie droite.
- *emb :* C'est une relation d'enfoncement qui relie les sommets du graphe de la partie gauche et ceux du graphe de la partie droite.
- *cond :* C'est un ensemble qui indique les conditions d'application de la règle.

I.6.2.2 Système de transformation de graphe

Un système de transformation de graphes est un système de réécriture de graphe qui applique les règles de la grammaire de graphe sur son graphe initial de façon itérative et dans un ordre bien défini jusqu'à ce que plus aucune règle ne soit plus applicable.

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

- ***Première étape :*** choisir une occurrence du graphe de partie gauche *LHS* dans *G*.
- ***Deuxième étape :*** Vérifier les conditions d'application d'après *cond*.

- ***Troisième étape :*** Retirer l'occurrence *LHS* (jusqu'à *K*) de *G* ainsi que les arcs pendillés (tous les arcs ayant perdu leurs sources et/ou leurs destinations). Ce qui fournit le graphe de contexte *D* de *LHS* qui a laissé une occurrence de *K*. [34]

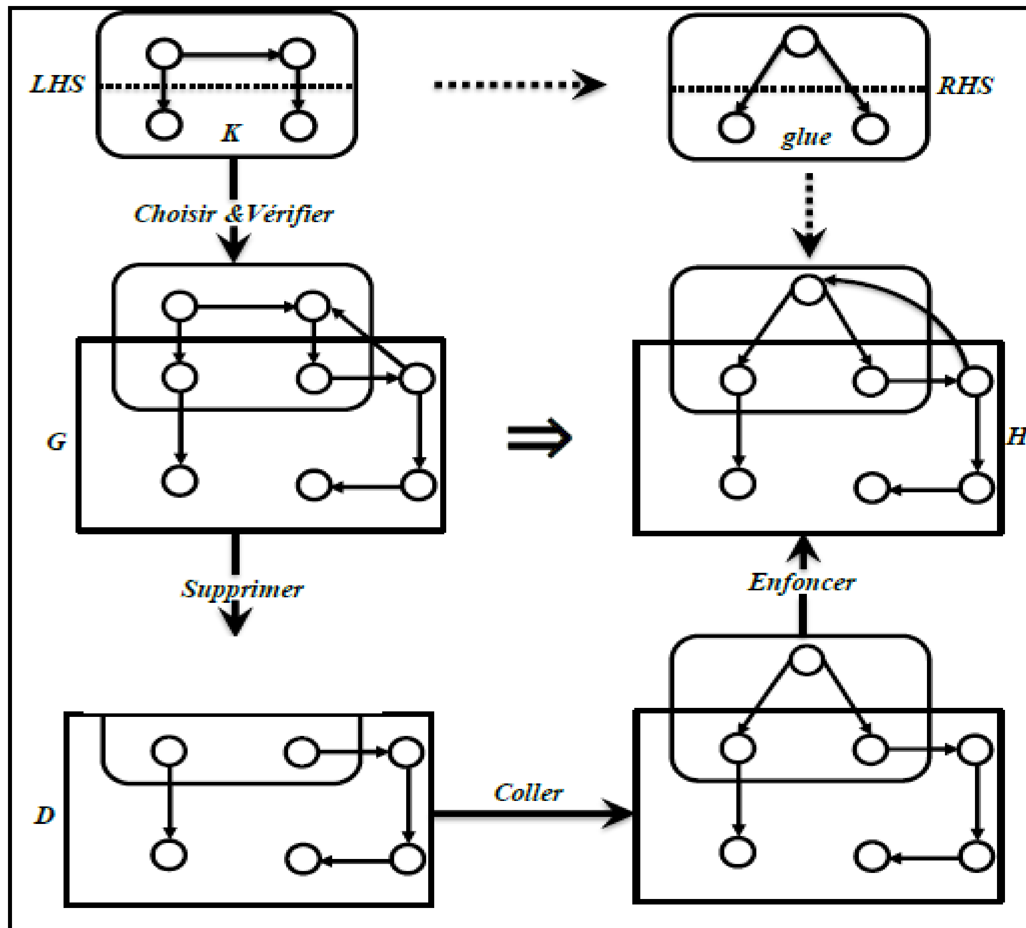


Figure I. 11 : Les étapes d'application d'une règle de transformation [34].

- ***Quatrième étape :***Coller le graphe de contexte *D* et le graphe de partie droite *RHS* suivant l'occurrence de *K* dans *D* et dans *RHS*, c'est la construction de l'union de disjonction de *D* et *RHS*, et pour chaque point dans *K*, identifier le point correspondant dans *D* avec le point correspondant dans *RHS*.
- ***Cinquième étape :***Enfoncer le graphe de partie droite dans le graphe de contexte de *LHS* suivant la relation d'enfoncement *emb*: pour chaque arc incident retiré avec un sommet *v* dans *D* et avec un sommet *v'* dans l'occurrence de *LHS* dans *G*, et pour chaque sommet *v''* dans *RHS*, un nouvel arc incident est établi (même étiquette) avec l'image de *v* et le sommet *v''* à condition que *(v', v'')* appartient à *emb*.

L'application de R à un graphe G pour fournir un graphe H est appelée une *dérivation* directe depuis G vers H à travers R, elle est dénotée par G ⇒ H.

Soit S un graphe initial, le langage engendré *L (P, S, T)* est l'ensemble des graphes dérivés à partir de *S* en appliquant les règles de *P* qui sont étiquetées par les symboles de *T*.

La figure (I.9) illustre les différentes étapes d'application d'une règle de transformation. Tandis que la figure (I.12) représente le principe du système de réécriture de graphes.

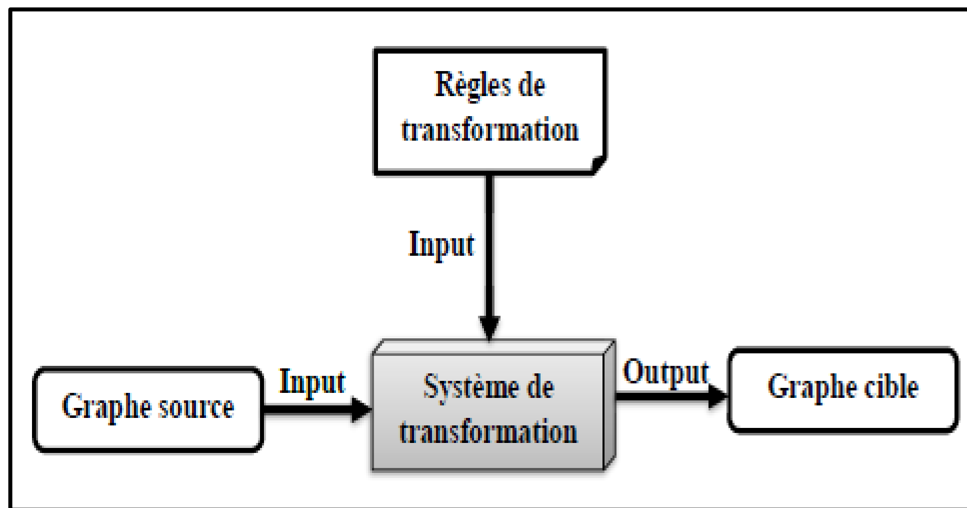


Figure I. 12 : Système de réécriture de graphe [35].

L'approche de transformations de modèles a plusieurs avantages par rapport aux autres approches, nous pouvons citer:

- Les grammaires de graphes sont un formalisme naturel, visuel, formel et de haut niveau pour décrire les transformations.
- Les fondements théoriques des systèmes de la réécriture de graphes permettent d'aider à vérifier certaines propriétés des transformations telles que la terminaison ou la correction.

I.7 Outils de transformation de graphes

Plusieurs outils ont été développés pour faire de manière efficace des transformations de Modèles à l'aide des transformations de graphes, parmi ces outils on peut par exemple citer :

- AGG [AGG]: The **A**ttributed **G**raph **G**rammar System.

- FUJABA [Fujaba]: **F**rom **U**ML to **J**ava **a**nd **b**ack **a**gain.

- ATOM3 [Atom3]: **A T**ool for **M**ulti-formalism and **M**eta-**M**odelling.

- VIATRA [Viatra]: **Vi**sual **A**utomated model **TRA**nsformations.

- GreAT [Great]: The **G**raph **Re**write **A**nd **T**ransformation tool suite.

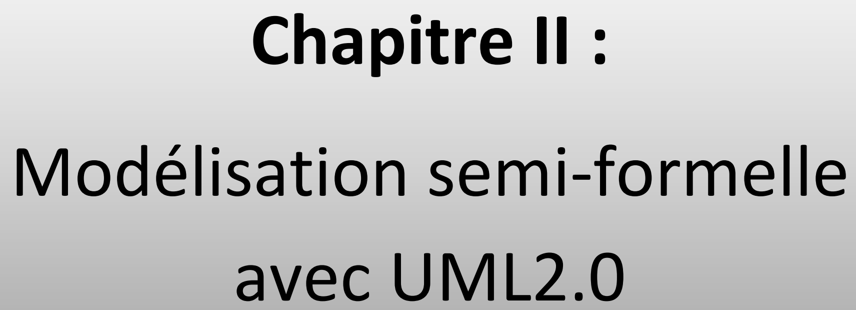
- booggie [booggie]: **b**rings **o**bject-**o**riented **g**raph **g**rammars **i**nto **e**ngineering.

- Et d’autre outils comme : DiaGen, GenGED, PROGRES, *GrGen.NET,* MoTMoT, GROOVE, etc.

Nous nous intéressons dans notre méthode par l’outil *AToM*3.

I.8. Conclusion

Dans ce chapitre, nous nous sommes intéressés à l’ingénierie dirigée par les modèles et ses différents principes. Nous avons survolé également les techniques de transformation et de traitement des modèles. Après nous avons présenté la démarche ADM ainsi qu’une introduction à la transformation de graphes.



Chapitre II :

Modélisation semi-formelle avec UML2.0

II. Modélisation semi-formelle avec UML2.0

I.2.1. Introduction :

Depuis dernière décennie, UML demeure le standard de facto de la modélisation objet conçu par l'OMG, il vient pour exprimer visuellement une multitude des solutions a plusieurs vues et ce, par la diversité des diagrammes qu'il offre, servant à documenter des composants de larges systèmes complexes à dominante logicielle [34].

Les importantes révisions apportées pour UML, viennent intégrer plusieurs nouveaux aspects statiques et dynamiques, représentant à la fois un objectif et un défit tenant à décrire la technologie logicielle dans des démarches Architectures Dirigée par les Modèles(ADM),Model Driven Engineering(MDE)et *service oriented architecture*(SOA) [38].UML2 se veut plus productif et devient le premier support des architectures dirigées par les modèles (ADM)[34].

Dans ce deuxième chapitre, nous introduisons les différents types de modélisations, ainsi les principes de la modélisation UML, un court aperçu sur ses diagrammes, et comme nous tenterons d'établir une transformation automatique des diagrammes d'activités vers les réseaux de Pétri temporellement temporisé(RPTT).

C'est pourquoi nous nous intéressons particulièrement aux diagrammes d'activités, nous présentons leurs intérêts et une description détaillée de leurs principaux éléments, qui seront modélisés dans le formalisme source de la transformation constituant l'objet de notre travail.

II.2. La modélisation

La modélisation d'un système est le processus de développement des modèles.

Selon MarvinL et Minsky [32] un modèle est une représentation abstraite qui contient un ensemble restreint d'informations sur un système réel et un point de vue différent ou perspective de ce système. D'autre part, la Modélisation offre des avantages considérables aux concepteurs des systèmes tels que : la facilité de compréhension du fonctionnement des systèmes avant sa réalisation et un bon moyen de maîtriser sa complexité et d'assurer sa cohérence.

En informatique la modélisation est vue comme une séparation entre les différent besoins fonctionnels et non fonctionnels (tels que : la sécurité, la fiabilité, l'efficacité, la performance, la flexibilité, ..., etc.)[40].

Par ailleurs La modélisation d'un système est venue à signifier ce qui représente un système en utilisant une sorte de notation graphique, qui est presque Toujours basée sur des notations dans UML.

Il y a plusieurs formes de modélisation, ils sont classés selon le degré du formalisme des langages ou des méthodes impliqués dans le processus de la modélisation. Nous intéressons à la modélisation semi-formelle.

II-2-1-modélisation semi-formelle :

Le processus de modélisation semi-formelle est basé sur un langage textuel ou graphique pour lequel une syntaxe précise est définie [41]. La sémantique d'un tel langage et souvent assez faible. Néanmoins, ce type de modélisation permet d'effectuer des contrôles et de réaliser des automatisations pour certaines tâches.

La plupart des méthodes de modélisation semi-formelles s'appuient fortement sur des langages graphiques. Ceci se justifie par la puissance expressive du modèle graphique. Par Ailleurs, l'appui de la modélisation semi-formelle (tels que : UML) sur des Langages graphiques, permet la production de modèles assez faciles à interpréter [42].

2-2-1-1- l'approche orienté objet : l'approche orientée se classe dans la modélisation semi-formelle. Elle considère le logiciel comme une collection d'objets dissociés. La fonctionnalité du logiciel émerge de l'interaction entre les différents objets qui le contient [43].

Cette modélisation a connu une large utilisation. Toutefois, l'absence de standards uniformisant a donné lieu à une génération multi variée de logiciels hétérogènes et souvent dupliqués, dispersant ainsi l'effort de la communauté informatique. Au milieu des années 90 des prémices de standardisation apparaissent donnant naissance à un langage unifié sous l'appellation abréviée **UML** (the unified Modeling Language for object-oriented developement).

II.3. Historique d'UML :

La naissance d'UML est due à la fusion des trois méthodes de références dans le domaine de modélisation objet durant les années 1990. En effet, il s'agit de la fusion en 1994 des deux méthodes, celle de Grady Booch Booch-93 adaptée à la construction et OMT- 2 (*Object Modelling Technique*) celle de James Rumbaugh qui se concentre sur l'analyse et l'abstraction. Cette fusion a donné naissance à une nouvelle méthode : la méthode unifiée (*The Unified Methode*).

Plus tard, en 1995 Ivar Jacobson, le créateur des cas d'utilisation (*Use Cases*), rejoint le groupe en fusionnant avec Booch-93 et OMT-2 la méthode OOSE (*Object Oriented Software Engineering*) dont l'objectif essentiel est la détermination des besoins du système.

En 1997 en sa version 1.1 UML a été adopté par l'OMG (Object Management Group).

A la fin de 2006 la version UML 2.0 devient un langage de modélisation de logiciels standardisés

La figure II.1 montre l'historique de constitution du langage UML

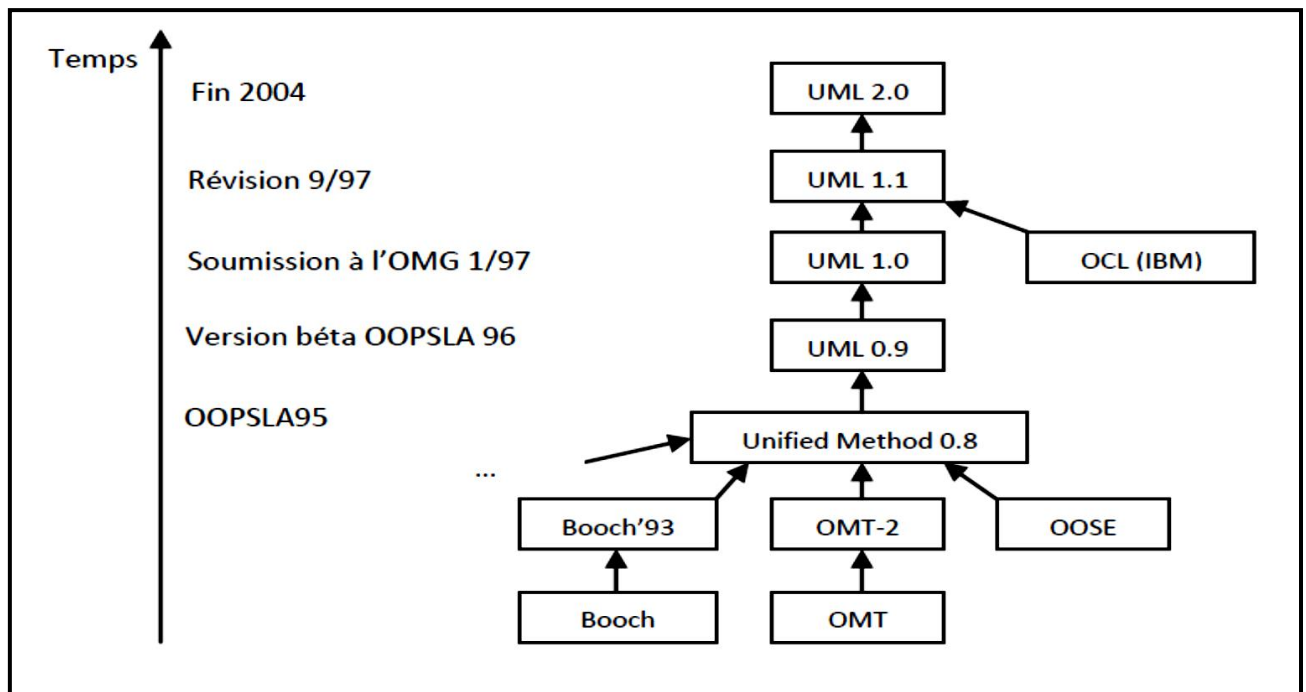


Figure II. 1: Evolution d'UML [34].

II.3.1. UML

UML (Unified Modeling Language) est un langage de modélisation semi-formelle permet de visualiser, spécifié, Construire et documenter tous les aspects et artefacts d'un système logiciel [44].

La notation UML est conçue pour servir de langage de modélisation objet indépendamment de la méthode suivie, il ne constitue qu'une partie d'une méthode de développement logiciel. Ses auteurs ont en effet estimé qu'il n'était pas opportun de définir une méthode en raison de la diversité des cas particuliers. Ils ont préféré se borner à définir un langage graphique qui permet de représenter et de communiquer les divers aspects d'un système.

UML est un langage de modélisation objet [45]. En tant que tel, il facilite l'expression et la communication de modèles en fournissant un ensemble de symboles (La notation) et de règles qui régissent l'assemblage de ces symboles (la syntaxe et la sémantique).

II.3.1.1. Les diagrammes UML

Un diagramme est la représentation graphique d'un ensemble d'éléments qui constituent un système. La plupart du temps, il se présente sous forme d'un graphe connexe où les sommets correspondent aux éléments et les arcs aux relations. Les diagrammes servent à visualiser un système sous différentes perspectives et sont donc des projections dans un système. Pour les systèmes complexes, un diagramme ne représente qu'une vue partielle des éléments qui composent ces systèmes [46].

UML 2.0 propose treize types de diagrammes (9 en UML 1.3) pour représenter les différents points de vue de modélisation. Ils se répartissent en deux grands groupes :

1- Diagrammes structurels (*Structure Diagram*).

2- Diagrammes comportementaux (BehaviorDiagram).

a. Diagrammes structurels ou diagrammes statiques (*Structure Diagram*)

Ces diagrammes permettent de visualiser, spécifier, construire et documenter l'aspect statique ou structurel du système informatisé.

- ***Diagramme de classes (Class diagram):*** Le but d'un diagramme de classes est d'exprimer de manière générale la structure statique système, en termes de classes et de relations entre ces classes. Une classe des attributs, des opérations et des relations avec d'autres classes.
- ***Diagramme d'objets (Object diagram):*** Le diagramme d'objet permet d'éclairer un diagramme de classe en l'illustrant par des exemples. Il montre des objets et des liens entre ces objets (les objets sont des instances de classes dans un état particulier).
- ***Diagramme de composants (Component diagram) :*** il montre les composants du système d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, bases de données...). Il montre la mise en œuvre physique des modèles de la vue logique avec l'environnement de développement.

- ***Diagramme de déploiement (Deploymentdiagram) :*** Ce type de diagramme UML montre la disposition physique des matériels qui composent le système (ordinateurs, périphériques, réseaux...) et la répartition des composants sur ces matériels. Les ressources matérielles sont représentées sous forme de nœuds, connectés par un support de communication.
- ***Diagramme des paquetages (Package Diagram) :*** un paquetage est un conteneur logique permettant de regrouper et d'organiser les éléments dans le modèle UML, il sert à représenter les dépendances entre paquetages.
- ***Diagramme de structure composite (Composite Structure Diagram) :*** Le diagramme de structure composite permet de décrire sous forme de boîte blanche les relations entre les composants d'une seule classe.

b. Diagrammes comportementaux ou diagrammes dynamiques (*BehaviorDiagram*)

Les diagrammes comportementaux modélisent les aspects dynamiques du système. Ces aspects incluent les interactions entre le système et ses différents acteurs, ainsi que la façon dont les différents objets contenus dans le système communiquent entre eux.

- ***Diagramme des cas d'utilisation (Use Case Diagram) :*** Les cas d'utilisation sont une technique de description du système étudié selon le point de vue de l'utilisateur. Ils décrivent sous la forme d'actions et de réactions le comportement d'un système. Donc, le diagramme des cas d'utilisation, permet d'identifier les possibilités d'interaction entre le système et les acteurs. Il permet de clarifier, filtrer et organiser les besoins.
- ***Diagramme d'activité (Activity Diagram) :*** Un diagramme d'activité est une variante des diagrammes d'états-transitions. Il permet de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation. Dans un diagramme d'activité les états correspondent à l'exécution d'actions ou d'activités et les transitions sont automatiques.
- ***Diagramme états-transitions (State Machine Diagram) :*** permet de décrire sous forme de machine à états finis le comportement du système ou de ses composants. Il est composé d'un ensemble d'états, reliés par des arcs orientés qui décrivent les transitions.

- ***Diagramme de séquence (SequenceDiagram) :*** Il représente séquentiellement le déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs. Le diagramme de séquence peut servir à illustrer un cas d'utilisation.
- ***Diagramme de communication (Communication Diagram) :*** C'est une représentation simplifiée d'un diagramme de séquence, en se concentrant sur les échanges de messages entre les objets.
- ***Diagramme global d'interaction (Interaction OverviewDiagram) :*** permet de décrire les enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquence (variante du diagramme d'activité).
- ***Diagramme de temps (Timing Diagram) :*** Le diagramme de temps permet de décrire les variations d'une donnée au cours du temps.

La figureII.2, montre la hiérarchie des diagrammes d'UML 2.0.

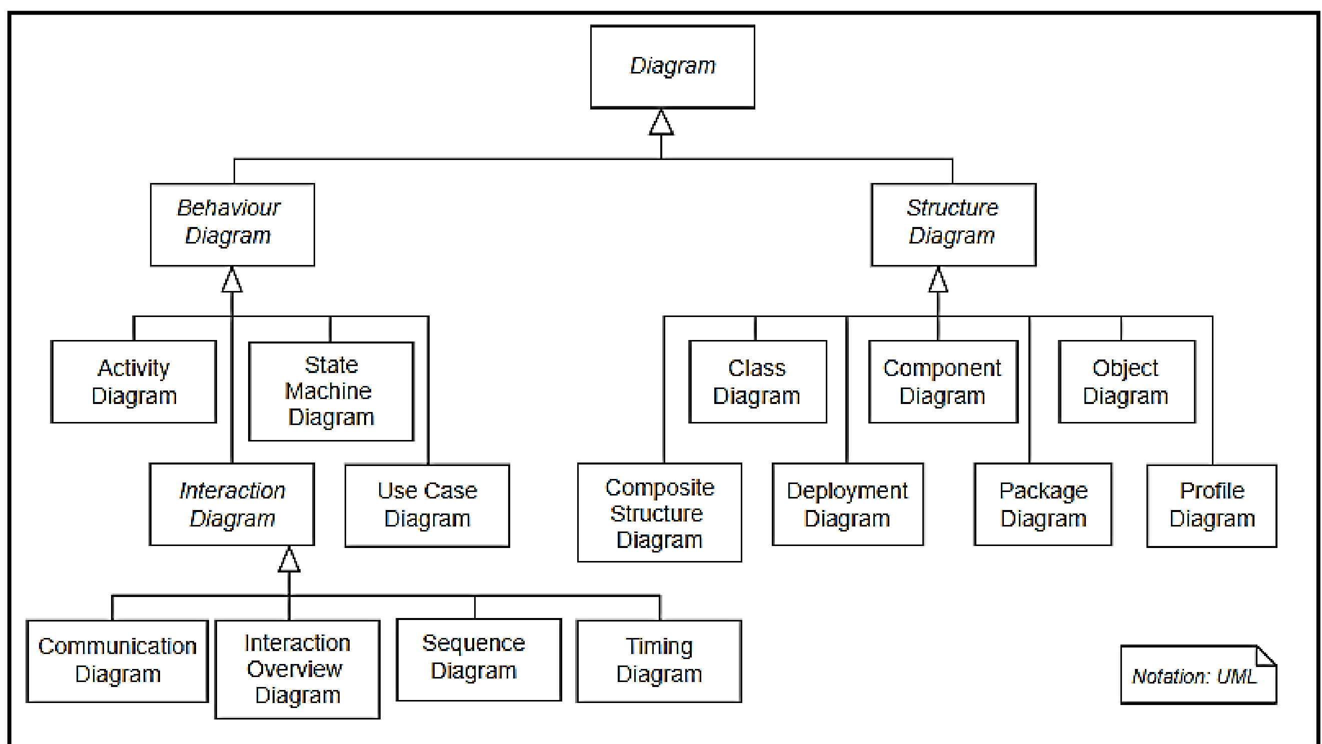


Figure II. 2 : La hiérarchie des diagrammes UML 2.0 [46].

II.4. Les diagrammes d'activité

II.4.1. Définition

UML permet de représenter graphiquement les aspects dynamiques des systèmes à l'aide de plusieurs diagrammes comportementaux. Parmi eux, on distingue les diagrammes d'activité qui permettent de mettre l'accent sur les traitements. Ils sont utilisés pour représenter le comportement d'une méthode ou le déroulement d'un cas d'utilisation [47].

Un diagramme d'activité est une variante des diagrammes d'états-transitions, dans lequel les états correspondent à l'exécution d'actions ou d'activités, et les transitions sont automatiques (les transitions sont déclenchées par la fin d'une activité et provoquent le début Immédiat d'une autre). Un diagramme d'activité peut être attaché à n'importe quel élément de modélisation afin de visualiser, spécifier, construire ou documenter le comportement de cet élément [48].

Ces diagrammes constituent un outil de modélisation des systèmes Workflows, des modèles orientés service et des processus métiers (ils montrent l'enchaînement des activités qui concourent au processus).Un modèle d'activité consiste en activités liées par des flux de données et de contrôle. Une activité peut varier d'une tâche humaine à une tâche complètement automatisée.

La différence principale entre les diagrammes d'interaction et les diagrammes d'activité, est que les premiers modélisent le flot de contrôle entre *objets*, alors que les seconds sont utilisés pour modéliser le flot de contrôle entre *activités*.

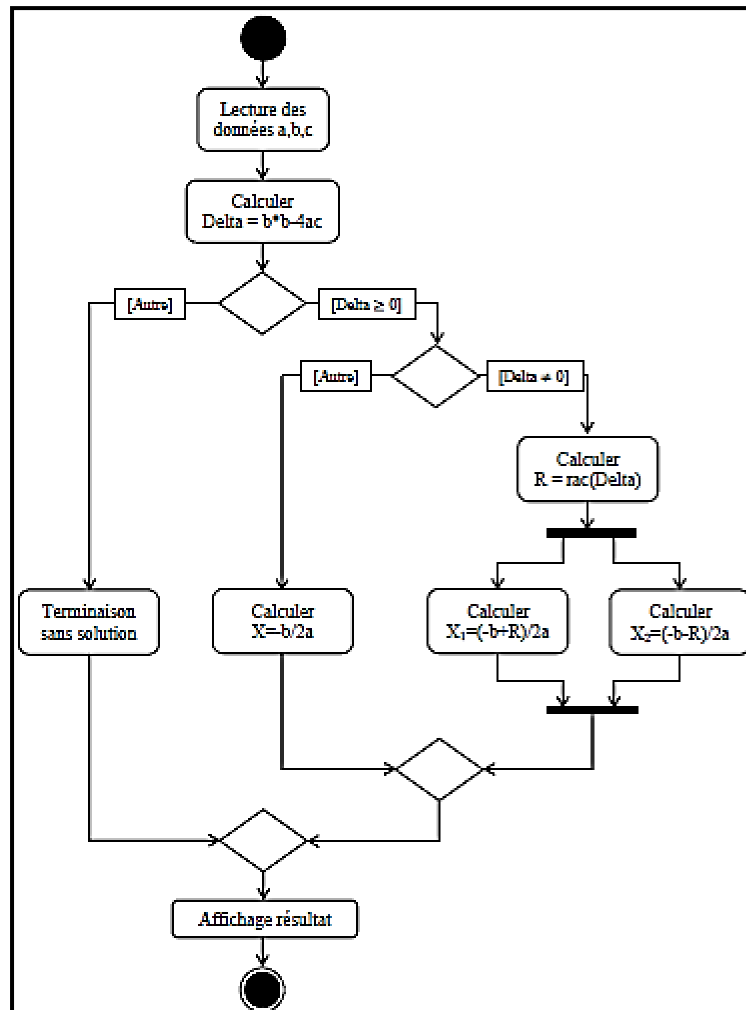


Figure II. 3 : Exemple de diagramme d'activité [48].

II.4.2. Intérêts des diagrammes d'activité

- Représenter graphiquement le comportement interne d'une opération, d'une classe ou d'un cas d'utilisation sous forme d'une suite d'actions.
- Utiliser le mécanisme de synchronisation pour représenter les successions d'états synchrones, alors que les diagrammes d'états-transitions sont utilisés principalement pour représenter les suites d'états asynchrones.
- Utiliser des transitions automatiques évite la nécessité d'existence d'évènement de transition pour avoir un changement d'états.
- Modéliser un workflow dans un cas d'utilisation, ou entre plusieurs cas d'utilisations.
- Définir avec précision les traitements qui ont cours au sein du système, Certains algorithmes ou calculs nécessitent de la part du modélisateur une description poussée.
- Spécifier une opération (décrire la logique d'une opération).

- Le diagramme d'activité est le plus approprié pour modéliser la dynamique d'une tâche ou d'un cas d'utilisation, lorsque le diagramme de classe n'est pas encore stabilisé [49].

II.4.3. Composition d'un diagramme d'activité

Les définitions de cette partie ont été inspirées essentiellement de [50].les éléments qui peuvent être contenus dans un diagramme d'activité sont :

1. Les nœuds (nodes) [51]

1.1. Nœud d'activité (activitynode)

A.* Nœud d'objet *(objectnode)

- Broche (*pin*).
- Nœud paramètre d'activité (*activityparameternode*).
- Nœud central de mémoire tampon (*central buffer node*).
- Nœud d'expansion (*expansion node*).

***B*. Nœud de contrôle (*control node*)**

- Noeud initial (*initial node).*
- Noeud final (*final node).*
- Nœud de fusion ou interclassement (*mergenode*).
- Nœud de décision (*decision node*).
- *N*œud de bifurcation (*fork node*).
- Nœud d'union (*joinnode*).

***C*. Nœud exécutable (*executablenode*).**

1.2. Une partition d'activité (activity partition).

1.3. Une région d'activité interruptible (interruptible activityregion).

1.4. Une région d'expansion (expansion region).

1.5. Une pré-condition ou post-condition locale.

1.6. Un ensemble de paramètres (parameter set).

2. Les arcs (edges)

2.1. Flot de contrôle (*control flow*).

2.2. Flot d'objet (*object flow*).

2.3. Un handler d'exception (*exception handler*).

II.4.3.1. Les nœuds

II.4.3.1.a. Nœud d'activité (*Activity Node*)

- **Une activité**

Une activité est la spécification du comportement paramétré par un séquencement organisé d'unités subordonnées, dont les éléments simples sont les actions. Le flux de contrôle reste dans l'activité jusqu'à ce que les traitements soient terminés, et le flot d'exécution est modélisé par des nœuds reliés par des arcs.

Une activité est un comportement, et à ce titre peut être associée à des paramètres. Une activité regroupant des nœuds et des arcs est appelée **un groupe d'activités**

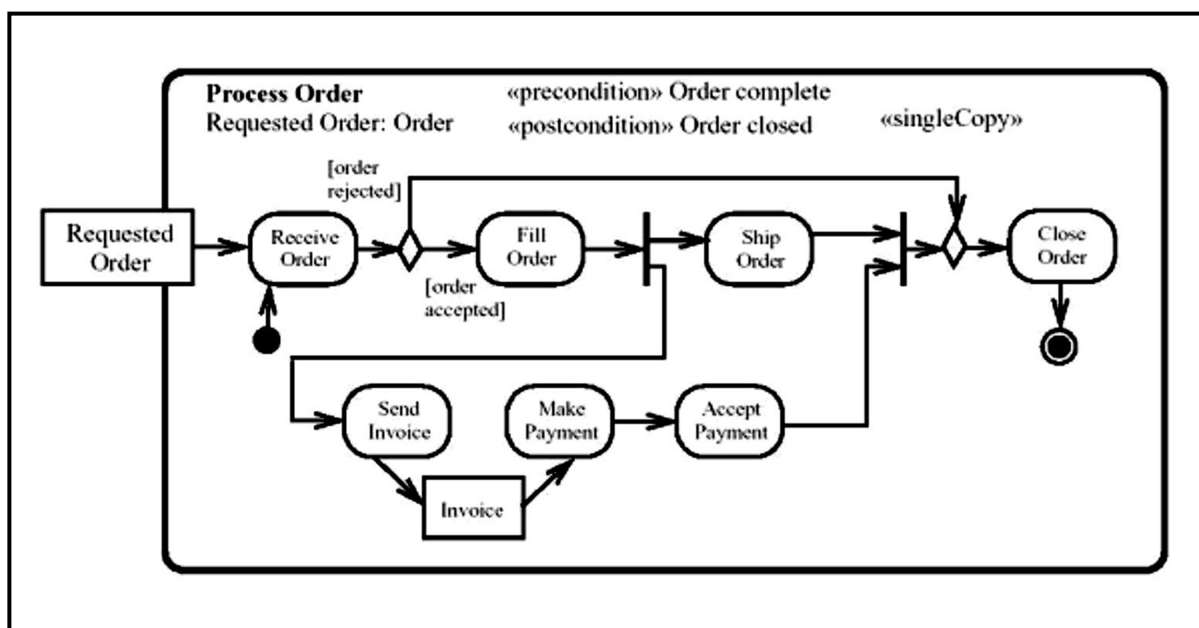


Figure II. 4: Exemple d'activité avec paramètre d'entrée [50].

- **Un nœud d'activité**

Un nœud d'activité est une classe abstraite permettant de représenter les étapes le long du flux d'une activité. Un nœud d'activité peut être l'exécution d'un comportement subordonné,

comme un calcul arithmétique, un appel à une opération, ou la manipulation du contenu d'un objet. Les nœuds d'activité comprennent également le flux de contrôle des constructions, tel que la synchronisation, la décision et la concurrence.

Il existe trois types de nœuds d'activités :

- Les nœuds d'exécutions (executablenode).
- Les nœuds objets (Object node).
- Les nœuds de contrôle (control nodes).

La figure ci-dessus représente graphiquement les nœuds d'activité.

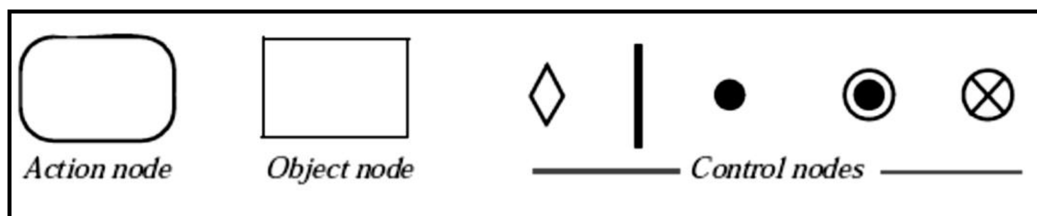


Figure II. 5: Notation nœuds d'activité [51].

On trouve de gauche vers la droite : le nœud d'action, un nœud objet, un nœud de décision ou de fusion, un nœud de bifurcation ou d'union, un nœud initial, un nœud final et un nœud final de flux.

A. Un nœud d'objet *(Object node)*

Un nœud d'objet est une méta-classe abstraite permettant de définir les flux d'objets dans les diagrammes d'activité. Il représente l'existence d'un objet généré par une action dans une activité et utilisé par d'autres actions. La figure 2.6 présente l'arbre de spécialisation du nœud d'objet.

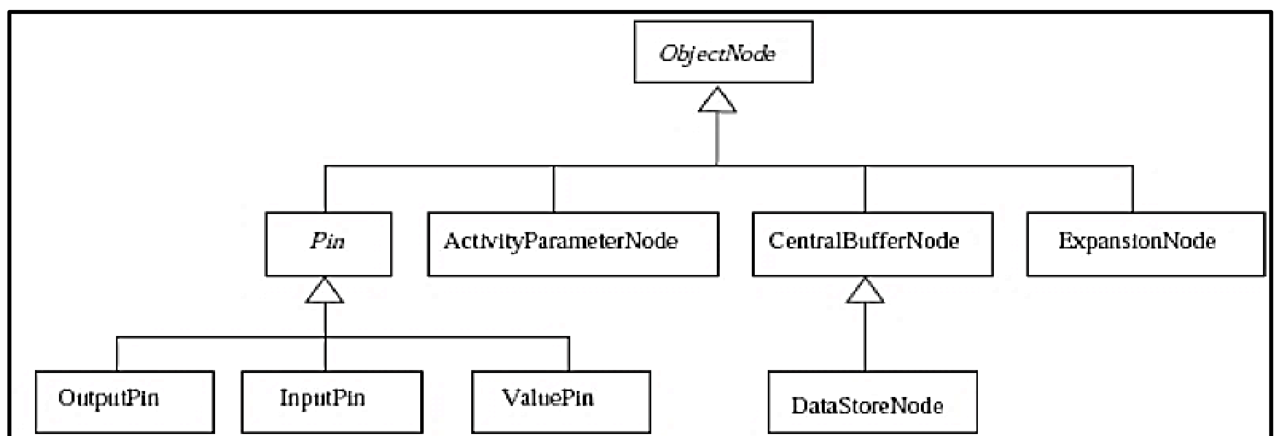


Figure II. 6 : Arbre de spécialisation du nœud d'objet [51].

Un nœud d'objet est noté par un rectangle contenant le nom du nœud. Le nœud d'objet avec un signal comme type, est affiché par le symbole à droite de la figure II.7.

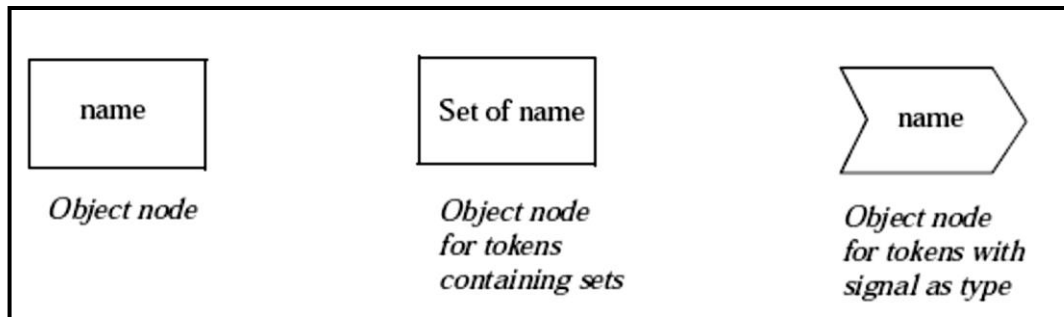


Figure II. 7 : Notation nœud d'objet [51].

A.1. Broche (Pin)

Une broche (*Pin)* est un nœud objet connecté en entrée ou en sortie d'une activité.

Conceptuellement, la notation de nœud objet ne devrait pas avoir d'instances dans les modèles (le nœud d'objet est une classe abstraite), La figure 2.8 donne deux représentations équivalentes de flux d'objets entre deux actions. La première représentation utilise des pins, alors que la deuxième utilise la notation d'un nœud objet.

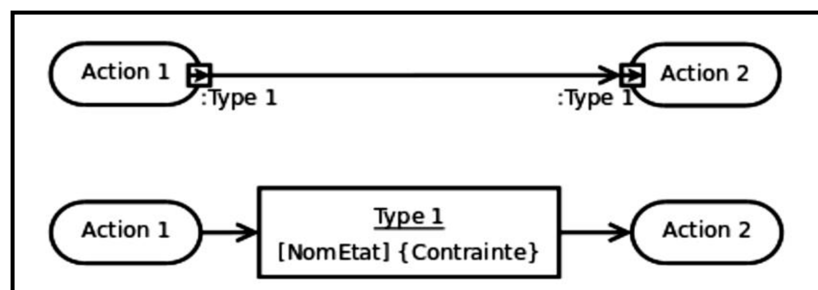


Figure II. 8 : Deux représentations équivalentes pour représenter un flux d'objet [51].

A.2. Nœud paramètre d'activité (*Activity Parameter Node) :*

C'est l'un des nœuds objet, il décrit les entrées ou les sorties des activités. Il est toujours associé avec un paramètre de l'activité.

A.3. Nœud central de mémoire tampon *(central buffer node) :*

Un nœud central de mémoire tampon est un nœud d'objet, destiné pour la gestion des flux provenant de multiples sources. Il peut voir plusieurs arcs entrants et plusieurs arcs sortants.

Graphiquement, on utilise le mot clé <centralBuffer> associé à la notation d'un nœud d'objet.

A.4. Nœud d'expansion (*expansion node*)

Un nœud d'expansion ou *expansion node*est un nœud d'objet qui peut être utilisé pour indiquer un flux à travers les limites d'une région d'expansion. (Paragraphe 2.4.3.1.4)

B. Nœud de contrôle (control node)

Un nœud de contrôle est un nœud d'activités abstrait utilisé pour coordonner les flux entre les d'une activité. La figure ci-dessous présente l'arbre de spécialisation des nœuds de contrôle.

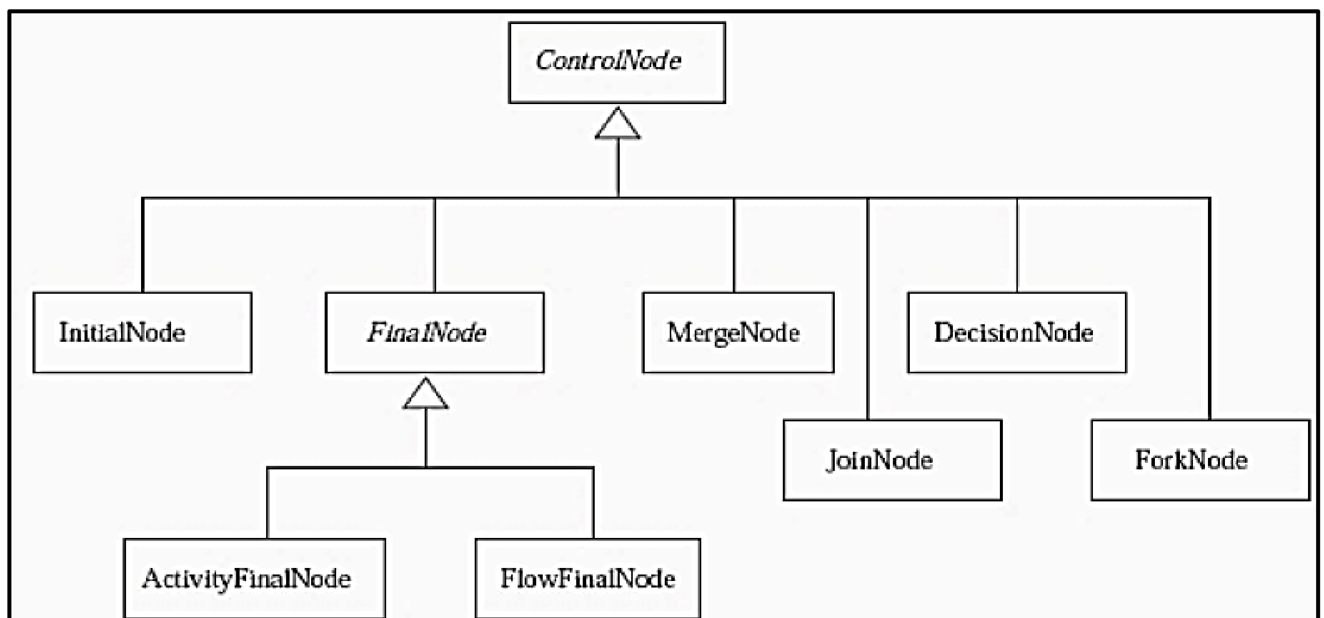


Figure II. 9: Arbre de spécialisation des nœuds de contrôle[51].

Graphiquement, les nœuds de contrôle sont présentés comme suit :

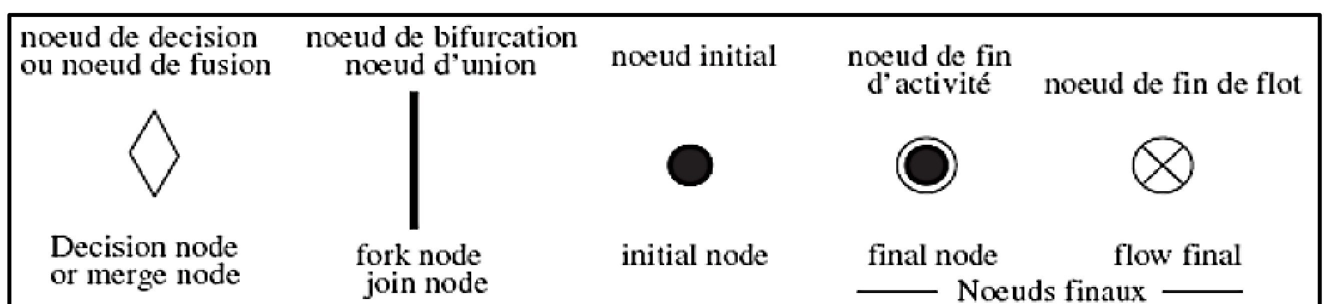


Figure II. 10: Représentation graphique des nœuds de contrôles [51].

B.1. Noeud initial (*initial node*)

Un nœud initial est un nœud de contrôle à partir duquel le flot débute

B.2.Un noeud final (*final node*)

Un nœud final est un nœud de contrôle dans lequel le flux d'activité s'arrête. On peut distinguer deux Types de nœuds finaux :

1. Les nœuds finaux d'activité (activity final node)
2. Les nœuds finaux de flot (flow final node)

B.3. Nœud de fusion (*mergenode)*

Un nœud de fusion est un nœud de contrôle, il rassemble plusieurs flots alternatifs entrants en un seul flot sortant

B.4. Nœud de décision (*decision node*)

Un nœud de décision est un nœud de contrôle, il permet de faire un choix entre plusieurs flux sortants. Les flux sortants sont sélectionnés en fonction de la condition de garde qui est associée à chaque arc sortant.

La notation du nœud de décision est présentée par la figure II.11.

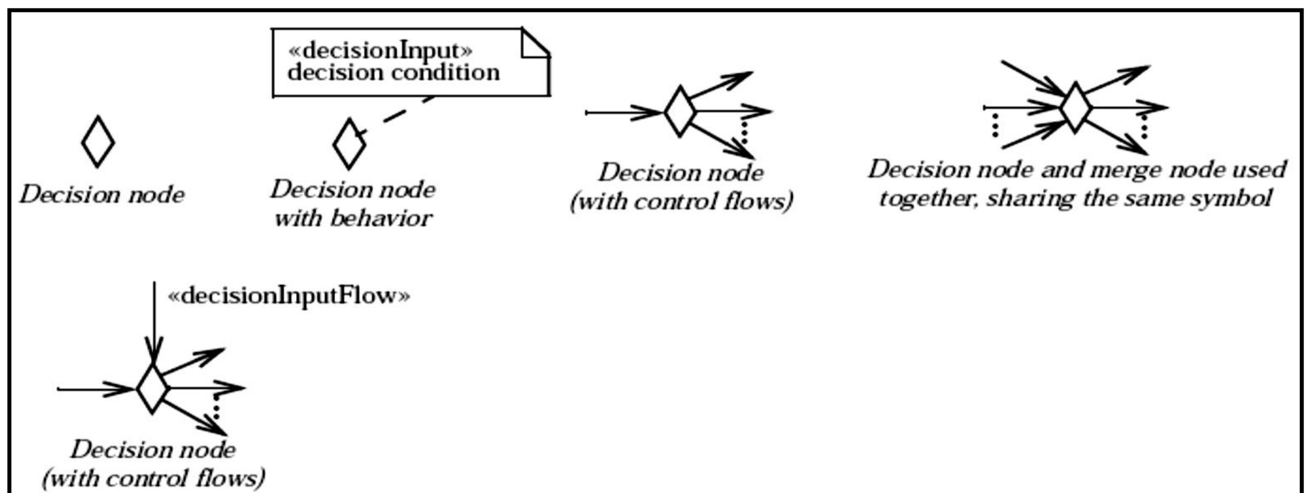


Figure II. 11: Notation nœud de décision[51].

B.5. Nœud de bifurcation (*fork node)*

Un nœud de bifurcation est un nœud de contrôle qui sépare un flux d'entrée en plusieurs flots concurrents en sortie.

B.6. Nœud d'union (*joinnode*)

Un nœud d'union (nœud de jointure) est un nœud de contrôle qui synchronise des flots multiples. Il possède plusieurs arcs entrants et un seul arc sortant. Ce dernier ne peut être activé que lorsque tous les arcs entrants sont activés.

C. Nœud exécutable (*executablenode*)

Un nœud exécutable est une classe abstraite pour les nœuds d'activité qui peuvent être exécutés. Il possède un gestionnaire d'exception qui peut capturer les exceptions levées par le nœud, ou par l'un de ses nœuds imbriqués.

Action :

Une action est un nœud d'activité exécutable, c'est le plus petit traitement qui puisse être exprimé en UML. L'exécution d'une action peut être une transformation ou un calcul dans le système modélisé (affectation de valeur à des attributs, création d'un nouvel objet, calcul arithmétique, émission ou réception d'un signal,…).

Nous citons ci-dessous quelques types d'actions prédéfinis dans la notation UML :

- Action appeler (*call operation*)
- Action comportement (*call behavior*)
- Action envoyer (*send*)
- Action accepter événement (*acceptevent*)
- Action accepter appel (*accept call*)
- Action répondre (*reply*)
- Action créer (*create*)
- Action détruire (*destroy*)
- Action lever exception (*raise exception*)

Une note qui contient des pré ou post-conditions peut être reliée à l'action avec les mots clés <localPrecondition> et <localPostcondition>, respectivement.

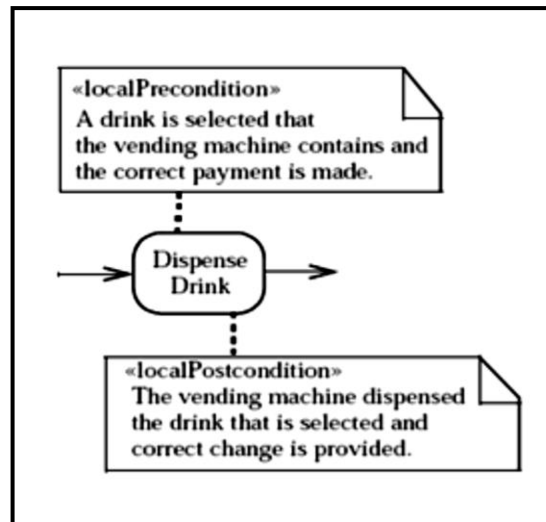


Figure II. 12 : Exemple d'une action avec des pré-conditions et post-conditions [51].

II.4.3.1.b. Partitions d'activité (*activity partition)*

Les partitions d'activité appelées aussi couloirs ou lignes d'eau (*swimlane*), divisent l'espace des nœuds et des arcs afin de montrer explicitement l'entité dans quelle les actions peuvent être effectuées. Cette division permet de faire des regroupements dans les diagrammes d'activité. Une partition peut être décomposée en sous-partitions et regrouper d'autres partitions selon une autre dimension.

Les partitions correspondent souvent à des unités organisationnelles dans un modèle de business, comme le montre l'exemple de la figure suivante.

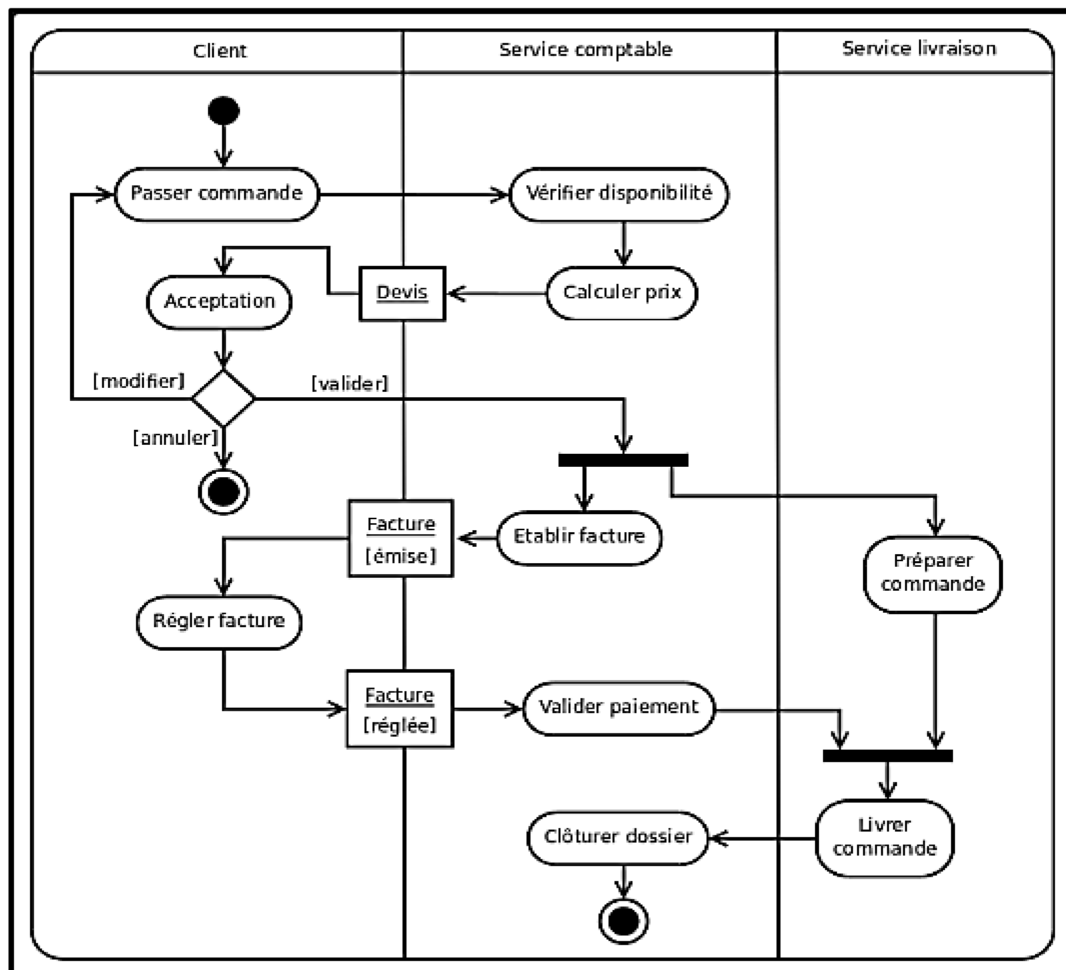


Figure II. 13 : Exemple illustratif (partitions d'activités)[51].

II.4.3.1.c. Région interruptible d'activité (*interruptible activity region)*

Une région interruptible d'activité est un groupe d'activité (regroupement de nœuds et d'arcs), pouvant contenir un arc jouant le rôle d'interrupteur pour cette région. L'activation de l'arc interrupteur implique l'arrêt de l'ensemble des flux dans la région.

II.4.3.1.d. Région d'expansion (*expansion region*)

Une région d'expansion est une région strictement emboitée dans une activité avec des entrées et des sorties sous forme de nœuds d'expansion. Ces derniers représentent une collection d'éléments. La région d'expansion est exécutée pour chaque élément de la collection d'entrée. Dans une région d'expansion, les sorties sont aussi modélisées par des nœuds d'expansion (Figure II.14).

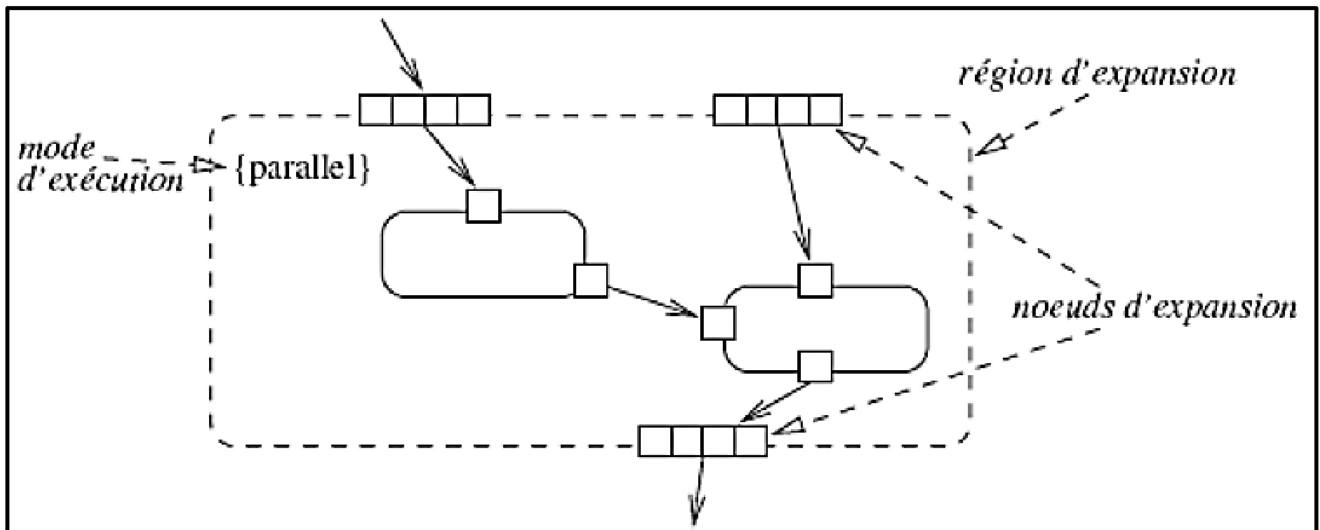


Figure II. 14: Région d'expansion [51].

II.4.3.1.e. Une pré-condition ou post-condition locale *(precondition or postcondition)*

Une pré-condition est un ensemble facultatif de contraintes, en précisant ce qui doit être rempli lorsque le comportement est invoqué. Alors qu'une post-condition est un ensemble de facultatif de contraintes, en précisant ce qui doit être accompli après la fin de l'exécution du comportement.

II.4.3.1.f. Ensemble de paramètres (*parameter set*)

Un ensemble de paramètres est définit comme un élément qui fournit des ensembles alternatifs d'entrées et de sorties nécessaires à un comportement. Chaque ensemble est exclusif des autres ensembles de paramètres du comportement.

II.4.3.2. Les arcs (edges)

2.4.3.2.1. Arc d'activité (*ActivityEdge*)

Un arc d'activité est une connexion dirigée entre deux noeuds d'activité. Si l'arc a un nom, il est noté près de la flèche.

Figure II. 15 : Arc d'activité [51].

II.4.3.2.2. Flux de contrôle (*Control Flow*)

Un flux de contrôle est un arc qui permet de décrire le séquencement de deux nœuds d'activité (un flux de contrôle démarre un nœud d'activité, après la terminaison d'une activité précédente). Il ne transmet pas des données.

Figure II. 16: Notation flux de contrôle [51].

II.4.3.2.3. Flux d'objet (Object Flow):

Un arc de flux d'objets est un arc qui permet de transmettre des données entre des nœuds d'objet.

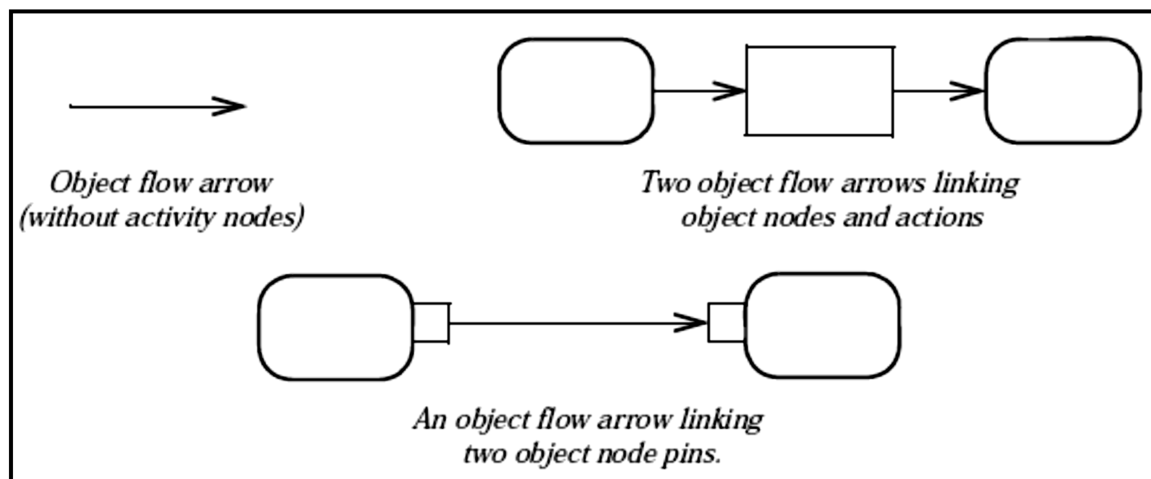


Figure II. 17 : Notation flux d'objet [51].

II.4.3.2.4. Handler d'exception (*Exception Handler*):

Un handler d'exception (appelé aussi gestionnaire d'exception) est une activité qui spécifie un organisme à exécuter. Il possède un pin d'entrée du type de l'exception qu'il gère, et lié à l'activité protégée par un arc. La figure suivante montre les deux représentations possibles du gestionnaire d'exception.

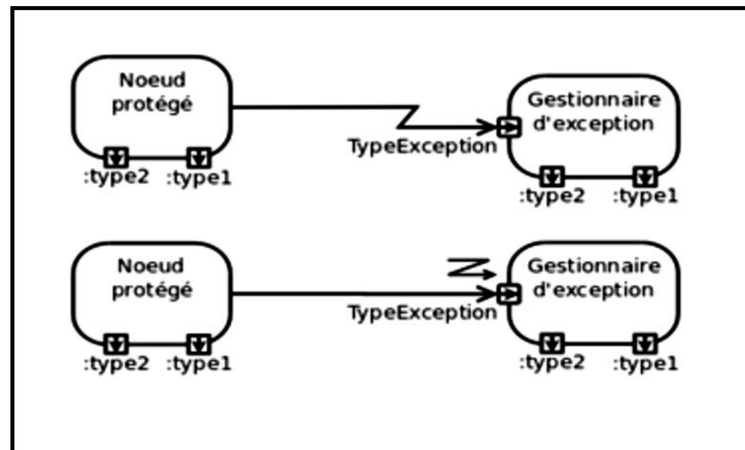
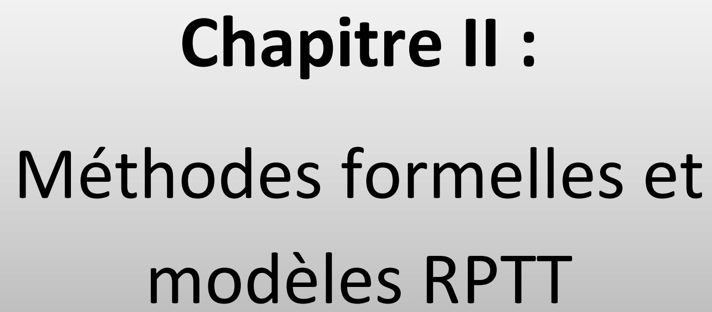


Figure II. 18 : Notation d'un handler d'exception[51].

II.5. Conclusion

Dans ce chapitre nous avons présenté brièvement le langage de modélisation unifié UML, son évolution et ses diagrammes. Ensuite, nous avons donné une description détaillée des diagrammes d'activité d'UML 2.0 qui constituent un outil de modélisation des systèmes Workflows, des modèles orientés service et des processus métiers. Nous avons insisté sur les nœuds d'activité et les arcs, qui seront modélisés dans le formalisme source de la transformation constituant l'objet de notre travail qui sera cité dans le chapitre4.

Dans le chapitre qui suit nous présenterons les réseaux de Pétri temporellement temporisé, nous introduisons les notions afférentes à ce modèle cible de notre étude.



Chapitre II :

Méthodes formelles et modèles RPTT

III. Méthodes formelles et modèles RPTT

III.1. Introduction :

Les méthodes formelles sont de plus en plus utilisées pour répondre aux exigences des systèmes critiques, en mettant l'accent sur une technique formelle couramment utilisée, réseaux de Pétri,. Qui représentent un outil de modélisation universellement connu et reconnu pour les possibilités d'analyse, de validation et de vérification dont ils font preuve. L'exploitation de la théorie associée aux RdP permet, par la recherche des Places et Transitions invariants de répondre à de nombreux problèmes.

En étant rôle d'outil graphique, il nous aide à comprendre facilement le système modélisé, et plus il nous permet de simuler les activités dynamiques et concurrentes. Avec le rôle d'outil mathématique, il nous permet d'analyser le système modélisé grâce aux modèles de graphes, aux équations algébriques.

Dans ce chapitre, nous allons voir les méthodes formelles employées pour prévenir des erreurs de conception des systèmes complexes, ainsi ses avantages, et ses différentes techniques. On présentera par la suite la modélisation par réseaux de pétri, ses propriétés sont ensuite discutées, Le chapitre se termine sur une discussion succincte sur les réseaux de pétri temporellement temporisés.

III.2. Les méthodes formelles :

Les méthodes formelles sont des techniques permettant de raisonner rigoureusement, à l'aide de logique mathématique, sur des programmes informatiques ou du matériel électroniques, afin de démontrer leur validité par rapport à une certaine spécification. Elles sont basées sur les sémantiques des programmes, c'est-à-dire sur des descriptions mathématiques formelles du sens d'un programme donné par son code source.

Ces méthodes permettent d'obtenir une très forte assurance de l'absence de bug dans les logiciels (Evaluation Assurance Level, Safety Integrity Level) (Niveau d'assurance de l'évaluation, niveau d'intégrité de la sécurité). Elles sont utilisées dans le développement des logiciels les plus critiques pour donner une spécification du système que l'on souhaite développer, au niveau de détails désiré. Une spécification formelle du système est basée sur un langage formel dont la sémantique est bien définie, Cette description formelle du système peut

être utilisée comme référence pendant le développement. De plus, elle peut être utilisée pour vérifier (formellement) que la réalisation finale du système respecte les attentes initiales.Leur amélioration et l'élargissement de leurs champs d'application pratique sont la motivation de nombreuses recherches scientifiques en informatique [52].

III.3 Classification des méthodes formelles

Selon J.M. Wing [52], les méthodes formelles peuvent être classées en trois approches:

III.3.1. L’approche axiomatique

Cette approche traite de façon indirecte le comportement du système. Elle est basée sur la construction d’une axiomatisation qui permet d’obtenir les propriétés comportementales. Nous procédons à cette axiomatisation par approches algébriques ou logiques [53].

L’approche logique exprime des systèmes transformationnels avec la logique temporelle. Elle exprime des propriétés dynamiques de sûreté et de vivacité des systèmes réactifs. Dans cette approche, nous nous intéressons à la démonstration de programmes en utilisant les théories de la démonstration automatique ou semi-automatique de théorèmes. Parmi les langages de spécification logiques nous citons: PVS [54], Isabelle/HOL [55] et Coq [56].

L’approche algébrique définit des types abstraits de données en spécifiant pour chaque opération le type de valeurs de ses paramètres et le type de résultat. Les expressions sont précisées à l’aide de variables appelées *termes*, les propriétés des opérations sont décrites sous forme d’équivalences entre ces termes (axiomes). L’application des axiomes à des termes permet d’obtenir d’autres expressions d’équivalence [57]. Parmi les langages de spécification algébrique connus, nous citons: OBJ [58], LPG [59] et Larch[60].

III.3.2. L’approche basée sur les états

Cette approche s’intéresse aux données du système. Elle construit un modèle en termes de structures mathématiques tout en gardant les propriétés du système. Le modèle obtenu servira à l’étude du fonctionnement du système, en lui appliquant des approches dynamiques et des approches par modèle abstrait.

Les approches dynamiques sont basées sur le principe des processus. Elles utilisent les automates, les réseaux de pétri et les algèbres de processus pour spécifier les systèmes de transitions.

Les approches ensemblistes, ou approches par modèle abstrait fournissent une sémantique et une syntaxe du modèle abstrait en utilisant la logique du premier ordre, la théorie des ensembles ou la théorie des types. Parmi les langages de spécification ensemblistes nous trouvons VDM [61], Z [53] et B [57].

La différence entre les approches ensemblistes et les approches algébriques réside dans l'utilisation des types abstraits prédéfinis pour la modélisation des états dans l'approche ensembliste. Chaque opération possède sa propre spécification et son impact sur le système.

III.3.3. L'approche hybride :

L'approche hybride combine l'axiomatisation avec le modèle de données [53]. LOTOS [62] est un bon exemple de cette approche. LOTOS est un langage algébrique car ses expressions de contrôle sont caractérisées par une algèbre dont les termes sont des processus, en même temps, ses données sont caractérisées par une algèbre de type abstrait dont les termes sont des expressions fonctionnelles [57]. CCS [63] et ACT ONE [64] sont des prédécesseurs de LOTOS.

La figure III.1 donne une classification des méthodes formelles selon le type d'approche qu'elles utilisent, axiomatique, basée sur les états ou bien hybride.

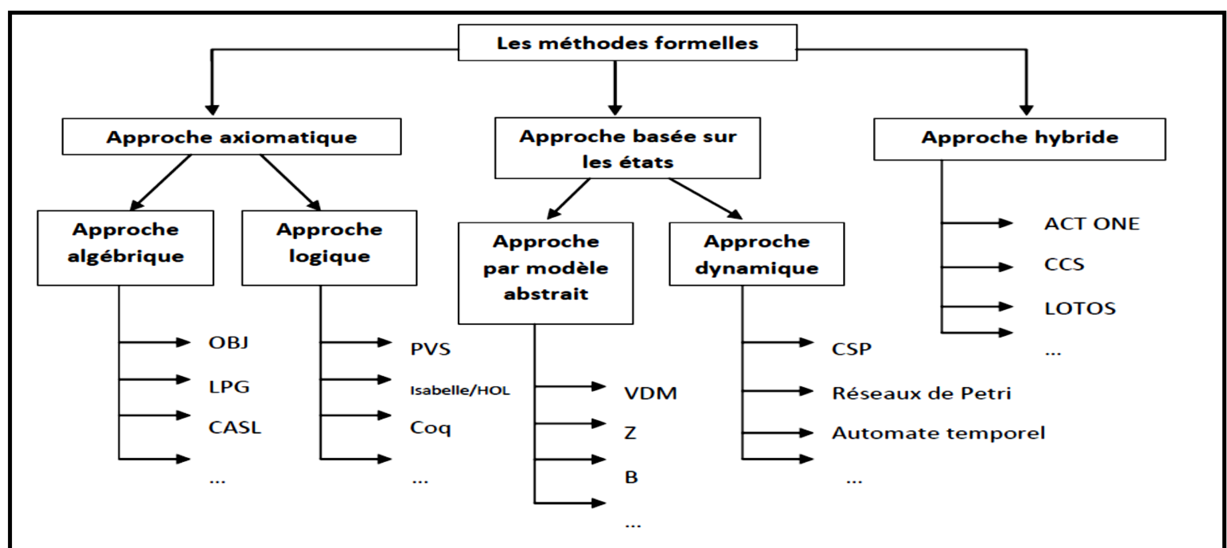


Figure III. 1: Classification des méthodes formelles [51].

III.4. Les avantages des méthodes formelles :

L'avantage principal des méthodes formelles est l'utilisation de concepts de la logique et de la technique mathématique. Ces concepts fournissent des outils effectifs qui organisent les

pensées des concepteurs et qui facilitent la communication entre toutes les personnes concernées par le développement. De plus, ils nous permettent de décrire de manière précise, non ambiguë, les demandes énoncées par l'utilisateur du système logiciel à réaliser. Les notions d'ensemble, de relation, de fonction et leurs différentes propriétés et opérations, avec les quantifications universelles et existentielles, nous permettent d'établir une spécification d'une manière simple et claire et de démontrer mathématiquement les propriétés de la spécification.

Les principaux avantages techniques d'une spécification formelle, par rapport à une spécification informelle, sont la précision et la clarté. Des imprécisions et des ambigüités peuvent facilement se glisser dans les spécifications informelles. Ceci peut ouvrir la voie à plusieurs interprétations. Par contre, les termes de spécifications formelles n'ont qu'une seule interprétation.

Un autre avantage des spécifications formelles est que les questions sont posées et répondues avec précision et d'une manière scientifique. De plus, les méthodes formelles fournissent des spécifications qui peuvent être rigoureusement vérifiées, analysées et testées dès les premières étapes du cycle de développement, ce qui n'est pas le cas dans les méthodes informelles. Cela signifie qu'il possible de détecter et de corriger des fautes dès les premières étapes, ce qui réduit le coût et la durée du développement et améliore la qualité du logiciel.

Les méthodes formelles nous permettent de spécifier ce qui est nécessaire à un niveau d'abstraction particulier. Certains comportements et propriétés peuvent être volontairement exclus s'il est préférable que leurs élaborations soient remises aux prochaines phases du cycle de développement.

Selon Hohan et Stooper,les spécifications rigoureuses jouent un triple rôle dans le développement du logiciel. D'abord, les spécifications documentent avec précision les décisions de conception, indépendamment de l'implantation, et servent de base pour la revue de cette conception. Durant l'implantation, les mêmes spécifications supportent le développement an parallèle. D'une part, elles renseignent les utilisateurs de ce qu'ils peuvent s'attendre et, d'autre part, elles renseignent les programmeurs de ce qu'ils doivent faire, et servent de base pour la phase de test. Finalement, durant la maintenance, ces mêmes spécifications supportent l'analyse de changements et aident à la formation de nouveaux personnels [65].

III.5. Les langages formels :

Un langage formel est défini comme un ensemble de mots de longueur finie obéissant à une sémantique mathématique rigoureuse [66,67], des règles d'interprétation et des règles de déduction.

Les règles d'interprétation garantissent l'absence d'ambiguïté d'interprétation (dans un langage informel ou semi-formel, une description peut avoir plusieurs interprétations).Les règles de déduction raisonnent sur des spécifications afin de détecter les manques et les inconsistances, et aussi pour vérifier des propriétés attendues.

La force des langages formels est de pouvoir faire abstraction de la sémantique, ce qui rend les théories réutilisables dans plusieurs modèles.

III.6. Classification des langages formels :

Plusieurs classifications ont été proposées afin de catégoriser les langages formels selon plusieurs critères. Nous les présentons, ci-après, par ordre chronologique (figure 3.2).

1. [51] a proposé une classification des langages de spécification formels selon la structure du système spécifié. Les langages ont été ainsi classés en langages basés sur les propriétés et langages basés sur les modèles.
2. [68] proposent de classer les langages formels en cinq catégories :
   – Langages basés sur les modèles comme Z et VDM,
   – Langages algébriques comme OBJ [69] et PLUSS [70].
   – L'algèbre de processus comme CSP [71] et CCS [72],
   – Langages basés sur les logiques comme les LTL et CTL,
   – Langages basés sur les réseaux comme les réseaux de Petri
3. [73],quant à lui, trouve que la classification basée sur les modèles/propriétés est confuse, car dans les systèmes réels, les deux classes (basée sur les modèles et basée sur les propriétés) sont utilisées conjointement. Il propose, alors, une nouvelle classification, cette fois-ci, basée sur l'expressivité du langage. Ainsi, les langages formels sont classés en :
   – Langages basés sur l'historique comme les logiques temporelles,
   – Langages basés sur les états comme Z et B,
   – Langages basés sur les transitions comme les automates,

– Langages fonctionnels qui peuvent être algébriques, comme OBJ [69] et ASL [74], ou bien d'ordre supérieur comme HOL [75]et PVS,

– Langages opérationnels comme les réseaux de Petri et l'algèbre de processus.

4. [76] classent les langages formels en : langages basés sur les modèles, langages algébriques et langages déclaratifs. Les langages basés sur les modèles sont divisés en:

– Langages basés sur les états abstraits comme ASM et B,

– Langages basés sur les ensembles et les catégories comme Alloy, Z etet VDM,

– Automates,

– Langages pour les systèmes temps réel, comme Lustre [77] et les automates temporisés.

5. [78] ont repris la classification de [51] et ont ajouté les logiques à la classe des langages basés sur les propriétés (figure 3.2).

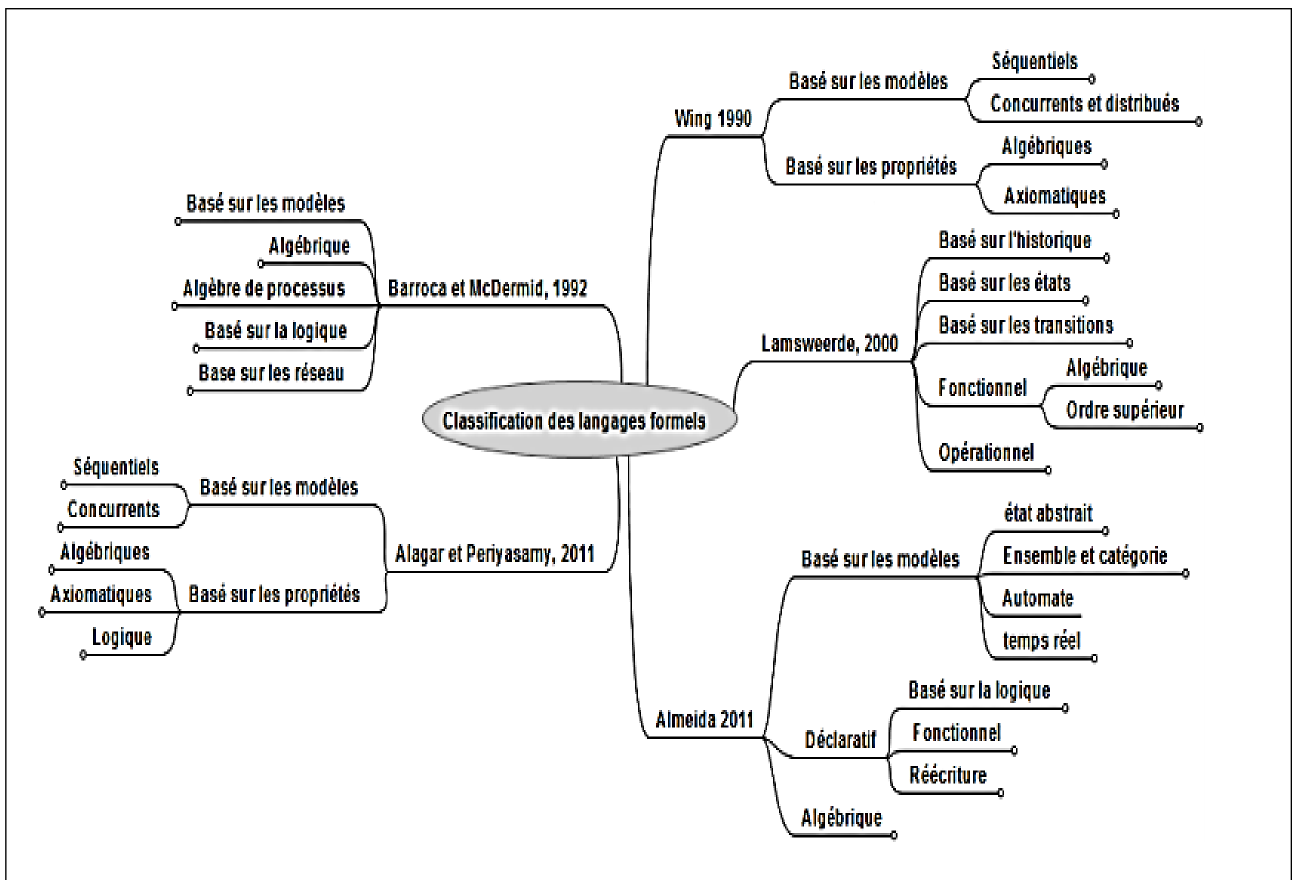


Figure III. 2 : classifications existantes des langages formels [79].

III.7. Technique de vérification formelle :

Les techniques de vérifications formelles ont été développées dans les années 80 afin de s'assurer de l'adéquation entre :

- ✓ Un modèle formel du système de commande qui peut être soit un modèle de conception soit un modèle d'exécution du code implanté dans un automate programmable.
- ✓ Un modèle formel des propriétés a vérifier conformément aux besoins informels(exprimés par les utilisateurs).l'obtention de ce modèle requiert une phase d'identification des exigences de sécurité fonctionnelle et une phase de formalisation pour obtenir les propriétés à respecter.

Il existe plusieurs outils de vérification formelle [80].

III.7.1. Le model checking :

Le model checking [81, 82, 83]est une technique de vérification qui explore tous les états possibles du système. Similaire à un programme d'échecs qui vérifie tous les mouvements possibles.

Le model checking a été appliqué avec succès à plusieurs systèmes et à leurs applications. Il passe par trois différentes étapes : la phase de modélisation, la phase d'exécution et la phase d'analyse [84] et dont l'articulation peut se voir sur la Figure III.4.

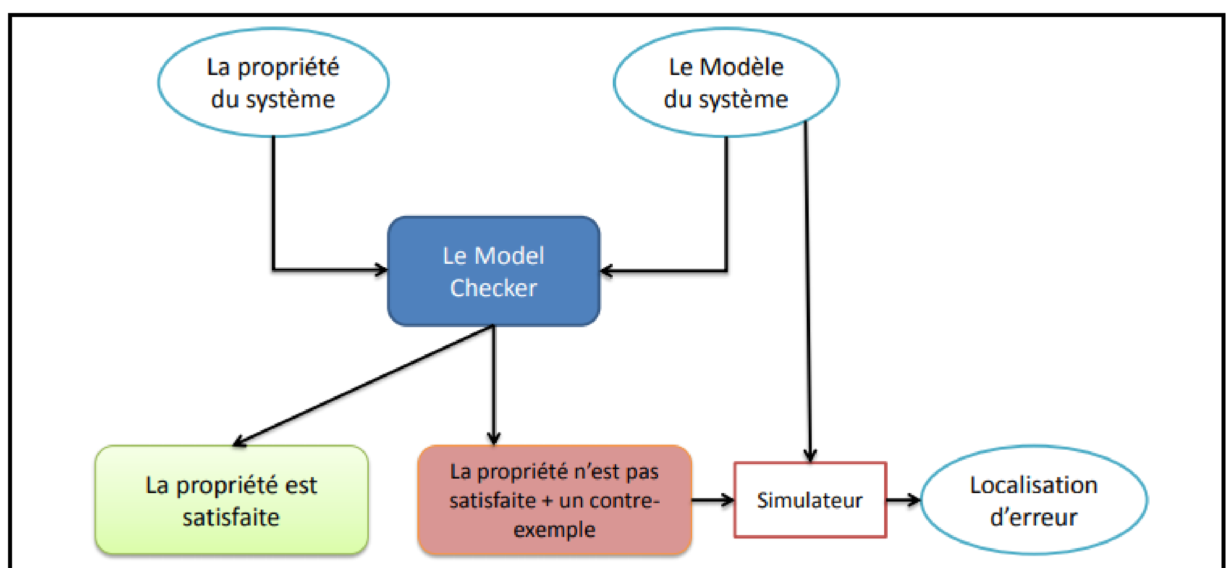


Figure III. 3: L'approche du model checking [81].

III.7.2. Techniques basées sur la simulation :

En pratique, l'une des techniques de vérification la plus connue et utilisée est la simulation [85,86]. Le simulateur permet à l'utilisateur d'étudier le comportement du système. Ceci passe par la détermination, sur la base du modèle du système, des réactions que doit avoir le système vis-à-vis de certains scénarios spécifiques. Ces scénarios sont fournis par l'utilisateur ou générés par des outils tels que les générateurs aléatoires de scénarios.

La simulation est généralement utile pour une première évaluation rapide de la qualité du prototype (l'étape de conception). Elle est cependant moins adaptée à détecter les erreurs subtiles, car pour le simulateur, il est impossible de générer tous les scénarios possibles du système, et encore moins de tous les simuler.

III.7.3. Techniques basées sur le test

Tandis que la simulation et le model checking sont basés sur une description du modèle où tous les états possibles du système peuvent être générés, la technique de vérification basée sur le test [87,88, 89] est applicable dans le cas où il est difficile, voire impossible, d'obtenir un modèle du système à vérifier. Avec le test, des séquences d'actions, représentant différents scénarios d'exécutions possibles, permettent de vérifier une réaction spécifique.

Un paramètre important du test est la mesure à laquelle l'accès à l'état interne du système testé peut être obtenu. Deux types de tests peuvent être effectués sur le système. Le test à boite blanche peut accéder totalement à la structure interne d'une implémentation, tandis que dans le test à boite noire, la structure interne est complètement cachée.

Les tests sont plutôt improvisés et pas très systématiques. Comme résultat, le test est une activité très élaborée, prédisposant aux erreurs, et difficilement gérable. Similaire au model checking.

III.7.4. Techniques basées sur la preuve de théorème :

La preuve de théorème, [90,91] nécessite que le système soit spécifié sous forme d'une théorie mathématique, ou devrait être transformé en une telle forme. En utilisant un ensemble d'axiomes (le théorème de base), un démonstrateur (le logiciel) tente de construire : soit une preuve de théorème en générant les étapes de preuves intermédiaires ; soit de réfuter les axiomes énoncés. Les axiomes sont intégrés ou fournis par l'utilisateur. Les démonstrateurs sont

également appelés assistants de preuves. La demande générale de prouver des théorèmes d'un type assez général et l'utilisation de logiques indécidables exige certaines interactions avec l'utilisateur.

Le principal avantage de la preuve de théorèmes est qu'il peut agir avec des espaces d'états infinis et peut vérifier la validité des propriétés pour des valeurs de paramètres arbitraires. Par contre, l'inconvénient principal de la preuve de théorèmes est la lenteur du processus de vérifications, le risque d'erreurs et le temps utilisé pour guider la preuve. Aussi, la logique mathématique utilisée par l'assistant de preuves exige un degré assez élevé d'expertise des utilisateurs.

III.8. Les Réseaux de Pétri :

Les réseaux de pétri ont été introduits par Carl Adam Petri en 1962 dans sa thèse de doctorat [92] à l'université de Bonn, Allemagne. Ce travail a ouvert un champ d'étude et a été exploité par Anatol W. Holt, F. Commoner, M. Hack et leurs collègues dans un groupe de recherche du Massachussetts Institute of Technology (MIT), dans les années 70 et c'est en 1975 que la première conférence sur les réseaux de pétri et les méthodes relationnelles a été organisée au MIT.

Un réseau de pétri est un outil graphique pour la description formelle des systèmes dont la dynamique est caractérisée par la concurrence, la synchronisation, l'exclusion mutuelle et le conflit des choix multiples. Ces attributs sont propres aux environnements distribués. La facilité d'adaptation des réseaux de pétri élargit leur champ de pratique jusqu'aux protocoles de communication, architectures des ordinateurs, etc. Leur aspect mathématique leur permet l'analyse des propriétés comportementales et structurelles essentielles à la validation du système.

III.8.1 Présentation des réseaux de Petri :

Un réseau de Petri*(RDP)* est un graphe biparti orienté value. Il a deux types de nœuds [93] :

1. ***les places*** **:** notées graphiquement par des cercles. Chaque place contient un nombre entier (positif ou nul) de marques (ou jetons). Ces derniers sont représentés par des points noirs.

2. ***les transitions*** **:** notées graphiquement par un rectangle ou une barre. Une transition qui n'a pas de place en entrée est appelée transition source et une transition qui n'a pas de place en sortie est appelée transition puits.

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

- *Un arc* relie, soit une place à une transition, soit une transition à une place mais jamais une place à une place ou une transition à une transition.
- Chaque arc est étiqueté par une valeur (ou un poids), qui est un nombre entier positif. L'arc ayant *k* poids peut être interprété comme un ensemble de *k* arcs parallèles. Un arc qui n'a pas d'étiquette est un arc dont le poids est égal à 1.

La figure III.4 illustre la notation graphique d'un Réseau de Petri.

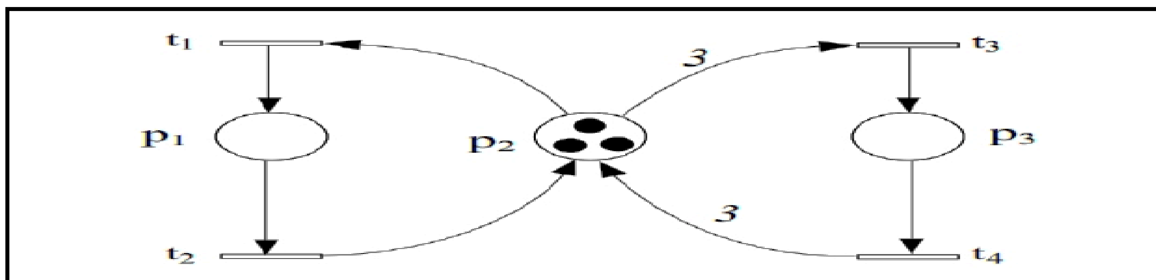


Figure III. 4:Exemple d'un Réseau de Petri[93].

III.8.2 Définition Formelle

Formellement, un Réseau de Petri marqué est un 5-uplet, *Rdp*= (*P, T, F, W ,M*0)où [93] :

- $P = \{P1, P2, \cdots, Pm\}$ est un ensemble fini non vide de places P.
- $T = \{t1, t2, \cdots, tn\}$ est un ensemble fini non vide de transitions T.
- $F \subseteq (P \times t) \cup (T \times P)$ est un ensemble d'arcs où :
  - ⇨ $(P \times T)$ est l'arc allant de *P* à *T*.
  - ⇨ $(T \times P)$ est l'arc allant de *T* à *P*.
- $W : F \rightarrow \{1, 2, 3, \cdots\}$ est une fonction du poids où :
  - ⇨ $W(P, T)$: "*Pré (p, t)*" est le poids de l'arc allant de *P* à *T*.
  - ⇨ $W(T, P)$: *"Post (p, t)*" est le poids de l'arc allant de *T* à *P*.
- $M0 : P \rightarrow \{0, 1, 2, 3, \cdots\}$ est le marquage initial.
  - ⇨ $P \cap T = \emptyset$et $P \cup T\ 6 \neq \emptyset$

III.8.3 Marquage d'un Réseau de Pétri

Un marquage est dénoté par un vecteur du nombre de jetons dans chaque place : la $i^{ième}$ composante correspond au nombre de jetons dans la $i^{ième}$ place [94].

Le marquage de réseau de pétri représenté par la figure III.5b est donc

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

Un réseau de Petri $N = (P, T, F, W)$ est un réseau de pétri sans marquage initial et un réseau de Petri *Rdp* avec marquage initial *M*0 est *Rdp*= (*N, M0*) [94].

Le réseau de pétri représenté par la figure III.5a est un réseau de pétri non marqué alors que le réseau de pétri représenté par la figure III.5b est un réseau de pétri marqué

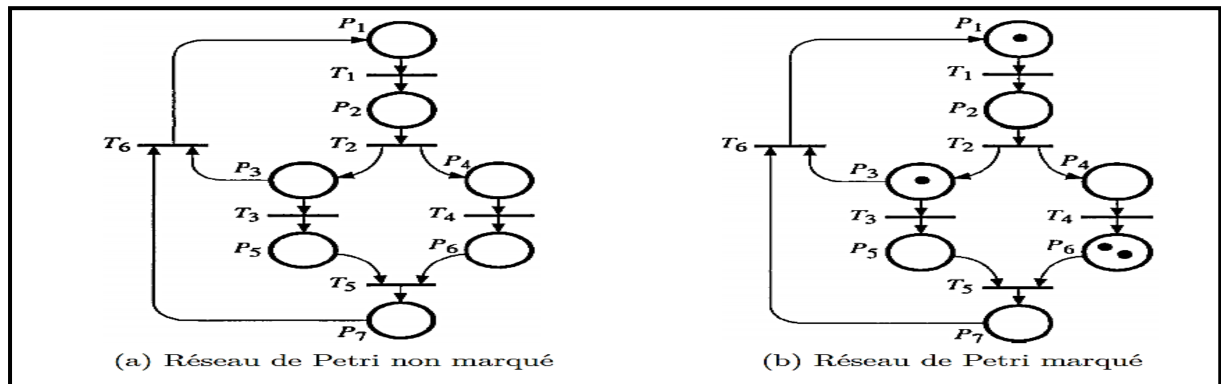


Figure III. 5 : Exemple de Réseau de Petri [94]

III.8.4 Évolution d'un Réseau de Petri

L'évolution d'un Réseau de Petri correspond à l'évolution de son marquage au fil du temps (évolution de l'état du système) : il se traduit par un déplacement des jetons pour une transition *t* de l'ensemble des places d'entrée vers l'ensemble des places de sortie de cette transition. Ce déplacement s'effectue par le franchissement de la transition *t* selon des règles de franchissement [94] qu'on va voir dans la section III.8.4.2.

III.8.4.1 Transition validée

On dit qu'une transition est validée si toutes les places en entrée de celle-ci possèdent au moins une marque. Une transition source est par définition toujours validée [93].

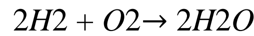
III.8.4.2 Règle de Franchissement

Si la transition est validée, on peut effectuer le franchissement de cette transition : on dit alors que la transition est franchissable [93].

Le franchissement consiste à :

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

La règle de franchissement est illustrée par la figure III.3 en utilisant la réaction chimique connue [93] :



$$2H2 + O2 \rightarrow 2H2O$$

La présence des deux jetons dans chaque place d'entrée (figure 3.6a), indique que 2 unités de H_2 et 2 unités d'O2 sont disponibles, donc la transition t est franchissable. Après avoir franchi t, le marquage va changer et on obtient le réseau ayant le marquage comme celui de la figure III.6b, maintenant t n'est plus franchissable.

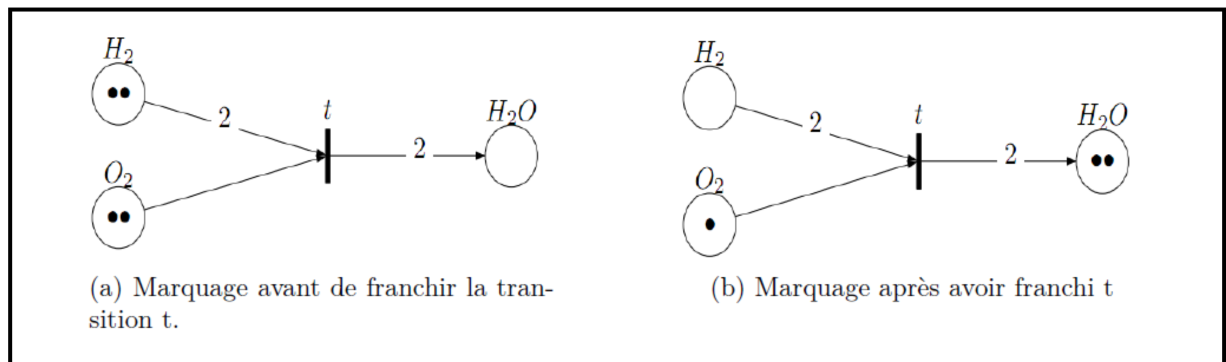


Figure III. 6: Exemple de règle de franchissement de transition [93]

III.9 Modélisation Avec les Réseaux de Petri

Les réseaux de Petri ont été conçus et utilisés principalement pour la modélisation. Plusieurs systèmes peuvent être modélisés par les réseaux de Petri, ces derniers peuvent être de natures très diverses : matériel informatique, logiciels informatiques, les systèmes physiques, les systèmes sociaux, etc [95].

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

III.9.1 Parallélisme

Dans le réseau de pétri représenté par la figure III.7 le franchissement de la transition *T1* met un jeton dans la place *P2* (ce qui marque le déclenchement du processus 1) et un jeton dans la place *P2* (ce qui marque le déclenchement du processus 2)

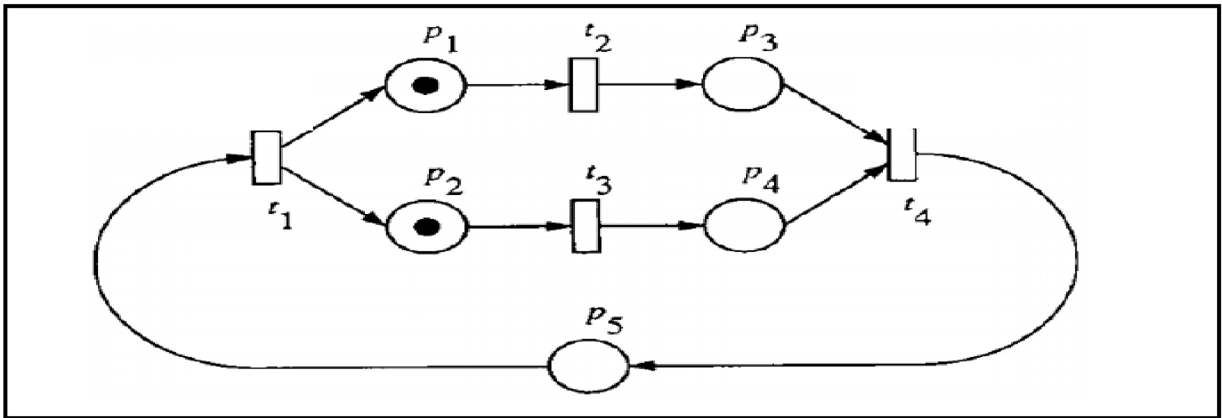


Figure III. 7: Parallélisme dans les Réseaux de Pétri [93]

III.9.2 Synchronisation

Les réseaux de pétri ont été utilisés pour modéliser une variété de mécanismes de synchronisation, y compris les problèmes de l'exclusion Mutuelle, producteur/consommateur, lecteurs/écrivains... etc [93].

III.9.2.1 Exemple 1 : Problème du producteur/consommateurs

Considérons le problème du producteur-consommateur où le producteur ne reprend que si le Buffer est vide. La transition produire ne pourra être tirée que si la place buffer ne contient pas de jetons.

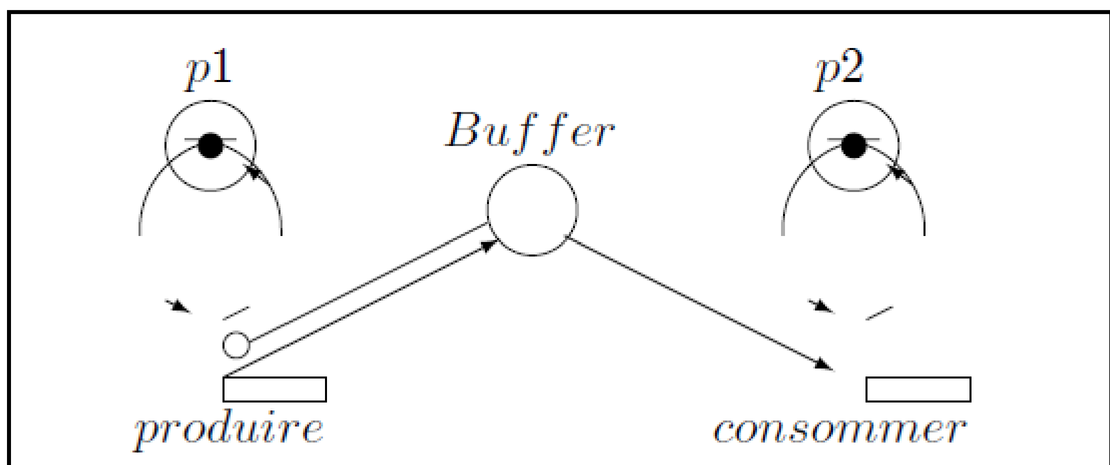


Figure III. 8 : Problème du producteur et consommateurs [95]

III.9.3 Calcul De Flux De Données

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

Les réseaux de pétri peuvent être utilisés pour représenter non seulement le flux de contrôle, mais également le flux de données. Les jetons dénotent les valeurs des données en cours ainsi que la disponibilité des données [93].

Le réseau de Petri de la figure III.9 représente un calcul de flux de données de la formule suivante [93] :

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

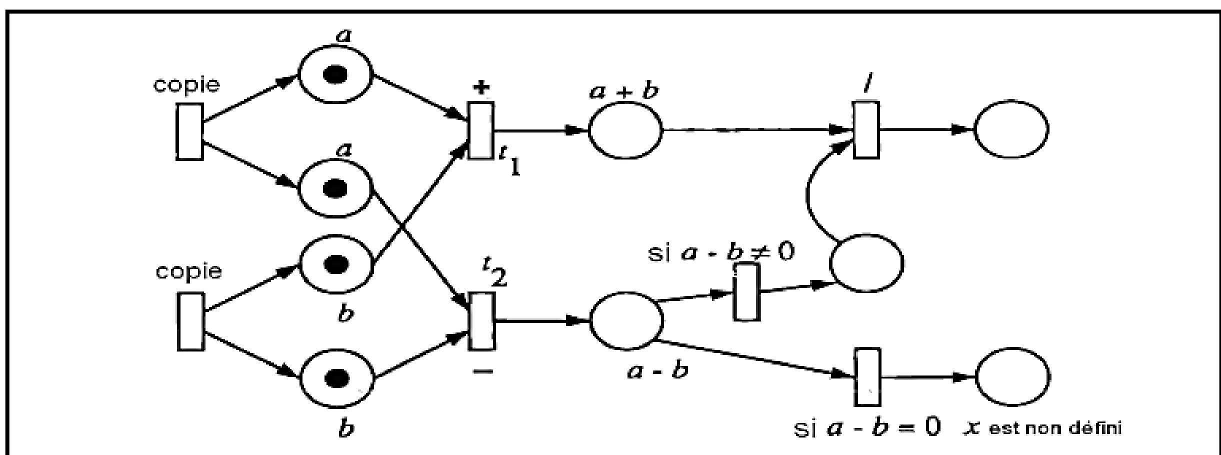


Figure III. 9: Exemple d'un calcul de flux de données par un Rdp [93].

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

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

III.9.4 Principales Propriétés des Réseaux de Petri

a. Accessibilité

L'accessibilité est une propriété fondamentale pour étudier les propriétés dynamiques du système. Le franchissement d'une transition validée va changer la distribution des jetons sur le réseau selon les règles définies dans la section III.9.4.2. Une séquence de franchissements αentraîne une séquence de marquages [93].

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

$$\alpha = M0t1M1t2M2 \cdot\cdot\cdot tnMn$$

Ou simplement :

$$\alpha = t1t2 \cdot\cdot\cdot tn$$

Dans ce cas M_n est accessible à partir de M_0 par αet on écrit :

$$M0[\alpha]> Mn$$

L'ensemble de séquences de franchissements à partir de M_0 dans un réseau ($N,M0$)est noté par $L(N,M0)$ ou simplement $L(M0)$.

L'ensemble de marquages possibles accessibles à partir de M_0 dans un réseau ($N,M0$)est noté par $R(N,M0)$ ou simplement $R(M0)$.

b.Bornitude et RDP Sauf

Un réseau de Petri *Rdp*= ($N,M0$) est *K-borné* ou simplement *borné* si le nombre de jetons dans chaque place ne dépasse pas un nombre fini K pour tout marquage accessible depuis $M0$[93].

C'est- à-dire :

$$\forall M \in R(M0), \forall p \in P: M(p) \leq k$$

Un réseau de pétri marqué *Rdp*= (N,M_0) est sauf si et seulement s'il est *1-borné* [93].

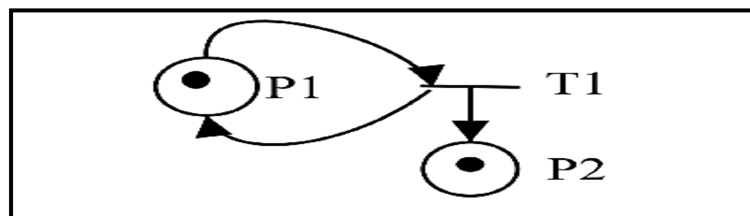


Figure III. 10 : Exemple d'un réseau de Petri non borné [96]

Dans l'exemple de réseau de Petri représenté par la figure 3.10 la transition *T1* admet la place *P1* comme l'unique place d'entrée. La place *P1* a un jeton : la transition *T1* est franchissable. Comme *P2* est aussi place de sortie de *T1*, le franchissement de *T1* ne change pas le marquage de *P2*. La transition *T1* est donc franchissable en permanence et peut donc être franchie un nombre de fois infini. Chaque franchissement de *T1* ajoute un jeton dans la place *P2*, le marquage de celle-ci peut donc tendre vers l'infini [96]. Par contre le réseau de pétri de la figure III.7 est borné [93].

c. Vivacité

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

c.1 Transition Vivante

Une transition T_j est vivante pour un marquage initial M_0 si pour tout marquage accessible

$M_i \in R(M_0)$, il existe une séquence de franchissements α à partir de M_i contenant T_j. c.à.d. quel que soit l'évolution du réseau à partir du marquage initial, le franchissement de cette transition est toujours possible [94].

Les transitions *T2* et *T3* du réseau de Petri marqué représenté par la figure III.11b ne sont pas vivantes et les transitions *T1* et *T2* du réseau de Petri marqué représenté par la figure III.11a sont vivantes

c.2 RDP Vivant

Un réseau de Petri$Rdp= (N,M_0)$ est vivant si toutes ses transitions sont vivantes [94].Le réseau de Petri marqué représenté par la figure 3.11a est vivant, alors que le réseau de pétri représenté par la figure III.11b n'est pas vivant.

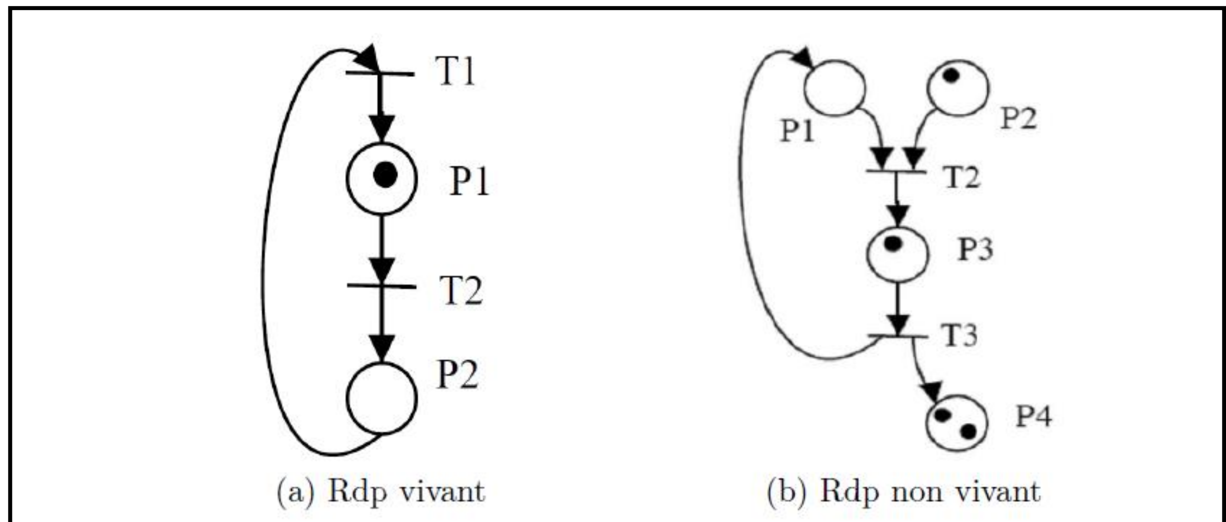


Figure III. 11 : Exemple de vivacité des Réseau de Petri [96]

d. Blocage

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

Un réseau est dit sans blocage si tout marquage accessible depuis M_0 n'est pas un marquage "puits" [94].

Le réseau de Petri marqué représenté par la figure 3.11b a pour blocage le marquage :

$$M3 = [1, 0, 0, 4]$$

e.Réinitialisable et État d'accueil

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

Un marquage M_0 est État d'accueil si pour chaque marquage M dans $R(M_0)$, M est accessible à partir de M_0 [93].

La figure III.12 représente un réseau de pétri réinitialisable.

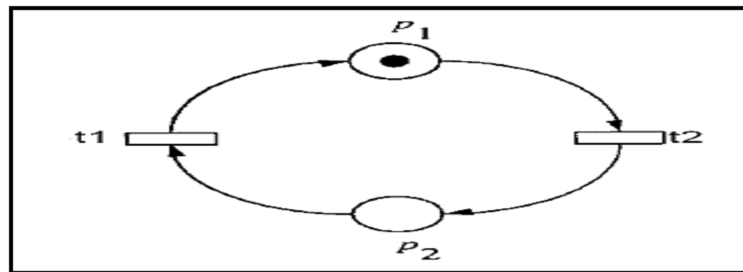


Figure III. 12 : Exemple d'un réseau de Pétri réinitialisable [93]

f. Couverture

Un marquage *M* dans un réseau de Petri (N,M_0) est dit couvrable s'il existe un marquage *M'* dans $R(M_0)$ tel que $M'(p) \geq M(p)$ pour chaque place du réseau [93].

g. Persistance

Un réseau de Pétri (N,M_0) est dit persistant si pour n'importe quelles deux transitions, le franchissement d'une transition ne doit pas inhiber l'autre transition. Si une transition dans un réseau de Pétri persistant est une fois validée, elle reste validée jusqu'à son franchissement [93].

La figure III.13 représente un réseau de Petri persistant.

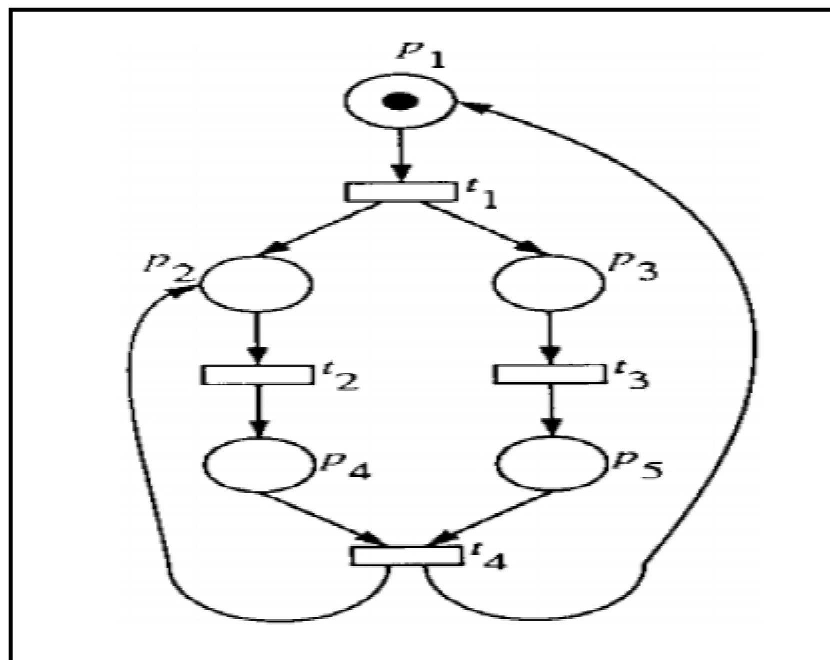


Figure III. 13 : Exemple d'un réseau de Pétri persistant [93]

III.10 Les Réseaux de Pétri de Haut Niveau

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

III.10.1 Réseau de pétri Coloré

Les réseaux de pétri colorés ont été introduits afin de permettre la modélisation de systèmes ayant des comportements symétriques avec des réseaux de taille réduite et sans perte d'informations.

En effet, les jetons appartiennent à des domaines de couleurs qui permettent de les distingués. Ainsi, à chaque place est associé un domaine de couleur qui permet de différencier les jetons que peut prendre cette place. Et à chaque transition et aussi associé un domaine de couleur afin de choisir les couleurs de franchissement.

Les préconditions deviennent des applications qui associent à chaque couleur tirée les couleurs qui seront prélevées des places en entrée.

Les post-conditions deviennent des applications qui associent à chaque couleur tirées les couleurs qui seront déposés dans les places en sortie.

- **Exemple** Partage de ressources

Considérons le cas de partage en exclusion mutuelle de deux ressources par deux processeurs, Ce réseau devient encore plus complexe avec n > 2 processus partageant m > 2ressources critiques. A l'aide des réseaux de pétri coloré ce modèle sera plus compact.

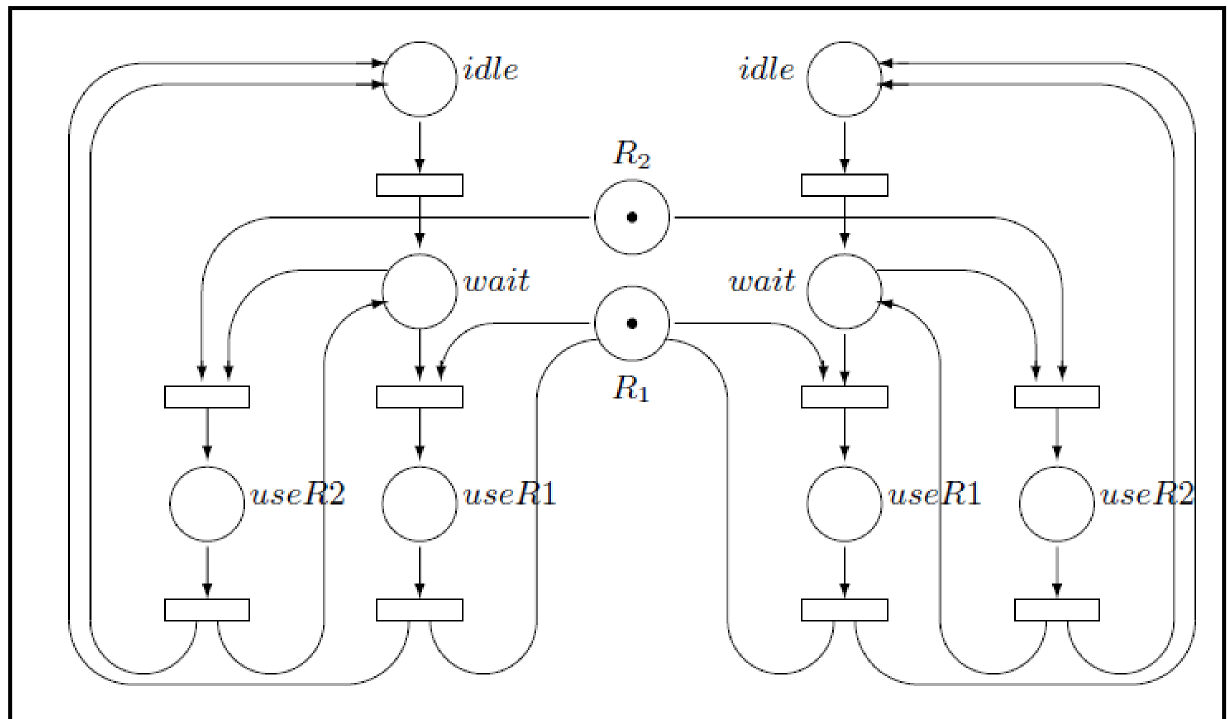


Figure III. 14 : Exemple de partage de ressources dans un réseau de Petri [95]

III.11. Réseaux de Pétri temporellement temporisés (RPTT)

Après la réussite de définir une sémantique de maximalité pour les réseaux de pétri dans la question est étendue vers les réseaux de pétri à gestion du temps explicite. Au cours des années présidentes, de nombreux modèles de réseaux de pétri à gestion du temps ont été proposés et comme les modèles étudiés (réseaux de pétri temporels, ou temporisés) sont incapables de modéliser les systèmes à temps contraintes avec durées d'action. On trouve dans la définition d'un nouveau modèle capable de modéliser ces systèmes c'est RPTT (Réseau de pétri Temporellement Temporisé pour) modèle qui regroupe les deux aspects : temporel et temporisé, nous allons le présenter dans ce qui suit, ainsi sa sémantique de maximalité opérationnelles [97].

III.11.1. Principe des réseaux de pétri temporellement temporisé

L'idée fondatrice des réseaux de pétri temporellement temporisés (pour Duration Action Time pétri Nets, en anglais) est d'associer deux dates *min* et *max* à chaque transition, c'est l'intervalle de tir de cette transition. Cet intervalle représente une latence durant laquelle la transition peut être tirée. Quoique le tir de la transition soit instantané, la durée d'exécution de l'action associée à cette transition peut avoir une durée non nulle. A titre d'exemple, soit une transition « *t* » dont l'action associée est de durée *d* ; cette transition a été sensibilisée à la date

«Θ», alors « *t* » ne peut être tirée avant la date «Θ+ *min* » et doit être tirée au plus tard à la date «Θ+ *max* », à moins que « t » ne soit désensibilisée par le tir d'une autre transition.[97]

Dans le cas où cette transition est tirée, l'action associée commence son exécution et elle dure unités du temps.

Soit le tir de «t » à la date Θ *avec* $\Theta+min <= \Theta <= \Theta+max$, l'action associée se termine à la date $\Theta+d$. Le tir de la transition marque le début d'exécution de l'action associée.

Dans un réseau de pétri temporellement temporisé le passage des jetons de l'état indisponible à l'état disponible est conditionné par l'écoulement de la durée de l'exécution de l'action associée (respectivement par le tir de la transition). Un jeton déposé dans une place *p* (partie droite) à la date ϑ passe de l'état indisponible à l'état disponible à la date $\vartheta+d$, le jeton est lié au tir de la transition durant l'intervalle $[\vartheta, \vartheta + d]$ et il devient libre à l'instant $\vartheta + d$ (devient dans la partie droite de cette place).

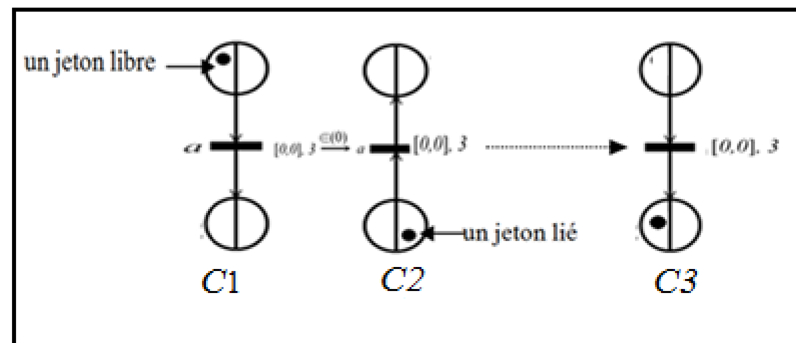


Figure III. 15 : Le passage d'un état à un autre pour le jeton [97]

Le jeton qui se trouve dans la place amont de la transition n'est lié à aucune transition, ce jeton est libre dans cet état-là. Dans le cas où la transition se tire, l'action associée au tir de cette transition a commencé son exécution, ce qui est marqué par la présence du jeton dans la place aval figure 3.15(C1), De ce fait, le jeton dans cette place est lié au tir de la transition, mais après l'achèvement de, après 3 unités du temps, le jeton deviendra libre figure3.5 (c2).

III.11.2 La dépendance causale des transitions

Si une transition « *t'* » dépend causalement de la transition « *t* », « *t'* » ne sera sensibilisée qu'après l'achèvement de l'action associée à « t », c'est-à-dire après les *d* unités du temps qui suivent le moment de tir de la transition « *t* ». [97] Pour capturer cette notion de dépendance causale entre les tirs des transitions, les jetons produits par le tir d'une transition sont dits *liés* à cette transition et ce tout au long de la durée de l'exécution de l'action associée.

III.11.3 Définition formelle des réseaux de Petri temporellement temporisés

Soit 𝕋 un domaine temporel. Un RPTT sur 𝕋 et de support Act, est un tuple N=(P, T,F,λ SIM,π) [97] tel que :

- N=(P,T,D,B) est un RDP marqué
- Un alphabet *Act* est un ensemble fini; nous supposons que π∈*Act* (désignera l'action invisible, *t* dite aussi action silencieuse ou interne).
- L'étiquetage d'un RPTT est une fonction$\lambda: T \rightarrow Act \cup \{\pi\}$. Si alors t dite observable ou externe ; dans le cas contraire, t dite silencieuse ou invisible.
- SIM :×𝕋∞est la fonction qui associée à chaque transition un intervalle statique de tir**.**
- π : *Act*→*D* est la fonction de durée statique, qui à chaque action associe sa durée statique
- Soit l'ensemble de tous les intervalles d'un RPTT tel que :

I(t)= [*min* ,*max*] est l'intervalle associé à la transition et on note par :

↑I(t)= min, ↑ I(t) =max les fonctions qui donnent respectivement la borne inférieure et la borne supérieure d'un intervalle.

III.11.4 Des sous-classes du RPTT

Il est clair que le fait d'associer deux dates *min* et *max* à chaque transition (dates qui représentent une latence) avec une durée fixe de l'action associée, donne une intuition que ces réseaux sont une extension native des réseaux de pétri t-temporisés , dans un contexte d'une sémantique qui oblige le tir des transitions (par notion de sémantique forte). Donc on peut décrire n'importe quel réseau t-temporisé par un RPTT. Le fait d'associer l'intervalle [0, +∞ [à toute transition du réseau de pétri t-temporisé permet de le voir comme un RPTT. Etant donné que les deux extensions temporisées des réseaux de pétri sont équivalentes (Réseaux de pétri t-temporisé et p-temporisés), les RPTTs sont encore une généralisation des réseaux de Petri p-temporisés.

Le fait que toutes les actions d‟un RPTT sont instantanées, ce dernier est vu comme un réseau de Petri temporel (T-TPN). Donc les réseaux de Petri temporels (T-TPNs) sont simulés par des RPTTs, où toutes les actions sont de durées nulles. Donc on peut déduire un résultat important, qui est la capacité des RPTTs de modéliser les chiens de gardes. En notant que les RPTTs, par définition, ne prennent pas en considération les âges des jetons .La remarque a tiré est que les RPTTs se présentent comme une généralisation de plusieurs modèles : t-temporisé, p-temporisé et T-TPNs. Une généralisation qui ne stipule pas des modifications au niveau de la structure d'un réseau, le nombre de places reste le même (respectivement le nombre de transitions) contrairement au T-TPN qui sont une généralisation des t-temporisés et p-temporisés, mais cette dernière nécessite une modification de la structure générale [97].

III.12Conclusion :

Dans ce chapitre, nous avons défini les concepts de bases , méthodes et langages formels et leurs techniques d'utilisation, notamment dans le contexte IDM. Nous avons également présenté le formalisme du réseau de Petri. Ils sont des outils graphiques et mathématiques puissants pour la modélisation, l'analyse et la vérification des systèmes.

De façon générale en mettant l'accent sur leur aptitude à décrire les aspects dynamiques d'un système. Des concepts tels que les concurrences ou la synchronisation entre interactions s'expriment aisément dans ce cadre.

Les RPTT qui sont une manière d'introduire la notion du temps dans les réseaux de pétri, sont introduits dans notre travail du fait qu'ils constituent la base de notre recherche.

Chapitre IV :

Contribution

IV. Contribution

IV.1. Introduction :

Dans ce chapitre, nous présentons notre contribution qui consiste à transformer les diagrammes d'activités vers les RPTT en se basant sur l'approche de transformation de graphes.

Nous commencerons, par établir un Meta- modèles associés au diagramme d'activités et un deuxième méta modèle associé au RPTT. Ensuite, nous proposons notre grammaire de graphe qui permet la transformation de modèles, et nous terminerons par deux cas d'études pour illustrer notre approche de transformation.

En effet, Cette transformation de graphes est réalisée à l'aide de l'outil de modélisation ATOM^3 et les contraintes sont exprimées en python.

IV.2 ATOM^3

*AToM*3 *(A Tool for Multi-formalism and Meta-Modelling*) est un outil pour la modélisation multi-paradigme développé dans le laboratoire MSDL *(Modelling, Simulation and Design Lab)* de l'institut d'informatique à l'université de McGill Montréal, Canada.

Il est développé avec le langage *Python* 5 en collaboration avec le professeur Juan de Lara de l'université Autónoma de Madrid *(UAM)*, Espagne [16].

Les deux tâches principales d'AToM^3 sont :

1. La méta-modélisation.

2. La transformation des modèles.

Les formalismes et les modèles dans *AToM*3 sont décrits graphiquement. À partir d'une méta-spécification (exemple : dans le formalisme Entité-relation) d'un formalisme, *AToM*3 génère un outil pour manipuler visuellement (créer et modifier) les modèles décrits dans le formalisme spécifié. Les transformations des modèles sont réalisées par la réécriture des graphes, qui peuvent être exprimées d'une manière déclarative comme un modèle de grammaire de graphes [99].

Les méta-modèles permettent de construire des modèles valides dans un certain formalisme et les méta-méta-modèles sont utilisés pour décrire les formalismes eux-mêmes.

*AToM*3 traite les modèles de la même manière dans n'importe quel méta-niveau. L'idée principale d'*AToM*3 est : *"tout est un modèle"* [100].

Certains méta-modèles sont disponibles avec *AToM*3 :

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

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

La figure IV.1 illustre l'interface d'*AToM*3 avec le formalisme de diagramme de classe chargé.

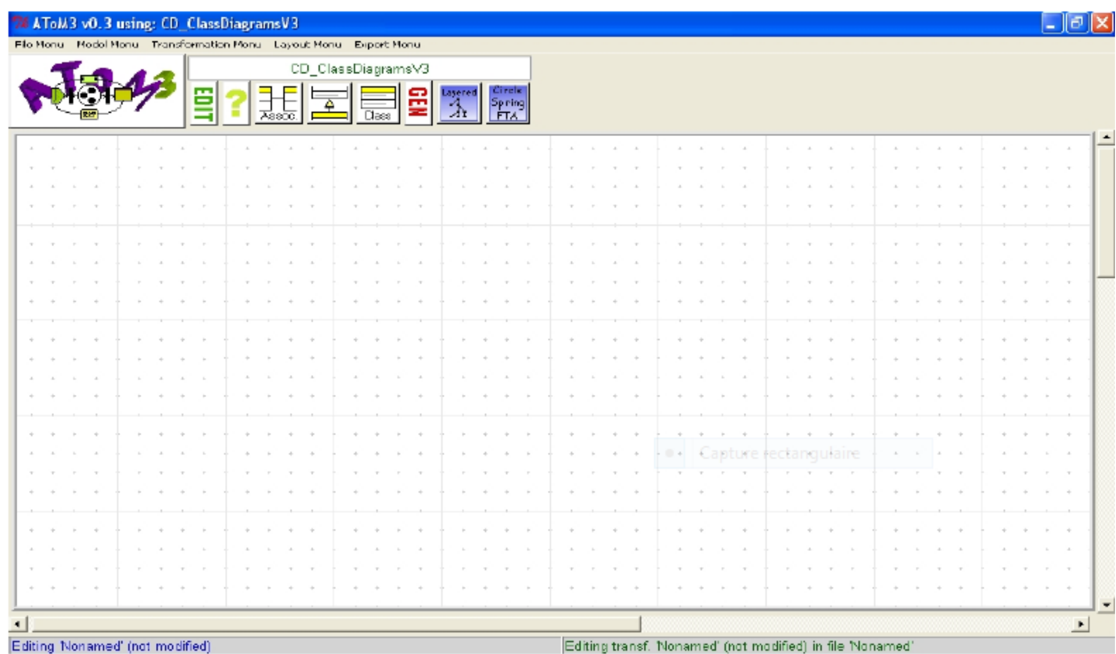


Figure IV. 1 : interface d'ATOM3

IV.2.1 Formalisme Diagrammes de Classes dans *AToM*3

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

- Nom
- Attributs
- Contraintes

- ➢ Action
- ➢ Cardinalités
- ➢ Apparence.

L'éditeur illustré par la figure IV.2 permet de manipuler ces propriétés.

IV.2.1.1 Contraintes

Les contraintes peuvent être spécifiées comme des expressions OCL (*Object Constraint Language*) ou Python [16]. Elles peuvent êtres locales associées à une entité, ou globales.

Elles ont les propriétés suivantes :

- Un nom de contrainte

- Un événement déclencheur : il peut être soit

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

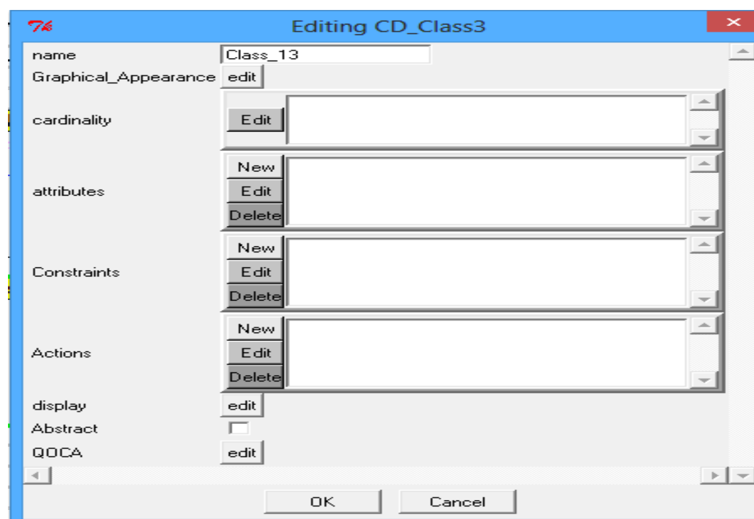


Figure IV. 2: Éditeur des propriétés

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

La figure IV.3 illustre l'éditeur des contraintes dans *AToM3*.

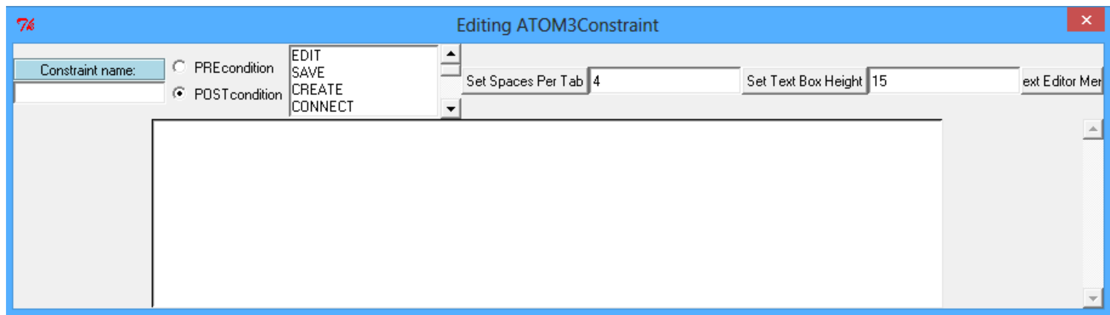


Figure IV. 3 : Éditeur de contraintes

IV.2.1.2 Action

Une action est similaire à une contrainte sauf qu'elle a d'autres effets et elle a un code en Python

Elles ont les propriétés suivantes :

-Un nom d'action.

- Un événement déclencheur : Il peut être soit

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

-L'exécution soit :

- Avant l'événement (pré-condition) ou
- Après (post-condition)

- Le code (soit en OCL ou Python)[99].

L'éditeur des actions est similaire à l'éditeur des contraintes illustré par la figure IV.3.

IV.2.1.3 Attributs

Les entités (classes et relation d'association) qui doivent apparaître sur les modèles sont spécifiées ensemble avec leurs attributs et leurs apparences graphiques. Par exemple, pour définir le formalisme de réseau de Pétri, il est nécessaire de définir à la fois les places et les transitions. En outre, pour les places nous avons besoin d'ajouter l'attribut nom et le nombre de jetons. Pour les transitions, nous avons besoin de spécifier le nom [99].

Deux types d'attributs existent dans *AToM3* :

1. ***Attributs réguliers* :** Ils sont utilisés pour identifier les caractéristiques d'une entité.

2. ***Attributs générateurs* :** Ils permettent de générer d'autres propriétés.

Il y a deux types de base dans *AToM3* :

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

La figure IV.4 illustre l'éditeur des attributs.

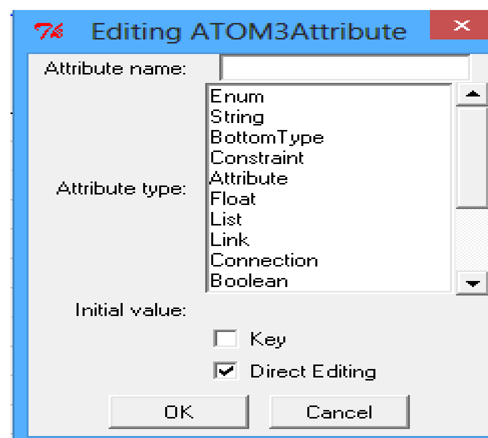


Figure IV. 4: Éditeur des attributs.

IV.2.2 Transformation de Graphes

Dans *AToM3* la grammaire est un modèle caractérisé par

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

L'éditeur de grammaire (figure IV.5) permet l'édition, la génération et l'exécution de la grammaire.

Chaque règle est constituée de :

- Un nom spécifique pour la règle.

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

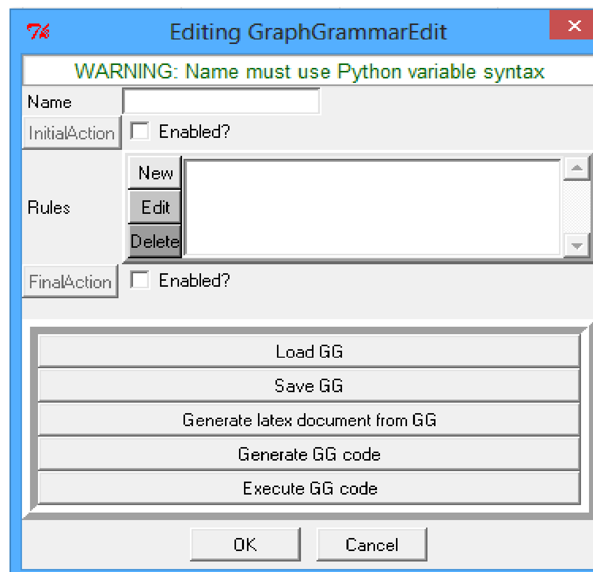


Figure IV. 5 : Éditeur de grammaire

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

L'éditeur de règle (figure IV.6) permet l'édition des différentes parties de la règle ainsi que l'action initiale et finale de chaque règle.

L'éditeur de condition et l'éditeur d'action d'une règle sont similaires à l'éditeur des contraintes présenté par la figure IV.3.

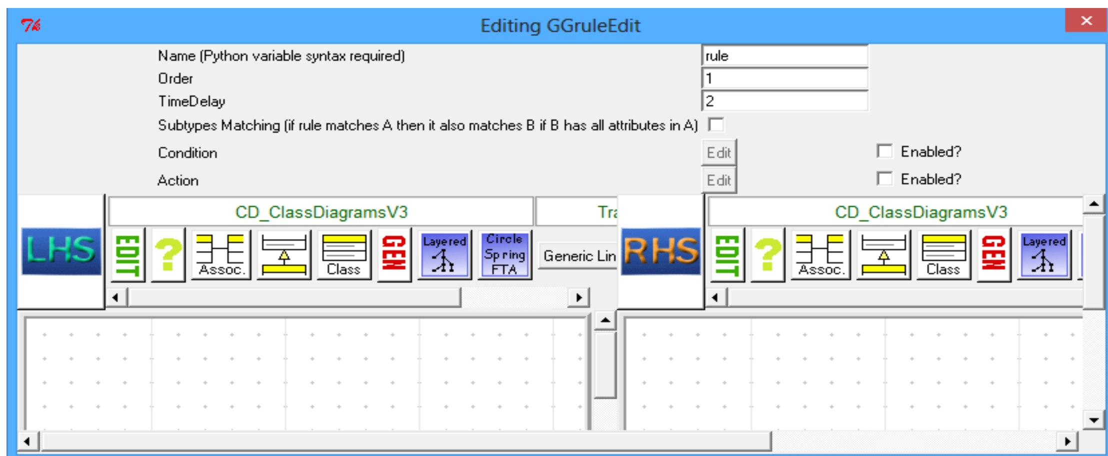


Figure IV. 6 : Éditeur de règle

IV.3 Présentation de l'approche de transformation:

Pour transformer les diagrammes d'activités vers les réseaux de RPTT, on propose les trois étapes suivantes :

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

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

3. Définition de la grammaire proposée (règles de transformation).

IV.3.1. Méta-modèle des diagrammes d'activité :

Le méta-modèle permet de spécifier les classes, attributs, contraintes, et relations, ainsi que l'apparence graphique des nœuds et des arcs. Le méta-formalisme utilisé dans notre travail est le diagramme de classe UML, et les contraintes sont exprimés en codes python.

Notre méta-modèle est composé de 12 classes et 06 associations comme illustré la figure IV.8.

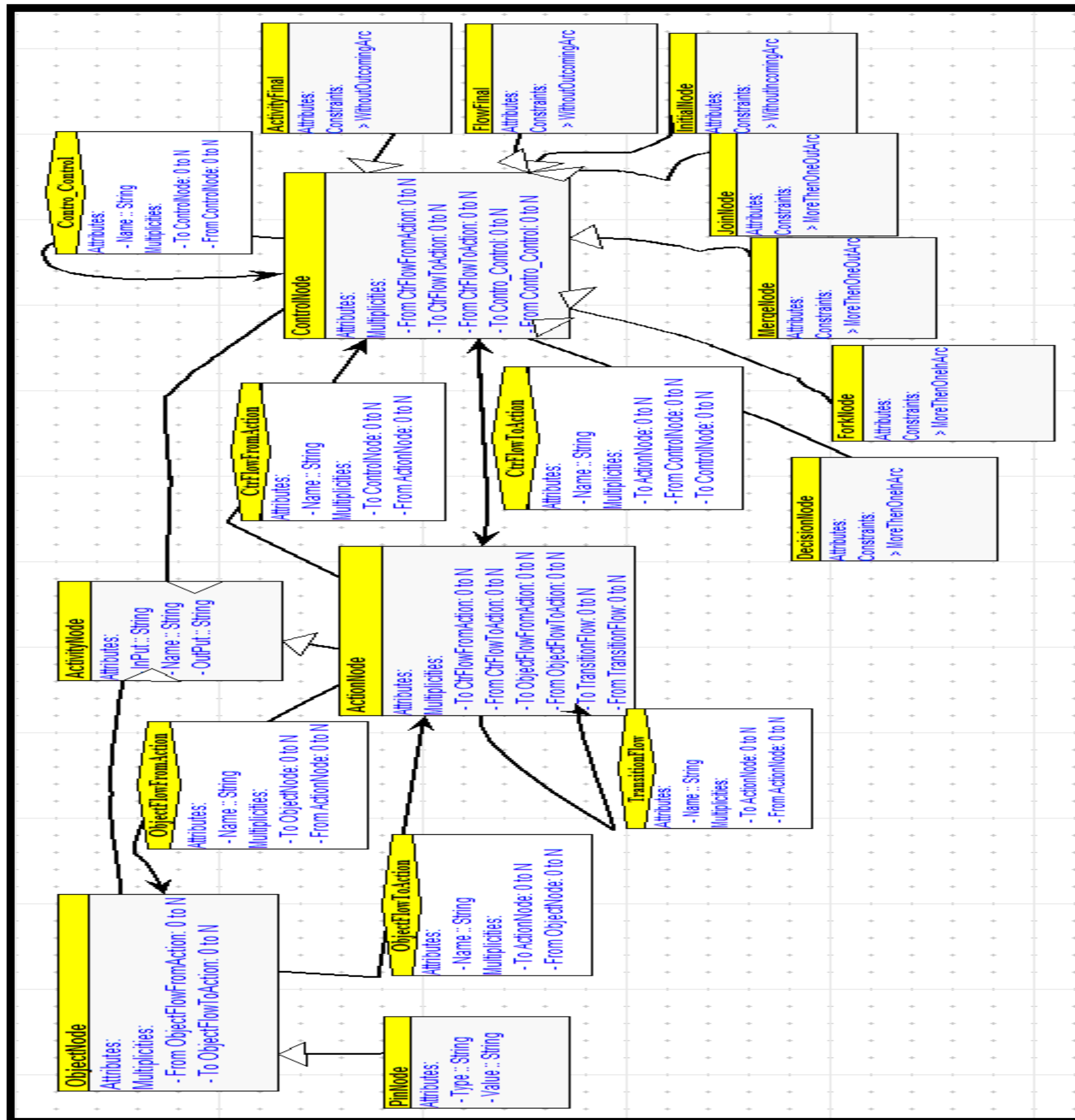


Figure IV. 7 : le diagramme d'activité

IV.3.1.1. Les classe :

Les classes sont décrites comme suit :

- **Classe Action Node :** cette classe représente une opération atomique non décomposable .elle est représentée visuellement par un ovale qui contient sa description textuelle. elle possède un attribut Name de type String.
  La classe action Node peut etre reliée par l'association :
  -CtrFlowFromAct :avec la classe Object node
  -ctrFlowToAct :avec la classe Object node.
  -ctrFlowFromAct : avec la classe conrol node .
  - ctrFlowtoAct : avec la classe conrol node.
  -transitionFlow :avec elle-même.
  -contraintPin :avec la classe pin node.

- **Classe control Node :** elle représente un nœud d'activité abstrait utilisé pour coordonner les flux entre les nœuds d'une activité. Elle est la classe mère des classes des nœuds finaux (final node, flow final) , initial node, fork node et décision node.

- **Classe initial node** : représente le début d'un diagramme d'activité. graphiquement elle est représentée par un petit cercle plein. elle possède une contrainte qui interdit l'existence de plus d'un arc sortant.

- **Classe décision Node :** cette classe spécifie les différentes alternatives possibles. elle a un arc entrant et deux ou plusieurs arcs sortants, ces derniers peuvent être gardés par des conditions. Graphiquement, elle est représentée par un losange. Elle possède un attribut Name de type String.

- **Classe fork Node :** elle représente un nœud de synchronisation qui possède un seul arc entrant et plusieurs arcs sortants qui doivent être déclenchés simultanément. Elle possède un attribut de type String.

- **Classe Merge Node :** cette classe rassemble plusieurs flots entrant en un seul flot sortant. Elle possède un attribut de type String.

- **Classe Join Node :** elle représente un nœud de synchronisation qui ne peut être franchit que lorsque toute les transitions en entrée ont été déclenchés. Elle possède un seul attribut Name de type String.

- **Classe Final Node :** indique une terminaison avec succès. Elle possède un ou plusieurs arcs entrants et aucun arc sortant. Elle est représentée visuellement par un cercle vide contenant un petit cercle plein. Cette classe possède un seul attribut Name de type String.

- **Classe Object Node :** elle représente une méta-classe abstraite permettant de définir les flux d'objets dans les diagrammes d'activités. Graphiquement, elle est noté par un rectangle contenant un attribut Name de type String.

- **Classe Pin Node :** elle représente un nœud d'objet connecté en entrée ou en sortie d'une activité. Visuellement, elle est représentée par un rectangle arrondi contenant deux attributs type, Value de type Enum et String respectivement.

IV.2.1.2. Les associations

Chaque association de méta-modèle possède un attribut Name de type String. Elle relie chaque instance de classe source avec tell instance de la classe destination. Les cardinalités pour chaque association sont :

-to Target : **1** to **1**

-from source : **1** to **1**

Les associations :

1-Contro_Contro

2-CtrFlowFromAction

3-ObjectFlowFromAction

4- ObjectFlowFromAction

5-transitionFlow.

6 –CtrFlowTo Action

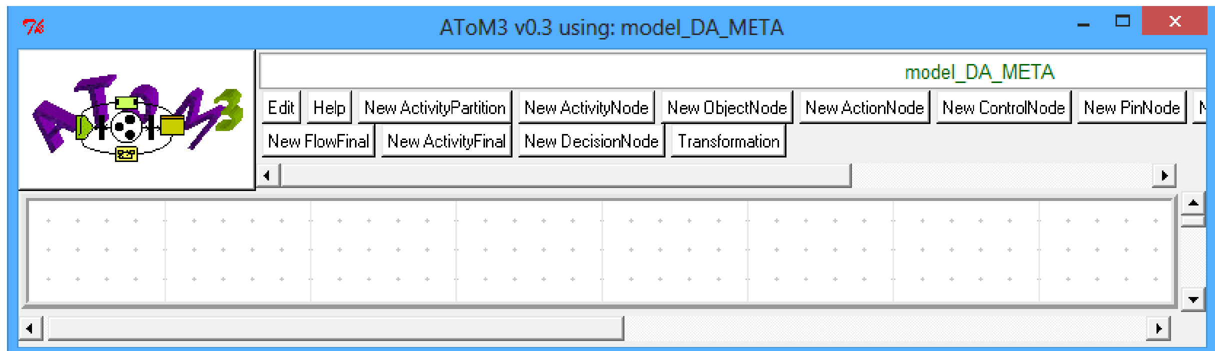


Figure IV. 8 : outil de modélisation du diagramme d'activité.

IV.3.2. Méta-modèle des RPTT :

Notre méta-modèle est composé de 02 classes et 01 seule association.

- **La classe PlaceP :** cette classe décrit les places du RPTT. Chaque place possède un nom et un nombre de jetons.
- **La classe TransitionT :** cette classe décrit les transitions du RPTT. Les transitions sont étiquetées par des noms, d'action, durée et d'un intervalle.
- **L'association ArcA :** cette association construit le modèle RPTT final en reliant les places aux transitions, et en prenant en compte les contraintes sur le modèle.

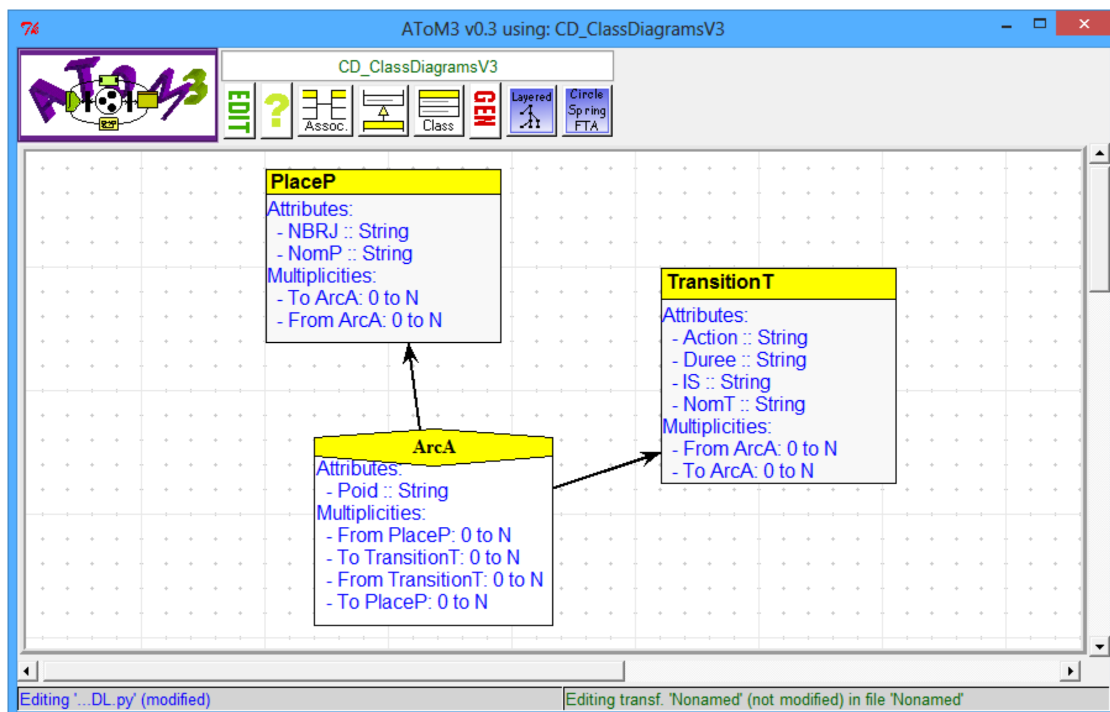


Figure IV. 9 : Méta-modèle du RPTT

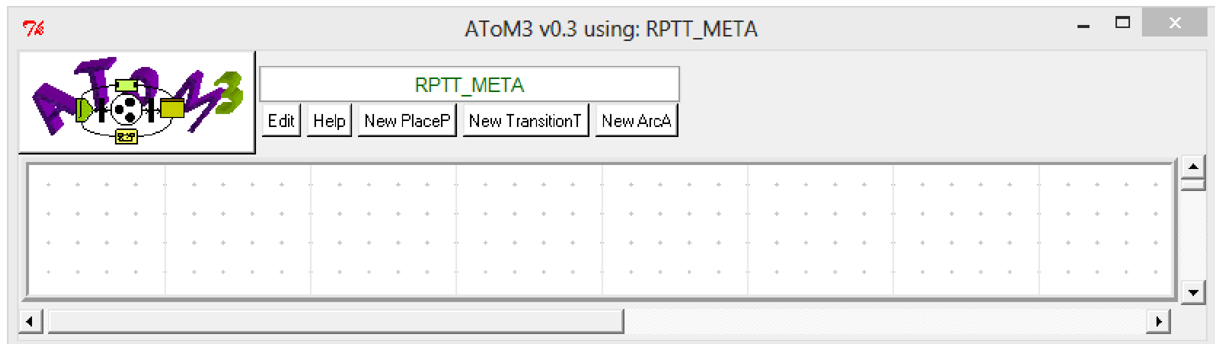


Figure IV. 10 : outil de modélisation du RPTT

IV.3.3. Définition des Règles de Transformation

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

Chaque partie peut être un sous graphe des formalismes considérés dans la transformation.

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

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

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

2. Définir les règles de la grammaire.

3. Générer le modèle cible (rptt).

IV.3.3.1. Grammaire de Graphes

Notre grammaire est composée de trois parties :

- ***Action initiale :*** Cette partie de notre grammaire sert à initialiser l'ensemble des variables globales utilisées. Ils sont utilisés pour satisfaire divers besoins. Par exemple, la variable *"visited"* est utilisé pour marquer qu'un nœud dans le graphe est déjà traité. La figure 4.10 représente le code de l'action initiale dans l'éditeur des contraintes.

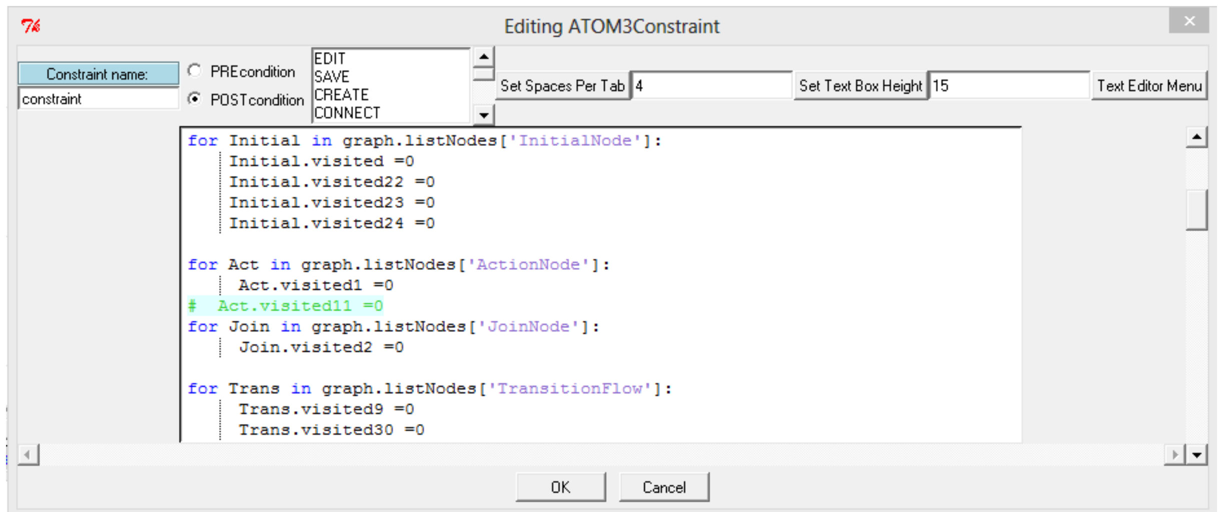


Figure IV. 11 : Action initial

➢ ***Action finale :*** À l'inverse de l'action initiale, l'action finale permet de libérer les variables globales. La figure IV.11 représente le code de l'action finale dans l'éditeur des contraintes.

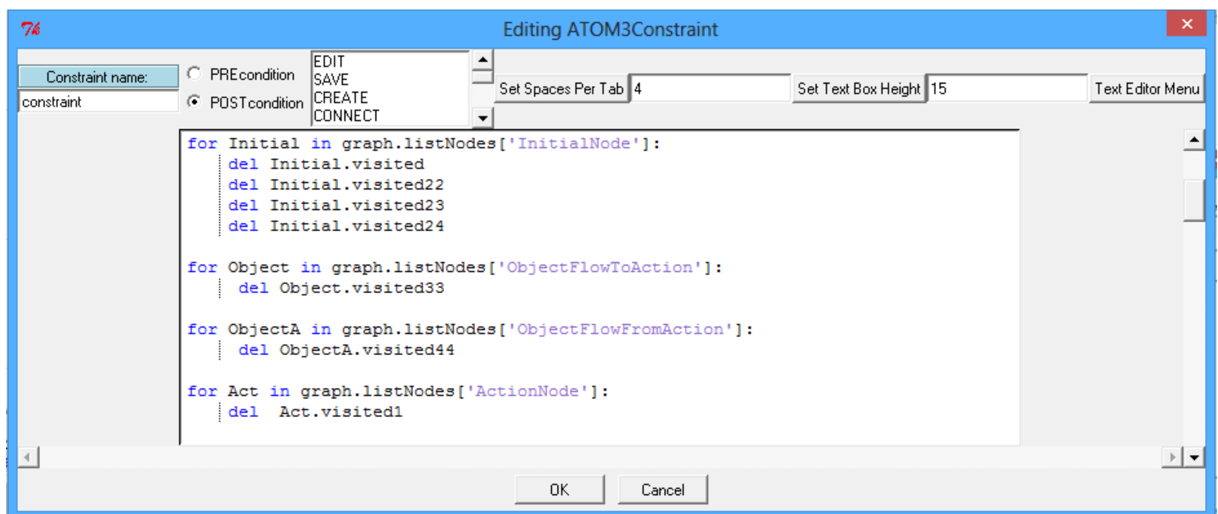


Figure IV. 12 : Action Finale

➢ ***Règles de transformation :***

Pour transformer les diagrammes d'activités vers les réseaux de Pétri Temporellement tomporisé, on propose une grammaire qui est composée de 56 règles qui sont classées en 03 catégories :

Nous citerons quelques règles:

***Catégorie 1 :* règles de transformation des nœuds du diagramme d'activités à leur correspondant dans le réseau de petri TT :** Ces règles sont appliquées pour localiser chaque nœud du diagramme d'activité et crée l'élément correspondant en réseau de pétri

Cette catégorie regroupe neuf règles :

Règle1 :

- **Nom** : Rule_Action
- **Priorité :** 1 (la première règle appliquée dans la grammaire).
- **Rôle :** cette règle est appliquée pour localiser un nœud d'action non encore traité (visited==0) en une transition dans le réseau de petri tel que les attributs (Name, Input, Output) de l'action sont affectés aux attributs (NomT, Is,Duree) respectivment.
- **Action :** le nœud d'action sera marqué comme **Visited**(visited=1)**.**

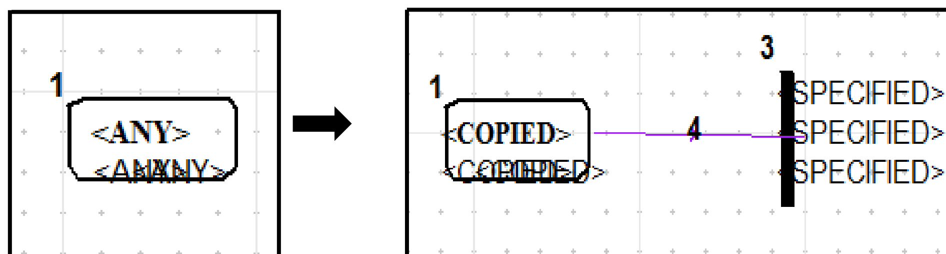


Figure IV. 13 : Rule_Action

Règle2 :

- **Nom** : Rule_joinNode
- **Priorité :** 2
- **Rôle :** cette règle permet de transformer un nœud d'union (join) non traité vers une transition dans le RPTT (de la même manière comme la règle précédente les attributs du nœud d'union serons affecté).
- **Action :** le nœud d'union sera marqué comme **Visited.**

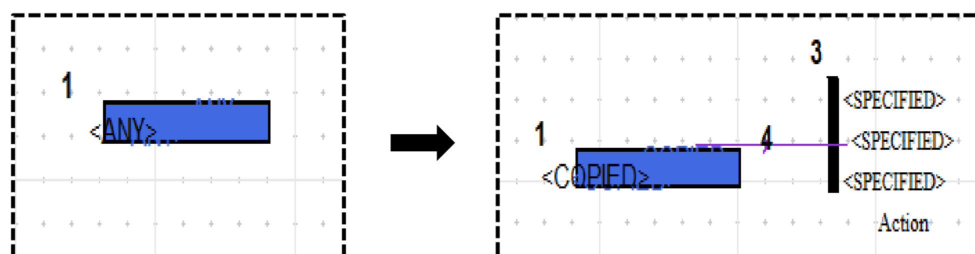


Figure IV. 14 : Rule_joinNode

Règle3 :

- **Nom** : Rule_forkNode.
- **Priorité :** 3
- **Rôle :** cette règle permet de transformer un nœud de bifurcation vers une transition en RPTT.
- **Action :** le nœud de bifurcation sera marqué comme **Visited.**

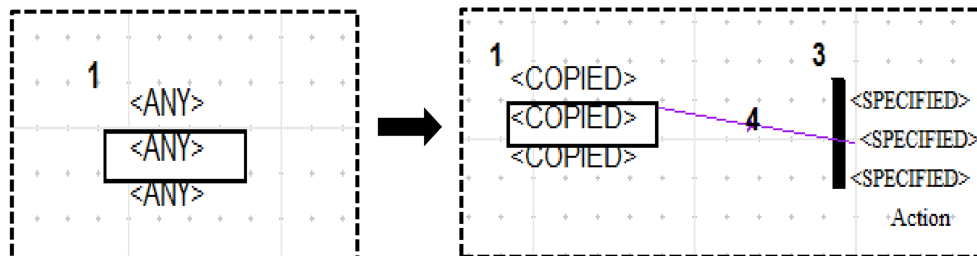


Figure IV. 15 : Rule_forkNode.

Règle4 :

- **Nom** : Rule_decesionNode.
- **Priorité :** 4
- **Rôle :** cette règle permet de localiser un nœud de décision non traité vers une place dans le RPTT, tel que l'attribut Name est affecté au NomP.
- **Action :** le nœud de décision sera marqué comme **Visited.**

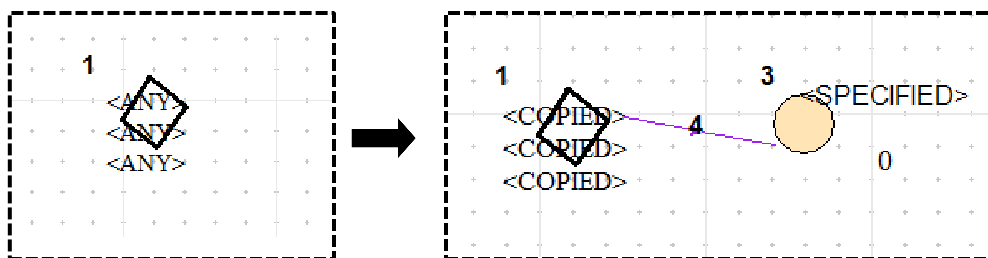


Figure IV. 16 : Rule_decesionNode.

Règle6 :

- **Nom** : Rule_PinNode.
- **Priorité :** 6
- **Rôle :** cette règle permet de transformer un nœud de pin vers une place, ainsi l'attribut Name est affecté à l' attribut NomP.
- **Action :** le nœud de pin sera marqué comme **Visited.**

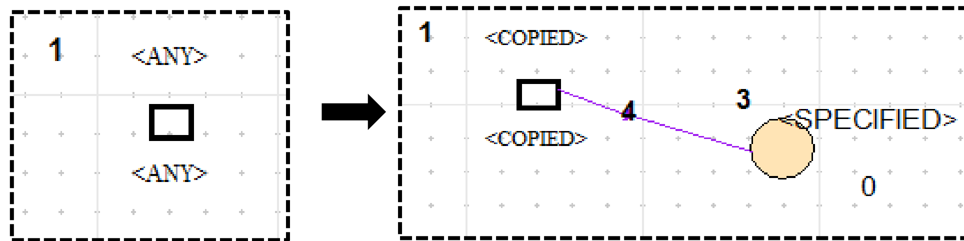


Figure IV. 17 : Rule_PinNode.

Règle8 :

- **Nom** : Rule_FlowFinal
- **Priorité :** 8
- **Rôle :** cette règle permet de transformer le flux final vers une place, ainsi l'attribut Name est affecté à l' attribut NomP.
- **Action :** le flux final sera marqué comme **Visited.**

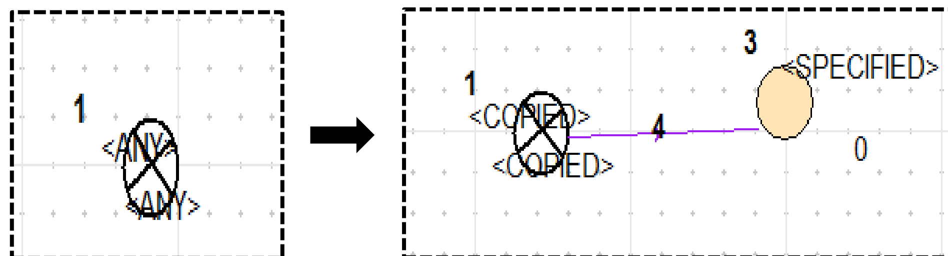


Figure IV. 18 : Rule_FlowFinal

Règle9

- **Nom** : Rule_ActiviryFinal.
- **Priorité :** 9
- **Rôle :** cette règle permet de transformer l'activité final vers une place, ainsi que l'attribut Name est affecté à l'attribut NomP.
- **Action :** l'activité finale sera marquée comme **Visited.**

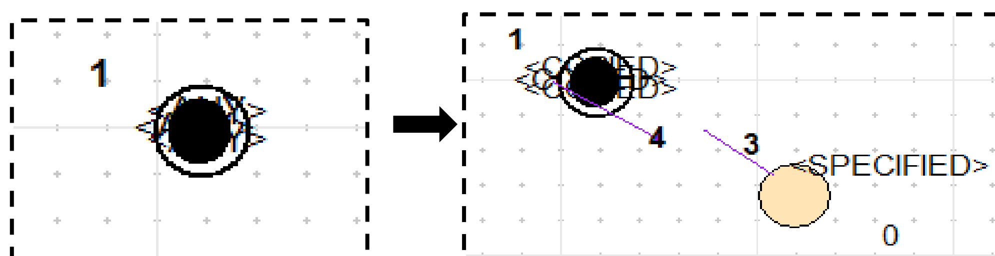


Figure IV. 19 : Rule_ActiviryFinal.

Catégorie 2 : **Règles de transformation d’une relation entre deux noeuds du diagramme d’activité vers les RPTT :** Ces règles sont appliquées pour localiser un arc reliant deux nœuds du diagramme d’activités vers un sous graphe du RPTT; Elle regroupe trente-sept règles.

<u>Règle10</u> :

- **Nom** :Rule_inittAction.
- **Priorité :** 10
- **Rôle :** cette règle est appliquée pour localiser un nœud d’action déjà traité, qui soit relié au nœud initial par un arc entrant étiqueté, ce dernier sera supprimé et transformer en une transition, son Nom est affecté à l’attribut NomP de la place crée.
- **Action :** le nœud initial sera marqué comme **Visited.**

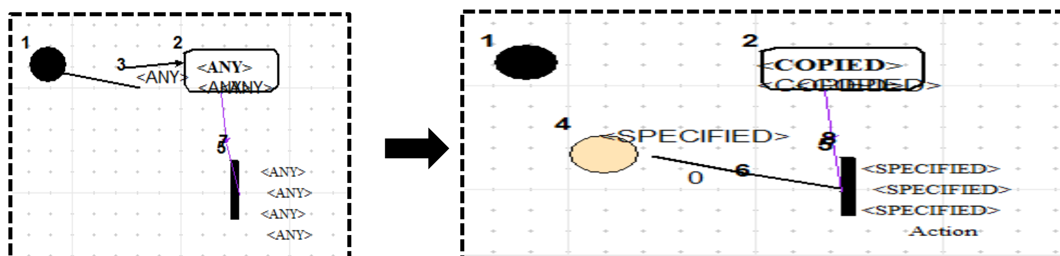


Figure IV. 20 : Rule_inittAction.

<u>Règle11</u> :

- **Nom** : Rule_inittoMerge.
- **Priorité :** 11.
- **Rôle :** cette règle permet de transformer l’arc reliant un nœud initial avec un nœud de fusion vers deux places reliées par une transition dans RPTT.
- **Action :** le nœud initial sera marqué comme **Visited.**

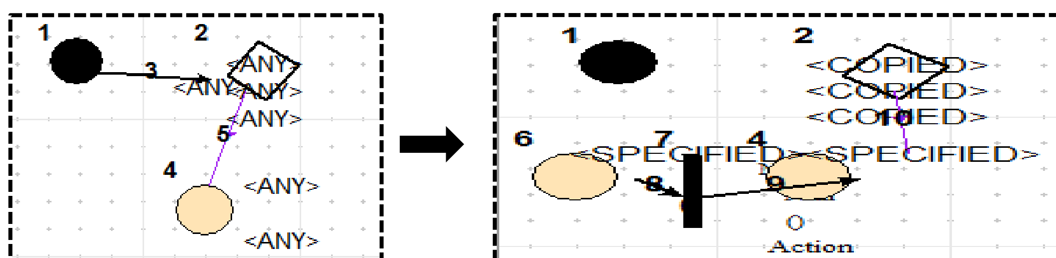


Figure IV. 21 : Rule_inittAction.

Règle12 :

- **Nom** : Rule_inittoFork
- **Priorité :** 12
- **Rôle :** cette règle permet de transformer l'arc reliant un nœud initial avec un nœud de bifurcation vers deux places reliées par une transition dans RPTT.
- **Action :** le nœud initial sera marqué comme **Visited.**

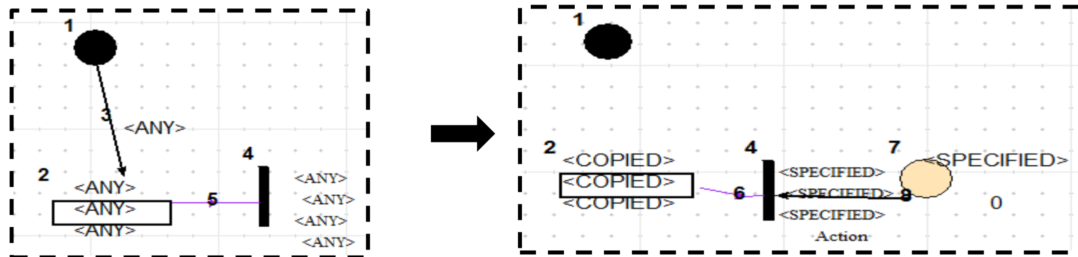


Figure IV. 22 : Rule_inittoFork

Règle14 :

- **Nom** : Rule_ActiontoAction.
- **Priorité :** 14
- **Rôle :** cette règle est appliquée pour localiser un arc non encore traité reliant les deux nœuds d'actions déjà traité, cet arc sera supprimé et son Nom sera affecté à l'attribut NomP de la place crée, qui relie les deux transitions.
- **Action :** l'arc sera marqué comme **Visited.**

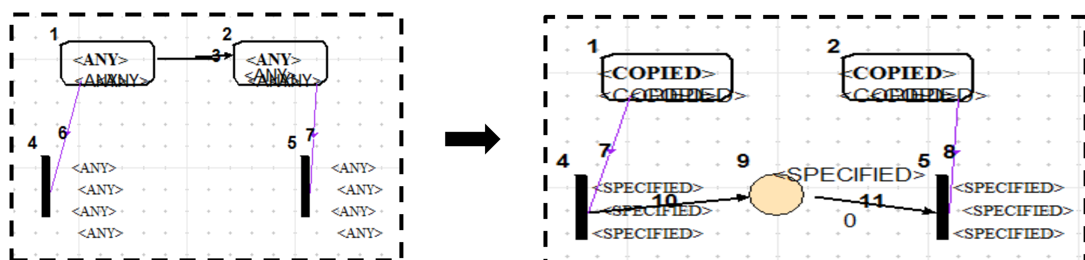


Figure IV. 23 : Rule_ActiontiAction.

Règle16 :

- **Nom** :RuleActiontoFork.
- **Priorité :** 16.
- **Rôle :** cette règle est appliquée pour localiser un arc non encore traité relié le nœud d'action avec le nœud de bifurcation qui sont déjà traités, cet arc sera supprimé et son Nom sera affecté à l'attribut NomP de la place crée, qui relie les deux transitions.

- **Action :** l'arc sera marqué comme **Visited.**

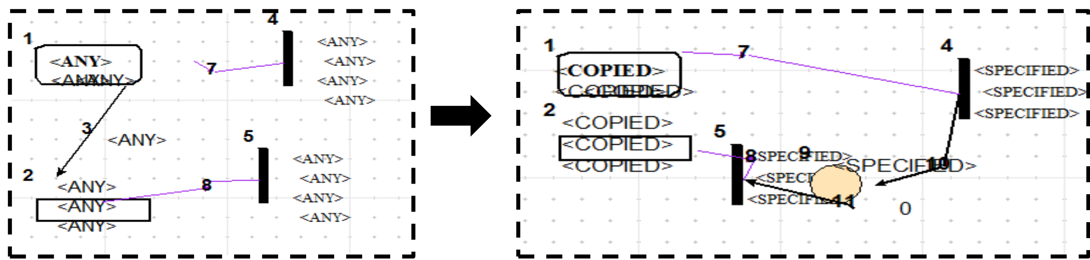


Figure IV. 24 : RuleActiontoFork.

Règle17 :

- **Nom** : Rule_ActiontoMerge.
- **Priorité :** 17
- **Rôle :** cette règle est appliquée pour localiser un arc non encore traité reliant le nœud d'action avec le nœud de fusion qui sont déjà traités, cet arc sera supprimé et remplacé par un arc qui relie la transition avec la place dans RPTT.
- **Action :** l'arc sera marqué comme **Visited.**

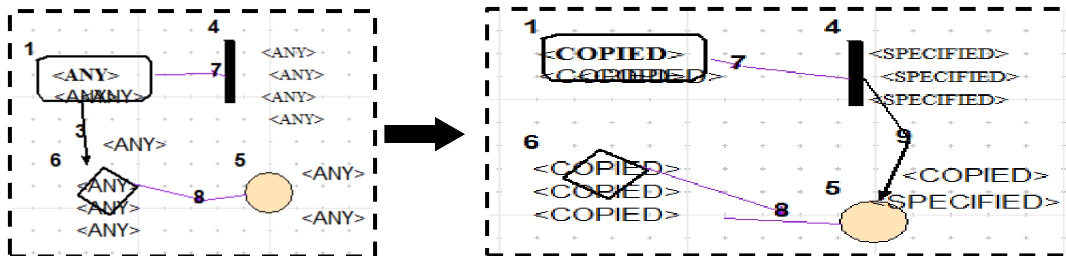


Figure IV. 25 : Rule_ActiontoMerge.

Règle19 :

- **Nom** : Rule_ActiontoJoin.
- **Priorité :** 19.
- **Rôle :** Appliquée pour localiser un arc non encore traité, reliant un nœud d'action à un nœud d'union. Le nom de cet arc est affecté au nœud destination (place).
- **Action :** l'arc sera marqué comme **Visited.**

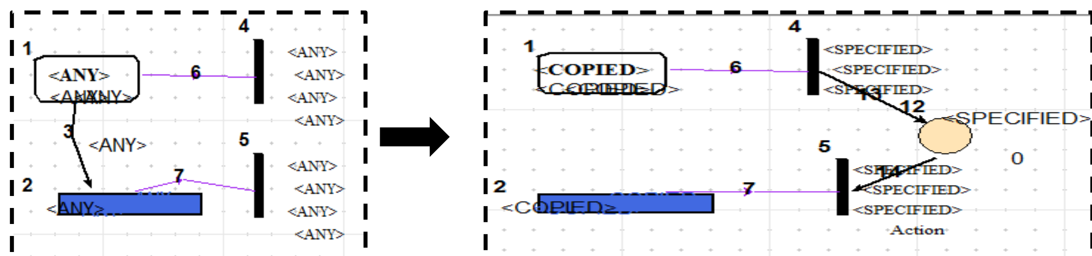


Figure IV. 26: Rule_ActiontoJoin.

Règle20:

- **Nom** : Rule_ForktoAction

- **Priorité :** 20
- **Rôle :** cette règle est appliquée pour localiser un arc non encore traité relié le nœud d'action avec le nœud de bifurcation qui sont déjà traités, cet arc est supprimé et son Nom est affecté à l'attribut NomP de la place crée, qui relié les deux transitions.
- **Action :** l'arc sera marqué comme **Visited.**

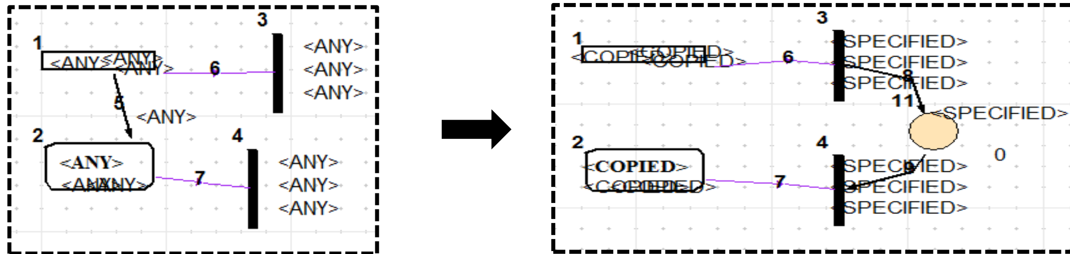


Figure IV. 27: Rule_ ForktoAction

Règle26 :

- **Nom** : Rule_DecisiontoDecision.
- **Priorité :26**
- **Rôle :** cette règle est appliquée pour localiser un arc non encore traité reliant deux nœuds de décision qui sont déjà traités, cet arc sera supprimé et remplacé par une transition qui relie les deux places déjà crées.
- **Action :** l'arc sera marqué comme **Visited.**

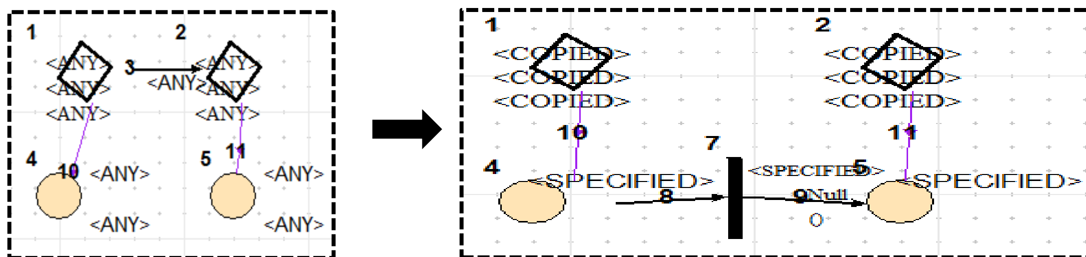


Figure IV. 28 : Rule_ DecisiontoDecision.

Règle39 :

- **Nom** : Rule_ActiontoSelf.
- **Priorité : 39.**
- **Rôle :** cette règle permet de transformer un nœud d'action vers lui-même en une place et une transition. Cette transition prend comme Nom le Nom de l'attribut du nœud d'action ainsi le Nom de l'arc reliant le nœud d'action en lui-même est affecté a l'attribut NomP de la place crée en reseau de petri.
- **Action :** l'arc (flux de control) sera marqué comme **Visited.**

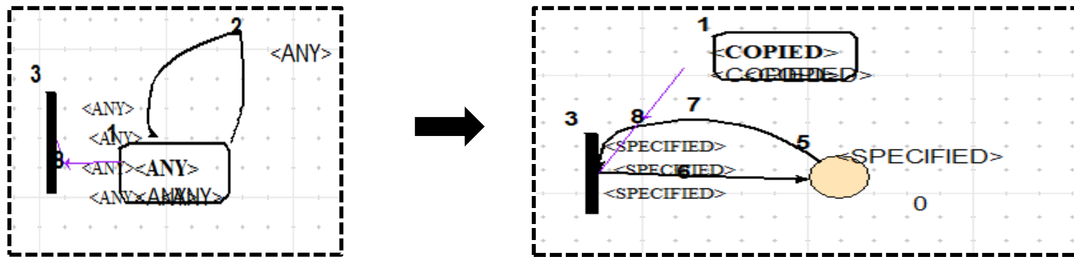


Figure IV. 29 : Rule_ ActiontoSelf.

Règle41 :

- **Nom** : Rule_ActiontoPin.
- **Priorité : 41**
- **Rôle :** cette règle est appliquée pour localiser un arc non encore traité reliant le nœud d'action avec le nœud pin qui sont déjà traités, cet arc sera supprimé et remplacé par un arc qui relie la transition avec la place dans RPTT.
- **Action :** l'arc sera marqué comme **Visited.**

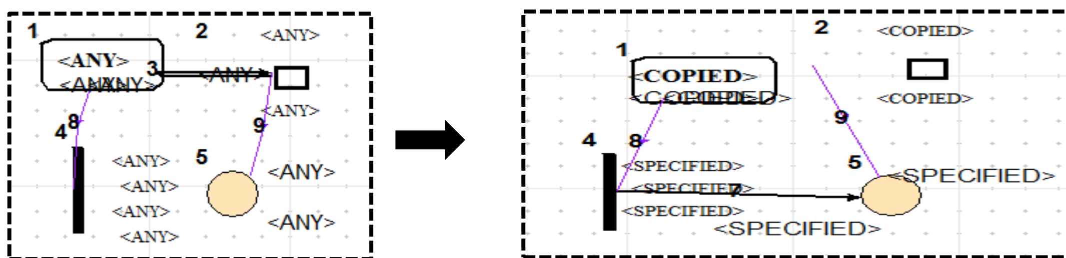


Figure IV. 30 : Rule_ ActiontoPin.

Règle42 :

- **Nom** : Rule_ActiontoActivityFinal
- **Priorité : 42**
- **Rôle :** cette règle permette de transformer le flux de contrôle entre une actions et un nœud de contrôle de type activité final à une transitions relié avec une place dans le RPTT.
- **Action :** l'arc sera marqué comme **Visited.**

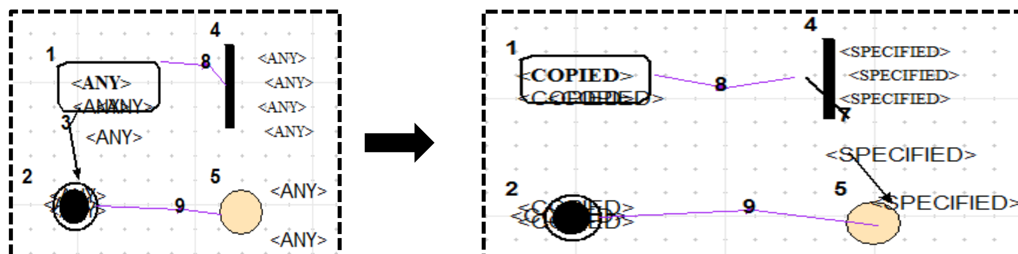


Figure IV. 31 : Rule_ ActiontoActivityFinal

Règle44 :

- **Nom** : Rule_MergetoActivityFinal.
- **Priorité : 44**
- **Rôle :** cette règle permette de transformer le flux de contrôle entre un nœud de fusion et un nœud de contrôle de type activité final à une transitions relié avec une place dans le RPTT.
- **Action :** l'arc sera marqué comme **Visited.**

Figure IV. 32 : Rule_ MergetoActivityFinal.

Catégorie 3 : **Règles d'éliminations des nœuds du diagramme d'activité :** cette catégorie regroupe huit règles qui visent à éliminer les nœuds du diagramme d'activité.

Règle47 :

- **Nom** : Rule_DelJoin.
- **Priorité :** 47.
- **Rôle :** cette règle permet d'éliminer le nœud d'union du diagramme d'activité.

Figure IV. 33 : Rule_ DelJoin.

Règle48 :

- **Nom** :Rule_DelFork.
- **Priorité :** 48.
- **Rôle :** cette règle permet de supprimer le nœud bifurcation du diagramme d'activité.

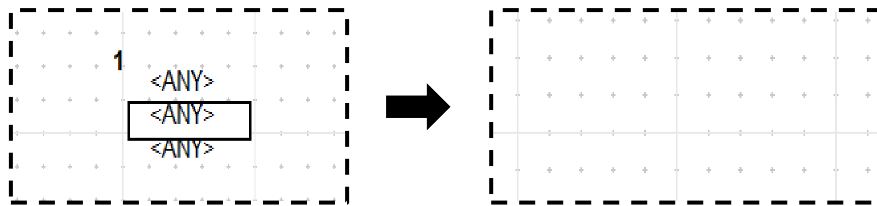


Figure IV. 34 : Rule_ DelFork.

<u>Règle49</u> :

- **Nom** : Rule_DelDecision.
- **Priorité :** 49.
- **Rôle :** cette règle permet de supprimer le nœud de décision du diagramme d'activité.

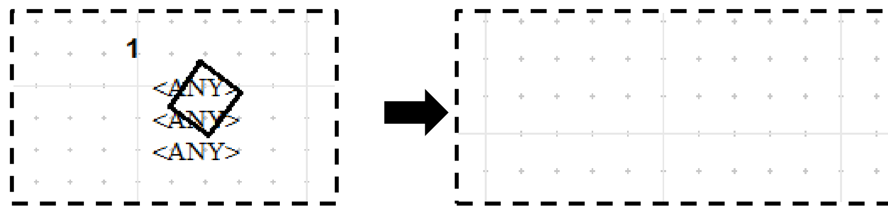


Figure IV. 35 : Rule_ DelDecision.

<u>Règle52</u> :

- **Nom** : Rule_DelAction.
- **Priorité :** 52.

Rôle : cette règle permet de supprimer le nœud d'action du diagramme d'activité.

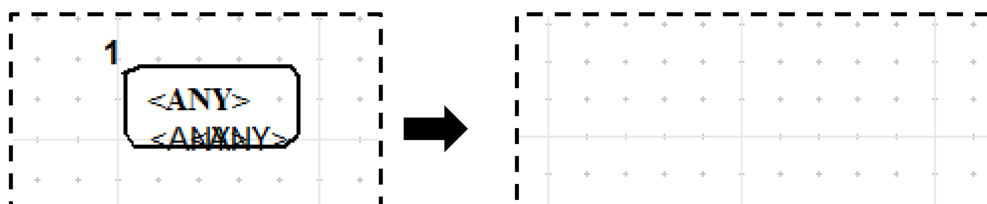


Figure IV. 36 : Rule_ DelAction.

<u>Règle53</u> :

- **Nom** : Rule_DelFlowFinal.
- **Priorité : 53.**

Rôle : cette règle permet de supprimer le flux final du diagramme d'activité.

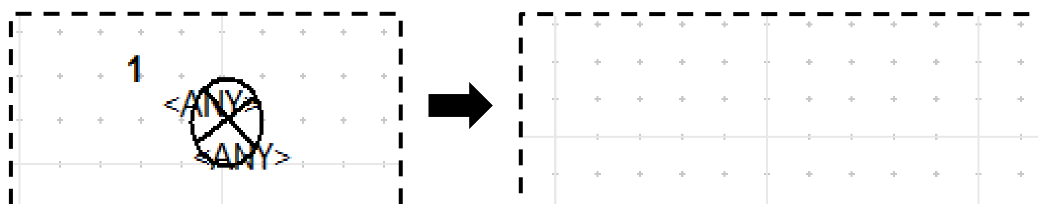


Figure IV. 37 : Rule_ DelFlowFinal.

<u>Règle55</u> :

- **Nom** : Rule_DelActivityFinal.
- **Priorité :** 55.

Rôle : cette règle permet de supprimer le nœud d'activité final du diagramme d'activité.

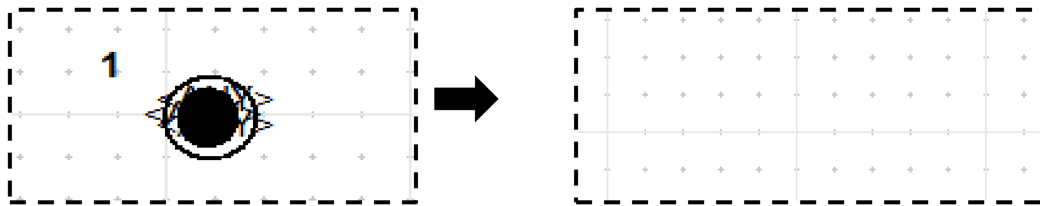


Figure IV. 38 : Rule_ DelFlowFinal

IV.4 Etude de cas :

Afin de mettre en évidence notre approche nous avons opté pour l'étude de cas deux exemples pour montrer les étapes de transformation (les règles) en utilisant l'outil AToM3. Nous avons appliqué notre approche sur « le processus de réalisation d'un mémoire de fin d'étude » Ainsi sur le comportement d'un distributeur automatique d'argent (CCP)

IV.4.1. Exemple1 : le processus de réalisation d'un mémoire de fin d'étude

Le principe de ce système: un étudiant admis en master 2 choisis un sujet de mémoire, il consulte son encadreur en lui monteront sa documentation sur le sujet, un plan de travail sera proposé par l'étudiant¸ si le plan proposé n'est pas validé par l'encadreur, l'étudiant doit présenter un autre plan de travail. Sinon l'étudiant passe à la rédaction du mémoire et la réalisation de l'application en parallèle avant le dépôt de mémoire l'étudiant doit procéder à la vérification est cela avant le délai de dépôt. Le mémoire déposé sera examiner et présenté par la suite.

Dans la Figure (IV.39), nous présentons la modélisation du comportement de ce processus avec le diagramme d'activité qui est notre modèle de base.

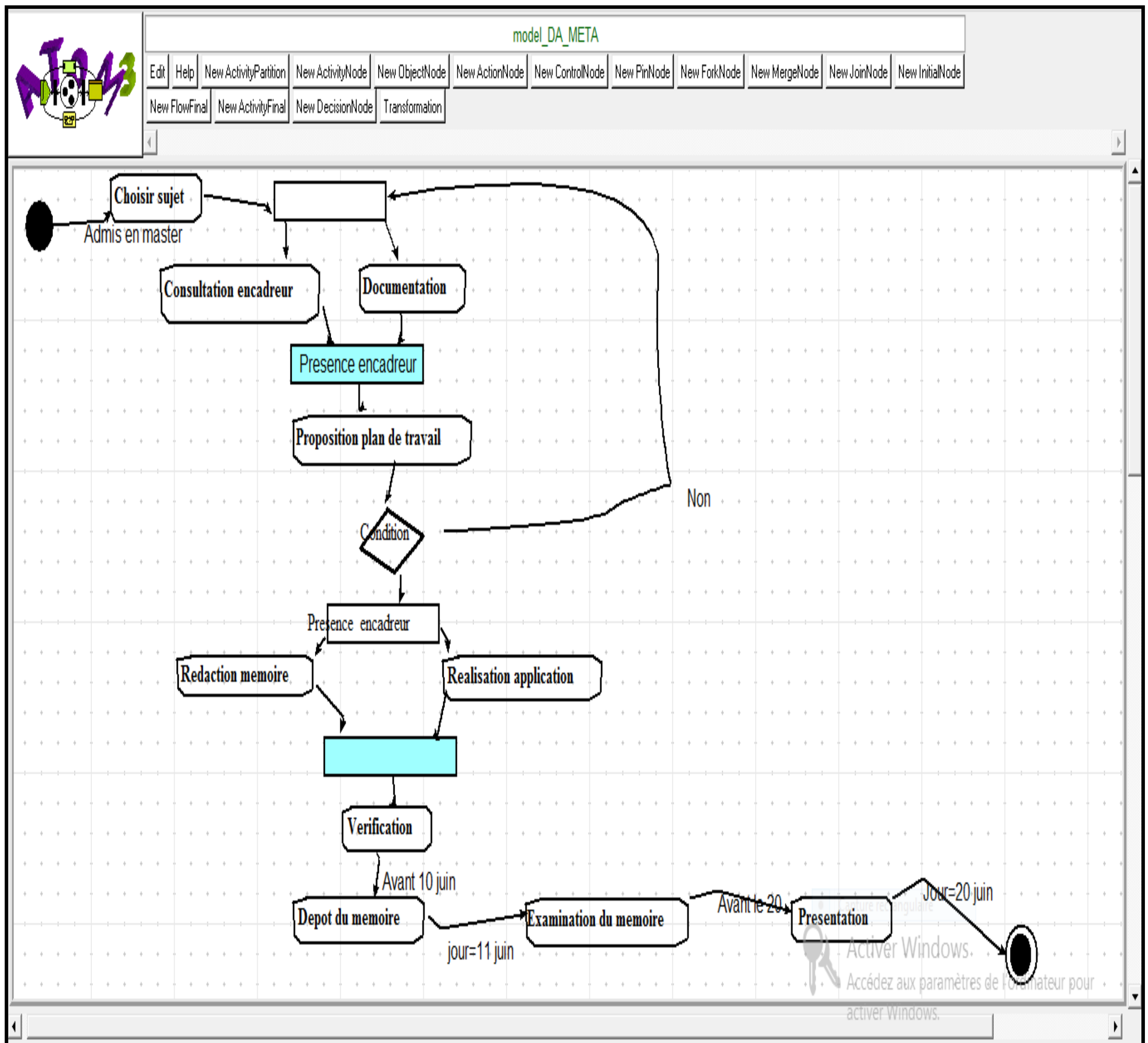


Figure IV. 39 : modèle source de processus de réalisation d'un mémoire de fin d'étude

Après l'application des règles de grammaire on obtient le modèle cible présenté dans la figure suivant.

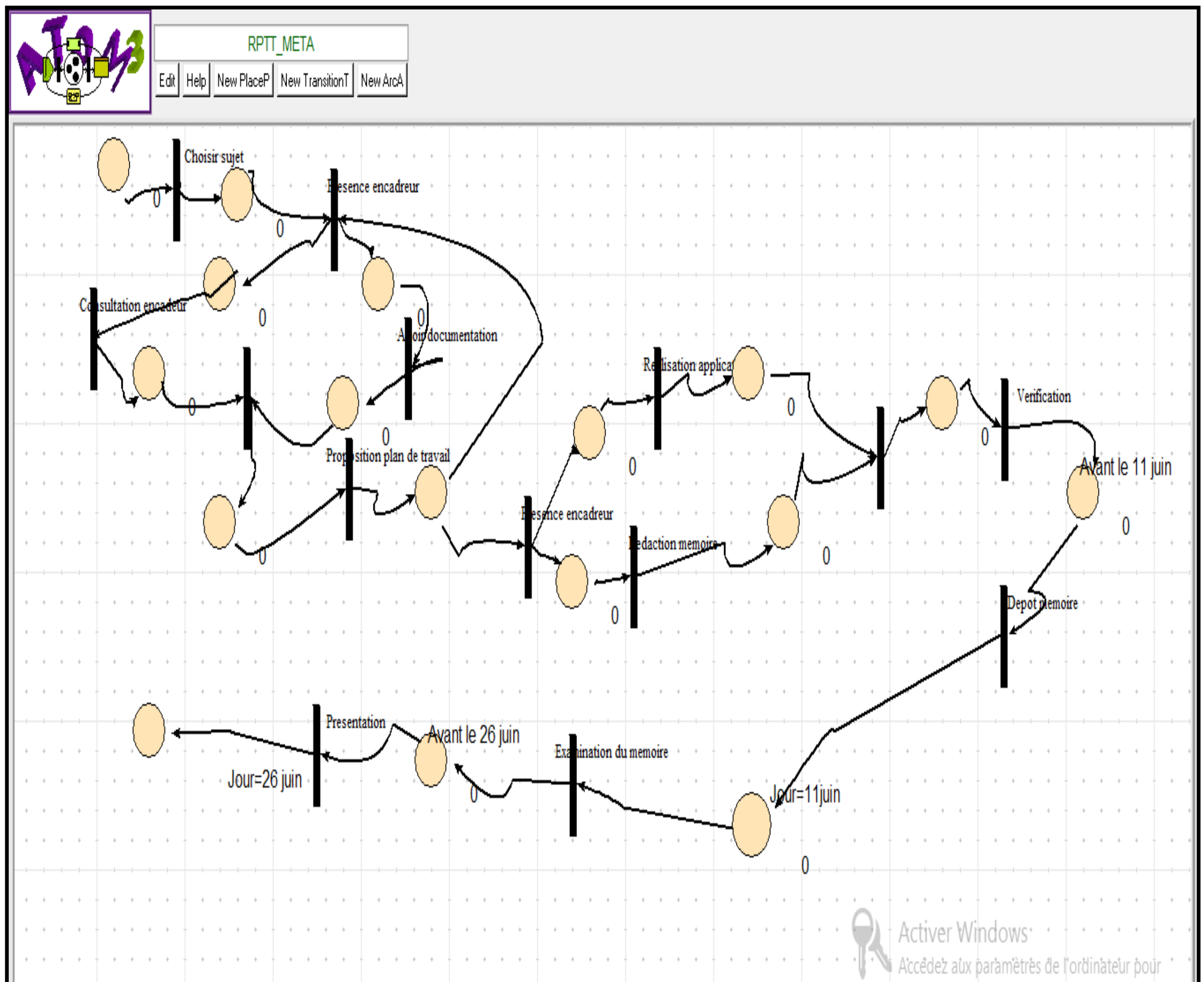


Figure IV. 40 : modèle cible de processus de réalisation d'un mémoire de fin d'étude

IV.4.2. Exemple2 : le comportement d'un distributeur automatique d'argent (CCP)

Le principe de ce système est comme suit : le client insère une carte contenant les données nécessaires à l'identification. Pour demander l'accès à son compte, le client saisit un code de chiffres. Si le code est saisi de façon incorrecte plusieurs fois de suite, le distributeur retient la carte dans le but d'éviter des fraudes. Dans le cas contraire le client sélectionne la somme correspondant au retrait souhaité puis il retire l'argent au distributeur.

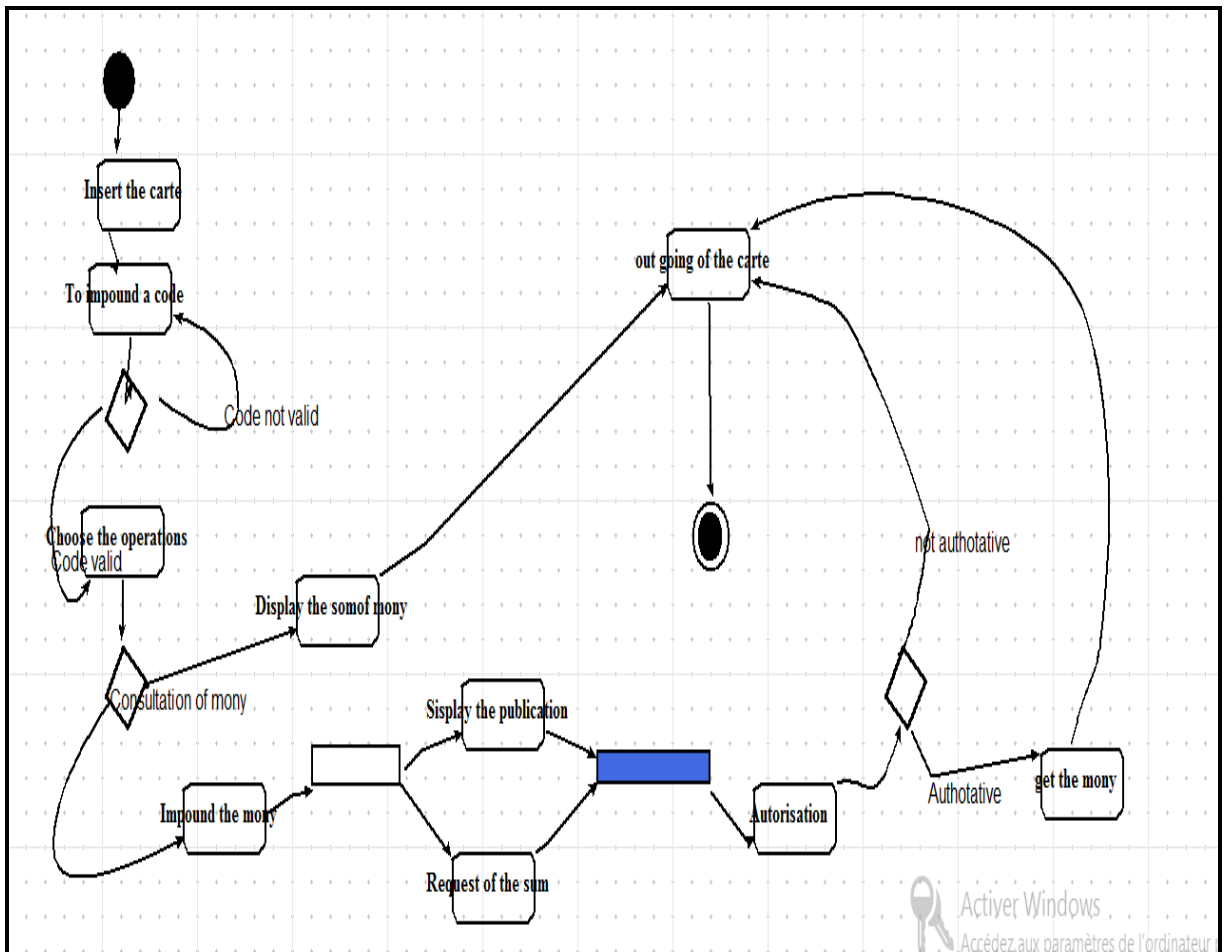


Figure IV. 41: modèle source CCP

Après l'application des règles de grammaire on obtient le modèle cible présenté dans la figure suivant.

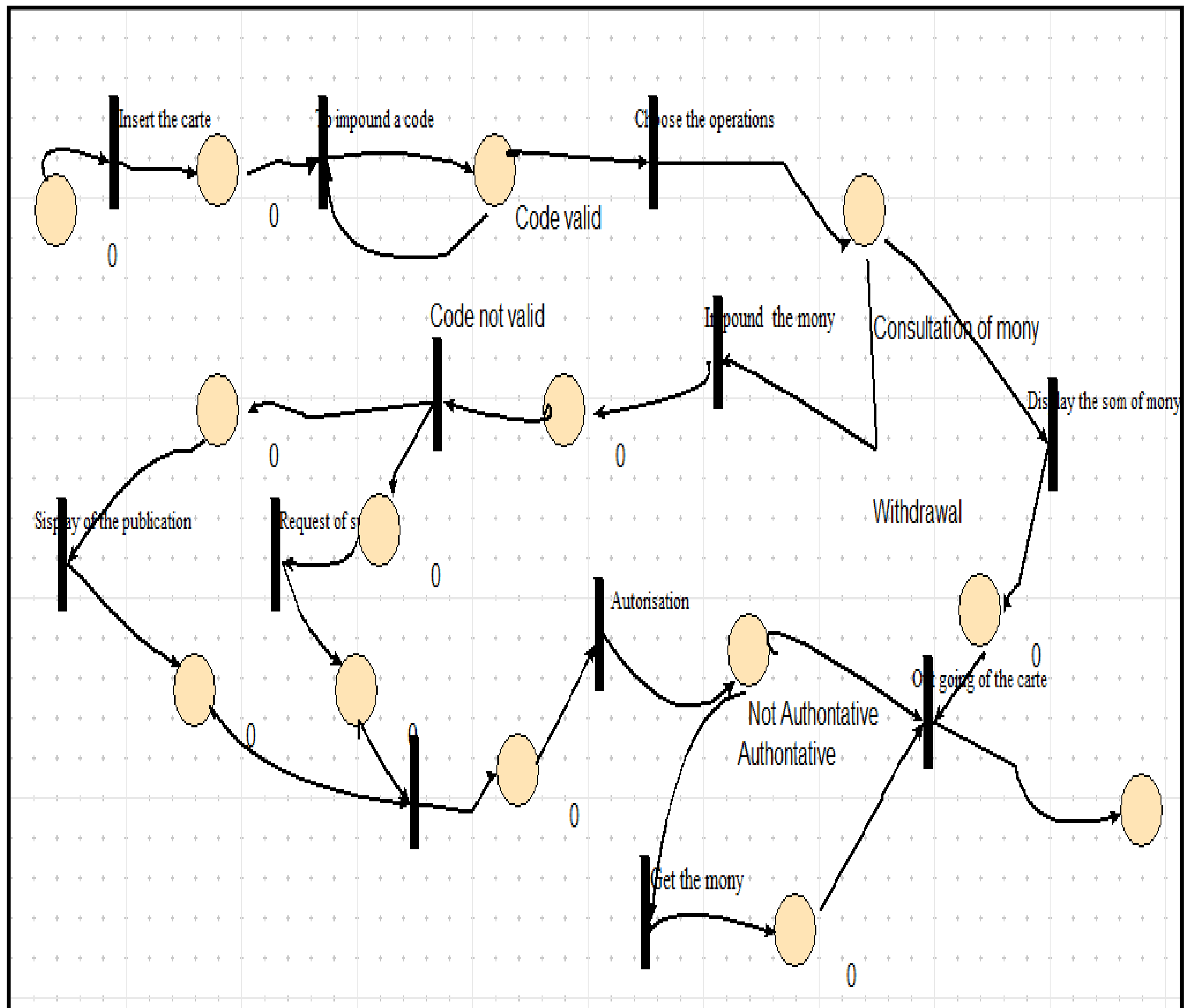


Figure IV. 42 : modèle cible CCP

IV.5. Conclusion :

Dans ce chapitre, nous avons proposé une approche de transformation de graphes pour générer les réseaux de pétri TT à partir des diagrammes d'activité. Cette approche est basée sur la méta-modélisation (les méta-modèles).

Nous avons proposé deux méta-modèles (le diagramme d'activité, réseau de pétri TT), Ensuite nous avons présenté une grammaire de graphes « ensemble de règles » permettant la réalisation de la transformation.

A la fin nous avons présenté deux cas d'études afin d'illustrer notre approche de transformation. Le premier sur le processus de réalisation d'un mémoire de fin d'étude. Le deuxième sur le comportement d'un distributeur automatique d'argent.

Finalement nous avons montré l'efficacité de notre approche à travers les résultats obtenus.

Conclusion générale

Conclusion générale

Le travail présenté dans ce mémoire s'inscrit dans le domaine de l'ingénierie dirigée par les modèles. Il se base essentiellement sur l'utilisation combinée de méta-modélisation et de transformation de modèle. Plus précisément, la transformation de graphe est utilisée comme outil de transformation de modèles.

Le résultat de notre travail est une approche automatique pour transformer les diagrammes d'activité d'UML2.0 vers RPTT « Réseaux de pétri temporellement temporisé ».

L'approche proposée est basée sur la transformation de graphes, et elle est réalisée à l'aide de l'outil ATOM³.

Le travail est réalisé dans deux étapes :

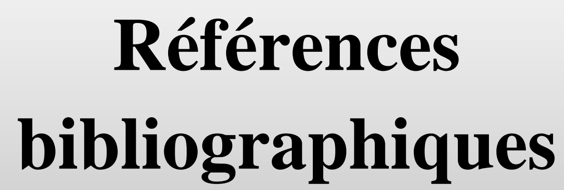
La première étape consiste à proposer deux méta-modèles des diagrammes d'activité, et RPTT afin de générer un outil visuel permettant la modélisation de ces diagrammes.

La deuxième étape propose une grammaire de graphe permettant de transformer en réseaux de pétri TT, des diagrammes d'activités graphiques.

Comme perspectives, nous comptons dans un premier lieu, de continuer la transformation des arcs et des noeuds non encore transformés dans le travail présenté (nœud d'expansion, nœud paramètre d'activité partition activités), afin d'arriver à une transformation complète des diagrammes d'activité.

Ensuite, et dans le but de valider notre approche, nous prévoyons de mettre en œuvre l'un quelconque des outils de vérification des RPTT obtenus.

Finalement, et afin d'arriver à développer une approche totalement automatique, incluant tous les diagrammes UML, nous proposerons de continuer la transformation des autres diagrammes UML (diagramme de séquence, diagramme de cas d'utilisation, diagramme d'état/transition, etc …) vers les RPTT en utilisant toujours la transformation de graphes et l'outil ATOM³.



Références bibliographiques

Références bibliographiques

[1]. L. Menet, “Formalisation d’une approche d’Ingénierie Dirigée par les Modèles appliquée au domaine de la Gestion des Données de Référence (in french)”, PhD thesis, University of PARIS VIII, Jun 2010.

[2]. Grady Booch, James Rambaugh et Ivar Jacobson : Le Guide de l’utilisateur UML, deuxième édition, Eyrolles, 2001.

[3]. J. Miller and J. Mukerji, “MDA Guide”, Object Management Group, Inc., Version1.0.1, omg/03-06-01, Jun 2003.

[4]. M. Minsky, “Matter, mind, and models”, Semantic Information Processing, pp. 425–432, 1968.

[5]. J.Bézivin and O. Gerbé, “Towards a Precise Definition of the OMG/MDA Framework”,in Proc. the 16th International Conference on Automated Software Engineering, pp.273–280, Nov 2001.

[6]. L. Lavagno, G. Martin and B. V. Selic, “UML for real: design of embedded real-time systems”, Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[7]. S. Cook, “Domain-Specific Modeling and Model Driven Architecture”, MDA Journal, pp. 1–10, Jan 2004.

[8]. J. M. Favre,”Towards a Basic Theory to Model Driven Engineering”, UML 2004 – Workshop in Software Model Engineering (WISME 2004), 2004.

[9]. OMG, “Unified Modeling Langage: Superstructure version 2.0 Final Adopted Specification”, OMG ptc/03-08-02. OMG, Aug 2003.

[10]. C. Rolland, “L’ingénierie des méthodes : une visite guide (in French)”, e-TI Journal, 2005, [Online]. Available: http://www.revue-eti.net/document.php?id=726.

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

[12]. K. Czarnecki and S.Simon Helsen, “Classification of Model Transformation Approaches”, OOPSLA’03 Workshop on Generative Techniques in the Context of Model- Driven Architecture, 2003.

[13]. http://java.sun.com/products/jmi/index.jsp {12/02/2018}

[14]. http://projects.ikv.de/qvt {08/02/2018}
[15] . http://www.eclipse.org/viatra2/ {08/02/2018}
[16]. http ://atom3.cs.mcgill.ca/ {15/12/2017}
[17]. http://www.eclipse.org/gmt/umlx/ {01/03/2018}
[18]. http://www.compuware.com/products/optimalj{03/03/2018}
[19]. http://www.sciences.univ-nantes.fr/lina/atl{15/03/2018}
[20]. http://jamda.sourceforge.net/{15/03/2018}
[21]. http://www.eclipse.org/modeling/m2t/?project=jet{15/12/2017}
[22]. G. Bernot, M. C. Gaudel and B. Marre, “Software testing based on formal specifications: a theory and a tool”, Software Engineering Journal, vol. 6, number. 6, pp.387-405, Nov 1991.
[23]. http://www.omg.org/ {20/03/2018}
[24]. S. Izza, “Intégration des systèmes d'information industriels: Une approche flexible basée sur les services sémantiques (in French)”, PhD Thesis, University Jean Monnet, Saint-Etienne, France, 2006.
[25]. Object Management Group OMG. “Unified modeling language 2.2 specification”, [Online]. Available: http://www.omg.org/spec/UML/2.2/Infrastructure/PDF/, Feb 2009.
[26]. OMG, “MetaObject Facility (MOF)”, version 2.0, [Online]. Available: http://www.omg.org/mof/.
[27]. OMG, “Object Constraint Language (OCL)”, version 2.0, [Online]. Available: http://www.omg.org/ocl/.
[28]. OMG, “XML Metadata Interchange (XMI)”, version 2.0, [Online]. Available: http://www.omg.org/xml/.
[29]. OMG, “Common Warehouse Metamodel (CWM)”, version 1.1. [Online]. Available: http://www.omg. org/technology/documents/formal/cwm.htm.
[30]. OMG, “MOFM2T Model-to-Text language”, [Online]. Available: http://www.omg.org/spec/MOFM2T/1.0/.
[31]. OMG: “Query/View/Transformation”, [Online]. Available: http://www.omg.org/spec/QVT/1.0/.
[32]. P. Roques and F. Vallée, “UML en action : De l’analyse des besoins à la conception en Java (in french)”, Eyrolles, 2000.
[33]. M. Andries, G. Engels, A. Habel, B. Hoffmann, H. J. Kreowski, S. Kuske, D. Pump, A. Schürr and G. Taentzer, “Graph transformation for specification and programming”, Science of Computer programming, vol. 34, number. 1, pp. 1-54, Apr 1999.

[34]. Object management group ,inc :Omg uml2.0, superstructure (07-2005).
http://www.omg.org/spec/Matre/1.1.9.10.

[35]. Sofiane Chemaa : « Une approche de composition de services Web à l'aide des Réseaux de Petri orientés objet » , PhD thesis, University of Constantine 2, Algeria.

[36]. E. Kerkouche, "Modélisation Multi-Paradigme: Une Approche Basée sur la Transformation de Graphes", PhD thesis, University of Constantine 2, Algeria.

[37]. G. Rozenberg, "Handbook of Graph Grammars and Computing by Graph Transformation", vol.1, World Scientific, 1999.

[38].H.Storrle and j.Haussmann.Towards a formal semantics of uml2 activities.proceedings Software Engineering, pages 117_128, 2005.5

[39]. Maria, A. (1997). Introduction to modelling and simulation,pages 7_13.

[40]. Kadima, H. (2005). Mda, une conception orientée objet guide par les modèles. DUNOD.

[41]. Jos´e-Celso Freire Junior_ - Jean-Pierre Giraudin – Agnès Front Atelier MODSI: Un Outil de M´eta-Mod´elisation et de Multi-Modélisation.§

[42]. Bahri, M. R. (2011). Une approche intégrée Mobile-UML/Réseaux de Petri pour l'Analyse des systèmes distribués à base d'agents mobiles. Thèse de doctorat, Université de Mentouri, Constantine

[43]. Laurent Audibert : UML 2.0, Institut Universitaire de Technologie de Villetaneuse, Département Informatique, Adresse du document :http://www-lipn.univ-paris13.fr/audibert/pages/enseignement/cours.htm, novembre 2007.

[44]. K. Hamilton and R. Miles, *Learning UML 2.0*. O'Reilly, April 2006. l'utilisateur UML, deuxième édition, Eyrolles, 2001.

[45]. Pierre-Alain Muller, Nathalie Gaertner : Modélisation objet avec UML, Deuxième édition, Eyrolles, 2000.

[46]. De M-UML vers les réseaux de Petri « Nested Nets » : Uneapproche basée transformation de graphes"

[47]. G. Booch, J. Rumbaugh and I. Jacobson, *"The Unified Modeling Language User Guide*", Publisher: Addison Wesley, First Edition October 20, 1998.

[48]. L. Audibert, *"UML 2", edition 2007-2008,* adresse du document : *http://www-lipn.univ-paris13.fr/audibert/pages/enseignement/cours.htm*

[49]. *"Diagramme d'activité"*, adresse du document
http://saoudyihab.voila.net/cours_uml/Diagramme_d_activite.pdf

[50]. *"OMG Unified Modeling Language TM (OMG UML), Superstructure", Février 2009,* URL Standard du document:http://www.omg.org/spec/UML/2.2/Superstructure

[51]. H.Malgouyres, J-P.Seuma-Vidal, G.Motet, *"REGLES DE COHERENCE UML2.0",* Version 1.1, 2005.

[52]. Jeannette M. Wing, "A Specifier's Introduction to Formal Methods", Computer, vol. 23(9):8-23, 1990.

[53]. J. Christian Attiogbé, "Contributions aux approches formelles de développement de logiciels : Intégration de méthodes formelles et analyse multifacette", In HDR, Université de Nantes, Nantes Atlantique Université, 13 septembre 2007.

[54]. Sam Owre, John Rushby and Natarajan Shnkar, "The PVS Specification Languages", In http:// www.csl.sri.sri.com/pvs-language/ 1993.

[55]. Graeme Smith, Florian Kammuller and Thomas Santen, "Encoding Object-Z in Isabelle/HOL", In la revue Lecture Notes in Computer Science, vol.2272, pp.82-99, 2002.

[56]. Gerd Behrmann, Alexandre David and Kim G. Larsen, " A tutorial on UPPAAL", In Proceedings on the International School on Formal Methods for the Design of Computer, Communication, and Software Systems, volume 3185 of Lecture Notes in Computer Science, pages 200-236, Bertinora, Italy, Springer-Verlag.2004.

[57]. Ninh Thuan Truong, "Utilisation de B pour la vérification de spécifications UML et le développement formel orienté objets", Dans la thèse de doctorat en Informatique de l'Université Nancy2, au LORIA à Vandoeuvre, 2006.

[58]. Joseph A. Goguen and Malcolm Grant, "Algebraic Semantics of Imperative Programs", The MIT Press, ISBN 978-0262071727, May 22, 1996.

[59]. Didier Bert, Rachid Echahed, Paul Jacquet, Marie-Laure Potet and Jean-Claude Reynaud, "Spécification, généricité, prototypage : aspects du langage LPG", In Technique et Science Informatiques, 14(9): pages 1097-1129, Hermes 1995.

[60]. John V. Guttag, James J. Horning and Jeannette M. Wing, "The Larch Family of Specification Languages", In Software, IEEE , ISSN 0740-7459 vol 2 . pp. 24-36. Sept. 1985.

[61]. Jones Cliff, "Systematic Software Development Using VDM", Prentice-Hall ISBN 978-0138807252, 1986

[62]. Kenneth J. Turner, "Representing and analysing composed web services using CRESS", in Journal of network and computer applications, ISSN 1084-8045 vol. 30, n°2, pp. 541-562, 2007.

[63]. Robin Milner, "A calculus of communicating systems", in Lecture Notes in Computer Science, vol.92, Springer-Verlag, Berlin, 1980.

[64]. Nathan Charles, Howard Bowman and Simon J. Thompson, "From Act-one to Miranda, a Translation Experiment", In Computer Standards and Interfaces Journal, 19(1), May 1997.

[65]. hassan diab : « evaluation de methodes formelles de specifation » pg10,quebec, canada, Mai1999.

[66]. Claude Benzaken, "Systèmes Formels: Introduction à la logique et à la théorie des langages", Editions Masson, 1991.

[67]. Michaël Petit, "Formal requirements engineering of manufacturing systems: a multiformalism and component-based approach", Thèse de doctorat de l'Université de Namur, Belgique, Octobre 1999.

[68]. BARROCA, Leonor M. ; MCDERMID, John A. : Formal methods : Use and relevance for the development of safety-critical systems. In : The Computer Journal 35 (1992), Nr. 6, p. 579–599

[69]. FUTATSUGI, Kokichi ; GOGUEN, Joseph A. ; JOUANNAUD, Jean-Pierre ; MESEGUER, José : Principles of OBJ2. In : Proceedings of the 12th ACM SIGACTSIGPLAN symposium on Principles of programming languages ACM (event), 1985, p. 52–66

[70]. GAUDEL, Marie-Claude : Structuring and Modularizing Algebraic Specifications : the PLUSS specification language, evolutions and perspectives. In : Annual Symposium on Theoretical Aspects of Computer Science Springer (event), 1992, p. 1–18

[71]. HOARE, Charles Antony R. et al. : Communicating sequential processes. Volume 178. Prentice-hall Englewood Cliffs, 1985

[72]. MILNER, Robin : Communication and concurrency. Volume 84. Prentice hall New York etc., 1989

[73]. LAMSWEERDE, Axel v. : Formal specification : a roadmap. In : Proceedings of the Conference on the Future of Software Engineering ACM (event), 2000, p. 147–159

[74]. ASTESIANO, Egidio ;WIRSING, Martin : An introduction ASL.
In : The IFIP TC2/WG 2.1Working Conference on Program specification and transformation
North-Holland Publishing Co. (event), 1987, p. 343–365

[75]. GORDON, Michael J. ; MELHAM, Tom F. : Introduction to HOL :
a theorem proving environment for higher order logic. Cambridge New York : Cambridge
University Press, 1993. – ISBN 978-0521441896

[76]. ALMEIDA, José B. ; FRADE, Maria J. ; PINTO, Jorge S. ; SOUSA, Simao
M. de : An overview of formal methods tools and techniques. In : Rigorous Software
Development. Springer, 2011, p. 15–44

[77]. HALBWACHS, Nicholas ; CASPI, Paul ; RAYMOND, Pascal ; PILAUD,
Daniel : The synchronous data flow programming language LUSTRE. In : Proceedingsof the
IEEE 79 (1991), Nr. 9, p. 1305–1320

[78]. ALAGAR, Vangalur S. ; PERIYASAMY, Kasilingam : Specification
of software systems. Springer Science & Business Media, 2011

[79]. Soraya Kesraoui. Intégration des techniques de vérification formelle dans une approche
de conception des systèmes de contrôle-commande : application aux architectures SCADA.
Automatique / Robotique. Université de Bretagne Sud, 2017. Français. ‘‘https://tel.archives-ouvertes.fr/tel-01738049’’.

[80]. Mahdi Gueffaz. ScaleSem : Model Checking et Web Sémantique. Calcul formel [cs.SC].
Université de Bourgogne, 2012. Français. ‘‘ https://tel.archives-ouvertes.fr/tel-00801730v1’’.

[81]. Schnoebelen, P., Bérard, B., Bidoit, M., Laroussinie, F., Petit, A. (1999). *Vérificationde logiciels : Techniques et outils de model-checking*, Vuibert, Paris.

[82]. Clarke, E. M., Grumberg, O. et Peled, D. A. (1999). *Model checking*. MIT Press.

[83]. Bardin, S. (2008). *Introduction au Model Checking*. École Nationale Supérieure
deTechniques Avancées.

[84]. Manna Z. et Pnueli, A. (1992). *The Temporal Logic of Reactive and Concurrent Systems*, volume I (Specification). Springer Verlag, 1992.

[85]. Monin J. F. (2000). *Introduction aux méthodes formelles*. Hermès, préface de G.
Huet. 2-7462-0140-2.

[86]. Baier C. et Katoen, J. P. (2008). *Principles of Model Checking*. ISBN-10: 0-262-02649-X ISBN-13: 978-0-262-02649-9

[87]. Myers, G. J. (1979). *The Art of Software Testing*. John Wiley & Sons.

[88]. Ammann, P. et Offutt, J. (2008). *Introduction to Software Testing*. Cambridge
University Press.

[89]. Mathur, A. P. (2008). *Foundations of Software Testing*. April 17, 2008 | ISBN-10:8131716600 | ISBN-13: 978-8131716601 | Edition: 1.

[90]. Cook, S. (1971). *The complexity of theorem-proving procedures*. In 3rd Annual

[91]. Dingel, J. et Filkorn, T. (1995). *Model Checking for infinite state systems using dataabstraction, assumption commitment style reasoning and theorem proving.* In 7th International Conference on Computer Aided Verification (CAV), volume 939 of Lecture Notes in Computer Science, pages 54–69. Springer-Verlag.

[92]. C. A. Petri, “Kommunikation mit automaten (in german)”, PhD thesis, University of Bonn, Germany, 1962.

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

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

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

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

[97]. Asma Chachoua et Radja Boukharrou « Mise en oeuvre distribuée de la sémantique opérationnelle des réseaux de Petri temporellement temporisés : Approche Orientée Objet »

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

[99]. J. de Lara, H. Vangheluwe, and M. Alfonseca, “Meta-modelling and graph grammars for multi-paradigm modelling in AToM3,” *Software and System Modeling*, vol. 3,no. 3, pp. 194–209, 2004.