

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Sadik Benyahia de Jijel
Faculté des Sciences Exactes et Informatique
Département d'Informatique



Mémoire de fin d'études

pour l'obtention du diplôme Master de Recherche en Informatique

Option : Système d'informations et aide à la décision

Thème

Une Approche Automatique de Transformation
des Diagrammes d'activités UML
vers les Modèles YAWL

Présenté par :
DALAL OUKHAF

Encadré par :
Dr. AISSAM BELGHIAT

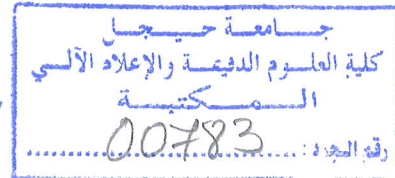
Promotion : 2019.

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Sadik Benyahia de Jijel
Faculté des Sciences Exactes et Informatique
Département d'Informatique

02
02

inf.SIAJ.09/19



Mémoire de fin d'études

pour l'obtention du diplôme Master de Recherche en Informatique

Option : Système d'informations et aide à la décision

Thème

Une Approche Automatique de Transformation
des Diagrammes d'activités UML
vers les Modèles YAWL

Présenté par :
DALAL OUKHAF



Encadré par :
Dr. AISSAM BELGHIAT

Promotion : 2019.

** Remerciements **

*Tout d'abord je remercie le **Dieu** tout puissant et miséricordieux, qui m'a donné la volonté, la force, la patience d'accomplir ce modeste travail.*

*Je tiens aussi à exprimer mon sincère reconnaissance et mon profonde gratitude à mon encadreur M^r "**BELGHIAT AISSAM**" pour son soutien, sa disponibilité, ses orientations, ses précieux conseils et ses encouragements qui m'a permis d'élaborer ce travail dans des bonnes conditions.*

*Je voudrais aussi exprimer ma reconnaissance envers **tous les membres du jury** pour leurs attentions et intérêts portés envers notre travail. Merci de m'ont honoré de votre présence.*

*Sans oublier de présenter mon sincères remerciements à **mes parents** qui ont étaient toujours de j'encourager durant mon parcours de mes études, ainsi que pour leurs aides, leurs compréhensions et leurs soutiens.*

*J'adresse mes remerciements les plus sincères à **mes soeurs et mes amis**, qui m'ont toujours encouragé au cours de la réalisation de ce mémoire.*

Je tiens également à remercier aussi au corps professoral et administratif de la Faculté des Sciences Exactes et Informatique "Département d'informatique", pour la richesse et la qualité de leur enseignement et qui ont déployé des efforts au profit de leurs étudiants.

*Je souhaite adresser mes remerciements les plus sincères et ma reconnaissance au professeur "**BOUAZIZ HAMIDA**" pour son soutien moral en tout les moments difficiles et pour ses conseils. Merci de votre aide, de votre disponibilité tout le temps et de vos encouragements.*

En fin, je tiens à remercier aux personnes qui nous ont apporté leur aide et qui ont contribué d'une façon ou d'une autre à l'élaboration de ce mémoire.

** Merci à tous et à toutes **

✳ Dédicaces ✳

*Tous les mots ne sauraient exprimer la gratitude, l'amour, le respect, la reconnaissance,
c'est tout simplement que :*

Je dédie ce mémoire de fin de cycle de Master 2 :

À mes parents : Aucun hommage ne pourrait être à la hauteur de l'amour Dont ils ne cessent de me combler. Que dieu leur procure bonne santé, longue vie et vous gardez.

*Mon très cher Père **Abderrahmane** : Aucune dédicace ne saurait exprimer l'amour, l'estime et le respect que j'ai toujours pour vous. Rien au monde ne vaut les efforts fournis jour et nuit pour mon éducation et mon bien être. Ce travail est le fruit de vos sacrifices que vous avez consentis pour mon éducation et ma formation le long de ces années.*

*Ma tendre Mère **Rokia** : Tu représentes pour moi le symbole de la bonté par excellence, la source de tendresse et l'exemple du dévouement qui n'a pas cessé de m'encourager et de Prier pour moi. Ta prière et ta bénédiction m'ont été d'un grand secours pour mener à bien mes études.*

Aucune dédicace ne saurait être assez éloquente pour exprimer ce que tu mérites pour tous les sacrifices que tu n'as cessé de me donner depuis ma naissance, durant mon enfance et même à l'âge adulte. Tu as fait plus qu'une mère puisse faire pour que ses enfants suivent le bon chemin dans leur vie et leurs études. Je te dédie ce travail en témoignage de mon profond amour. Puisse Dieu, le tout puissant, te préserver et t'accorder santé, longue vie et bonheur.

*Mes très chères sœurs **Chaima, Ahlam, Rania** pour leurs encouragements permanents, et leurs soutiens morales surtout ma belle jumeaux **Sarra**.*

Tous mes enseignants depuis ma première année d'études.

*Mes très chères amies **Wafa, Houda, Zineb, Selma, et Asma**.*

*Je le dédie aussi à mes camarades de promotion 2018 **Rahima et Sabah**.*

Tous ceux qui me sentent chers et que j'ai omis de citer.

Merci d'être toujours là pour moi

Dalal

Table des matières

Table des matières	1
Liste des tableaux	4
Table des figures	6
Liste des abréviations	7
Introduction générale	8
1 Le diagramme d'activité d'UML 2.0	11
1.1 Introduction	11
1.2 Modélisation	11
1.2.1 Définition	11
1.2.2 Les types de modélisation	11
1.3 Langage de modélisation UML	13
1.4 Historique de langage UML	13
1.5 les avantages et les inconvénients de langage UML	14
1.5.1 les avantages	14
1.5.2 les inconvénients	14
1.6 Les vues d'un système	14
1.7 Les diagrammes UML	15
1.7.1 Diagrammes structurels ou diagrammes statiques	16
1.7.2 Diagrammes comportementaux ou diagrammes dynamiques	17
1.8 Diagramme d'activité	18
1.8.1 Définition	18
1.8.2 Intérêts des diagrammes d'activité	18
1.8.3 Composants d'un diagramme d'activité	19
1.9 Exemple	24
1.10 Conclusion	25
2 Le langage YAWL	26
2.1 Introduction	26

2.2	Le processus métier	26
2.2.1	Aspect des processus métier	26
2.2.2	Gestion de processus métier (BPM)	27
2.2.2.1	Les niveaux d'abstraction de BPM	28
2.3	Langages de modélisation de processus métier	28
2.4	Langage YAWL	28
2.4.1	Définition de workflow	28
2.4.2	présentation de langage	29
2.4.3	les éléments de YAWL	29
2.4.4	les avantages et les inconvénients	33
2.4.4.1	Les avantages	33
2.4.4.2	Les inconvénients	33
2.5	Exemple	34
2.6	Outils d'analyse des modèles YAWL	34
2.6.1	L'outil YAWL 4.2	35
2.7	Conclusion	35
3	Transformation de modèle	36
3.1	Introduction	36
3.2	Ingénierie Dirigée par les Modèles	36
3.2.1	Définition de Modèle	37
3.3	L'approche MDA	37
3.3.1	Présentation de l'approche MDA	37
3.3.2	l'architecture de l'approche MDA	38
3.3.3	Standards liée à l'MDA	39
3.3.4	Typologie des modèles dans l'approche MDA	40
3.3.5	Transformation de modèle	41
3.3.5.1	Définition de transformation	41
3.3.5.2	Principe de transformation	41
3.3.5.3	les types de transformation	42
3.3.6	Transformation de modèles dans MDA	43
3.3.7	Propriété de transformation des modèles	44
3.3.8	Classification des approches de transformation de modèles	45
3.4	Transformation de diagramme d'activité vers le langage Yawl	47
3.5	Conclusion	50
4	Une Approche de transformation des diagrammes d'activités vers YAWL	51
4.1	Introduction	51
4.2	Transformation de graphe	51
4.2.1	Principe de transformation de graphe	51

4.2.2	notion de graphe	52
4.2.2.1	Graphe non orienté	52
4.2.2.2	Graphe orienté	53
4.2.3	Outils de transformation de graphes	54
4.2.4	Présentation de l'outil AToM ³	54
4.3	Transformation du diagramme d'activité UML vers le langage YAWL	55
4.3.1	Scénario de la transformation	55
4.3.2	Méta-modèle de diagramme d'activité	56
4.3.3	Méta-modèle de langage YAWL	59
4.3.4	Génération de l'environnement	62
4.3.5	La grammaire de graphe proposée	64
4.3.6	Exemple de transformation d'un diagramme d'activité vers le YAWL	79
4.4	La transformation des modèles YAWL vers une description textuelle	81
4.4.1	la grammaire de graphe proposée	81
4.4.2	Exemple	86
4.5	vérification	88
4.6	Conclusion	90
	Conclusion générale	92
	Bibliographie	93

Liste des tableaux

1.1	Les noeuds de contrôle [11] [13] [3].	21
2.1	les situations de branchement [25] [19].	30
3.1	Transformation des éléments de diagramme d'activité vers les éléments de YAWL.	49

Table des figures

1.1	Catégorisation des langages [2].	12
1.2	Historique de langage de modélisation UML [6].	13
1.3	Modélisation de l'architecture d'un système [9].	15
1.4	La hiérarchie des diagrammes d'UML 2.0 [11].	16
1.5	Notation de noeud d'objet [7].	21
1.6	Notation d'exception [7].	22
1.7	Les types d'action en UML [1].	23
1.8	Exemple d'un diagramme d'activité [13].	24
2.1	Cycle de vie d'un processus métier [20].	27
2.2	Exemple de modèles de synchronisation avancés [22].	31
2.3	Exemple de modèles impliquant plusieurs instances [22].	31
2.4	Exemple de modèles d'annulations [22].	32
2.5	Les éléments utilisés dans YAWL [24].	32
2.6	Exemple de diagramme YAWL [23].	34
3.1	Pyramide de modélisation à quatre niveaux [12].	38
3.2	Les relations dans l'IDM [2].	39
3.3	Les standards de l'Architecture Dirigée par les Modèles [12].	40
3.4	Principe des transformations de modèles [7].	41
3.5	les types de transformation et leur principale utilisation [2].	43
3.6	Les modèles et les transformations dans l'approche MDA [12].	43
3.7	Les approches de transformations de modèles [2].	45
4.1	Le principe de l'application d'une règle [2].	52
4.2	Graphe non orienté.	53
4.3	Graphe G et Sous-graphe H.	53
4.4	Graphe orienté.	53
4.5	Graphe orienté étiqueté.	54
4.6	L'interface de l'outil AToM ³	55
4.7	Le méta-modèle du diagramme d'activité proposé.	56
4.8	Le méta-modèle du modèle YAWL proposé.	60
4.9	Environnement de diagramme d'activité sous AToM ³	63

4.10 Environnement de modèle YAWL sous AToM ³	63
4.11 Exemple de diagramme d'activité sous AToM ³	79
4.12 Graphe intermédiaire de l'exemple.	80
4.13 Le modèle YAWL.	80
4.14 l'action initiale sous AToM ³	82
4.15 l'action finale sous AToM ³	86
4.16 modèle YAWL durant l'exécution.	86
4.17 le fichier « .yawl » généré.	87
4.18 le fichier XML Schema	88
4.19 exemple de modèle YAWL charger sur l'outil.	89
4.20 exemple de modèle YAWL avec un problème de solidité.	90
4.21 Les résultats de vérification.	90

Liste des abréviations

UML	Unified Modeling Language
OMT	Object Modeling Technique
OOSE	Object Oriented Software Engineering
OMG	Object Management group
API	Application Programming Interface
BPM	Business Process Management
BPMN	Business Process Modeling Notation
BPMI	Business Process Management Initiative
EPC	Event-Process Chains
YAWL	YetAnother Workflow Language
MOF	Meta Object Facility
MDA	Model Driven Architecture
IDM	Ingénierie Dirigée par les Modèles
PIM	Platform Independent Model
PSM	Platform Spécifique Model
CIM	Computation Independent Model
PDM	Platform Description Model
OCL	Object Constraint Language
XMI	XML Metadata Interchange
AToM³	A ToolForMulti-formalism and Meta-Modeling
LSH	Left Hand Side
RSH	Right Hand Side

Introduction générale

Dans le domaine du génie logiciel, les modèles des systèmes sont généralement représentés par deux approches à savoir l'approche semi-formelle et l'approche formelle. L'approche semi-formelle consiste en l'utilisation d'un mélange de graphes et de textes, elle est plus facile à comprendre et à communiquer. L'approche formelle repose sur de solides notations et de preuves mathématiques, elle est très importante pour permettre l'analyse et la vérification des systèmes.

L'un des langages les plus utilisés est sans doute le langage UML (Unified Modeling Language). C'est un langage de modélisation visuel développé pour modéliser les systèmes orienté-objet (OO). Il est devenu un standard incontournable pour spécifier, visualiser, construire et documenter les artefacts de tels systèmes logiciels.

Le besoin d'adaptation rapide des entreprises à de nouveaux contextes du marché, afin d'offrir aux clients de meilleurs services dans les plus courts délais, a créé des demandes d'établir des méthodologies de modélisation pour organiser et coordonner leurs actions, ou plus exactement leur processus d'entreprise qui est le processus de mise en oeuvre au sein d'une organisation et dont les sorties répondent aux besoins d'un client interne ou externe à cette organisation. Aujourd'hui, le concept de processus métier occupe une place majeure dans le domaine des systèmes d'information. Toutefois, ces processus doivent être en constante évolution afin de permettre aux entreprises de répondre aux exigences du marché.

Il existe plusieurs façons pour modéliser un processus métier. Chacune avec son propre vocabulaire (les éléments graphiques ou les primitives d'un langage de bas niveau) et sa propre grammaire. Dans notre travail nous focalisons sur le modèle YAWL, qui est un langage de modélisation de processus métier.

YAWL (Yet Another Workflow Language) est un langage à la fois graphique et d'exécution destiné à représenter les processus métier en étendant la syntaxe des réseaux de

Petri pour qu'elle supporte tous les patrons de flux de contrôle, tels que la modélisation de l'instance multiple, les tâches composites, le retrait des jetons et les transitions connectées directement.

Donc, UML et YAWL ont des caractéristiques complémentaires : UML peut être utilisé pour la modélisation alors qu' YAWL peut être utilisé pour la vérification, et un passage des diagrammes d'activités UML vers des modèles YAWL sera très bénéfique car il permet d'interconnecter deux communautés de modélisation indépendantes.

UML est un langage riche et largement utilisé, mais les modèles UML restent toujours en besoin d'être vérifiés pour assurer que le comportement spécifié dans ces modèles est correcte et que ce comportement répond exactement aux besoins fonctionnels du système. Cela est dû de la nature graphique et semi-formelle du langage UML caractérisée par une sémantique qui n'est pas formellement spécifiée. Ce mémoire vise à leur associer une plate-forme de vérification formelle afin d'assurer les activités d'analyse nécessaires. De l'autre côté, le modèle YAWL est un langage qui donne la possibilité de vérification à cause de sa nature formelle.

Notre travail est inscrit dans le contexte de l'IDM (Ingénierie dirigée par modèles) on exploitant la transformation des modèles, avec un objectif de proposer une approche automatique de transformation des diagrammes d'activités UML vers YAWL basée sur la transformation de graphe.

L'objectif de l'approche est de proposer et développer :

1. Un outil de modélisation des diagrammes d'activités et des modèles YAWL.
2. Une grammaire de graphe qui contient des règles pour la transformation des diagrammes d'activités vers le modèle YAWL. Pour réaliser la transformation, il existe plusieurs outils dans la littérature, par exemple (AGG, VIATRA2, et VMTS). Nous avons opter pour l'utilisation de l'outil AToM³ (A Tool for Multi-formalism and Meta-Modelling) pour créer les méta-modèles et éditer leur apparence visuel pour les deux formalismes, puis il génère l'outil de modélisation intégré, ainsi de spécifier les règles de transformation.
3. Ensuite, et dans le but de valider notre approche, nous devons transformer les

modèles obtenus (YAWL) vers le format textuel XML afin de pouvoir les charger dans un outil de vérification à savoir l'outil YAWL 4.2, pour la vérification.

Nous avons organisé notre mémoire en quatre chapitres :

- Dans Le premier chapitre, nous définissons brièvement les diagrammes UML 2 en concentrons sur le diagramme d'activité et ses composants.
- Dans Le deuxième chapitre, nous présentons le langage de modélisation de processus métier YAWL en décrivant les différents éléments de ce dernier.
- Le troisième chapitre est consacré à la présentation de l'approche IDM et la transformation de modèle. En plus, nous proposons dans ce chapitre les règles théoriques de transformation de diagramme activité vers YAWL.
- Enfin, nous détaillons dans le dernier chapitre, notre Approche de transformation qui est basée sur la transformation des graphes en utilisant l'outil AToM³.

Enfin, nous terminons notre travail par une conclusion générale dans laquelle nous résumons les points essentiels de ce travail et des perspectives pour des éventuelles améliorations dans le futur.

Le diagramme d'activité d'UML 2.0

1.1 Introduction

Au milieu des années 90, plusieurs méthodes objet étaient disponibles, mais aucune méthode ne prédomine. L'unification et la normalisation des trois méthodes dominantes, à savoir Booch, OOSE et OMT, sont à l'origine de la création du langage UML.

Actuellement, le langage UML est devenu un standard largement accepté dans l'industrie de développement de logiciels orienté objet. UML est un langage de modélisation graphique utilisé pour la description des modèles dans le processus de développement des systèmes. Il fournit plusieurs diagrammes pour la modélisation de la structure des systèmes ainsi que leur comportement.

Le but de ce chapitre est de faire introduire le langage UML et ses diagrammes en focalisant sur le diagramme d'activité.

1.2 Modélisation

1.2.1 Définition

La modélisation d'un système représente l'opération d'abstraction de leurs concepts. Elle est caractérisée par des avantages tels que : la facilité de compréhension du fonctionnement des systèmes avant sa réalisation, apporte une grande rigueur et facilite la comparaison des solutions de conception avant leur développement. Cette démarche se fonde sur des langages de modélisation, qui permettent de s'affranchir des contraintes des langages d'implémentation [1].

1.2.2 Les types de modélisation

Selon le degré du formalisme des langages ou des méthodes impliquées dans le processus de la modélisation ils sont classifiés en trois types : formelle, semi-formelle ou informelle.



La figure 1.1 présente une définition des catégories de langages et des exemples de langages ou de méthodes qui y appartiennent [2].

Categories de langages			
Langage informel		Langage semi-formel	Langage formel
Simple	Standardisé		
Langage qui n'a pas un ensemble complet de règles pour restreindre une construction.	Langage avec une structure, un format et des règles pour la composition d'une construction.	Langage qui a une syntaxe définie pour spécifier les conditions sur lesquelles les constructions sont permises.	Langage qui possède une syntaxe et une sémantique définies rigoureusement. Il existe un modèle théorique qui peut être utilisé pour valider une construction.
Exemples de langages ou méthodes			
Langage naturel.	Texte structuré en langage naturel.	Diagramme Entité-Relation, Diagramme à Objets.	Réseaux de pétri, Machines à états finis, VDM, Z.

FIGURE 1.1 – Catégorisation des langages [2].

✓ Modélisation formelle :

Une méthode formelle est un processus de développement rigoureux basé sur des notations formelles avec une sémantique définie. Ces notations formelles se caractérisent par leur capacité à exprimer un sens précis, ce qui permet de vérifier la cohérence du système et sa complétude [2].

✓ Modélisation semi-formelle :

Le processus de modélisation semi-formelle est basé sur un langage textuel ou graphique pour lequel on définit une syntaxe précise. La modélisation semi-formelle permet d'effectuer des contrôles et de réaliser des automatisations pour certaines tâches, bien que la sémantique de ce langage souvent très faible. La plupart des méthodes de modélisation semi-formelles s'appuient sur des langages graphiques [2].

✓ Modélisation informelle :

Le processus de modélisation informelle à base de langages informelles, se justifie pour plusieurs causes :

- ◆ Le langage est facile à comprendre.
- ◆ Elle représente une manière familière de communication entre personnes.

L'utilisation d'un langage informel rend la modélisation imprécise et parfois ambiguë [2].

1.3 Langage de modélisation UML

UML se définit comme un langage de modélisation graphique et textuel qui permet de modéliser les activités des différents domaines et n'est pas seulement les applications informatiques. Plus clairement, UML permet d'offrir des outils d'analyse, de conception et d'implémentation des systèmes logiciels, et pour la modélisation d'entreprise et des systèmes non logiciels [3].

Le langage UML n'est pas un langage de programmation, mais il existe des outils qui peuvent être utilisés pour générer le code en plusieurs langages à partir de diagrammes UML [4].

1.4 Historique de langage UML

En 1994, parmi les plusieurs méthodes de conceptions objets existants émergèrent trois méthodes importantes sont : La méthode OMT de James Rumbaugh, BOOCH'93 de Grady Booch, OOSE de Ivar Jacobson. Ces trois analystes ont décidé de constituer un langage commun, en 1996 la version 0.9 d'unified modeling language UML a été présentée [5].

En janvier 1997, la version 1.0 d'UML est proposée initialement à l'OMG qui accepte en Novembre 1997 dans la version 1.1 où UML a été normalisé par l'OMG comme un standard international. L'évolution de langage UML est continué jusqu'à la version actuelle qui est la version 2.5 depuis septembre 2013 [5].

La figure ci-dessous montre l'évolution de langage UML.

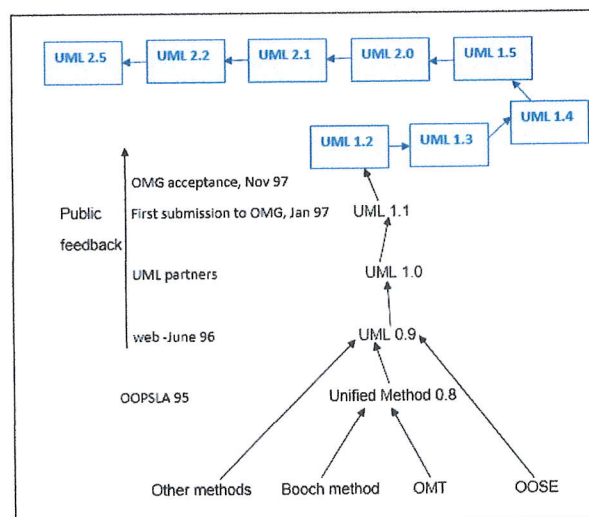


FIGURE 1.2 – Historique de langage de modélisation UML [6].

1.5 les avantages et les inconvénients de langage UML

1.5.1 les avantages

- ❖ UML est un langage normalisé et largement accepté aujourd'hui [7].
- ❖ UML est conceptuellement riche, il y a beaucoup de concepts intéressants ont été incorporés à UML en raison des besoins pratiques [7].
- ❖ UML est un langage d'usage universel pour la modélisation des systèmes à cause de sa souplesse et polyvalence [6].
- ❖ UML est un support de communication facile et performant [6].
- ❖ UML propose un cadre d'analyse et permet la représentation des éléments abstraits et complexes, et aussi la compréhension est facile [6].

1.5.2 les inconvénients

- ❖ La sémantique d'UML est floue ou mal définie [8].
- ❖ UML nécessite un apprentissage et l'expérience [6].
- ❖ UML n'est pas une méthode, certaines méthodes intègrent au moins partiellement l'utilisation d'UML [6].
- ❖ L'intégration d'UML dans un processus et améliorer un processus est une tâche complexe et longue [6].

1.6 Les vues d'un système

Une façon de mettre en oeuvre UML est de considérer différentes vues d'un système, Ces vues sont indépendantes et complémentaires qui permettent de définir l'architecture du système modélisé [9] [8] [9].

- ◆ **Vue des cas d'utilisation** : c'est une description du modèle comme vu par les acteurs du système lui-même [8].
- ◆ **Vue d'implémentation** : indique les différentes ressources nécessaires à la réalisation du projet, comme : les bases de données. Cette vue permet de regrouper les composants de système en modules après de créer les liens entre ces composants [8].
- ◆ **Vue des processus** : représente la vue temporelle et technique en manipulant des notions tels que : les tâches concurrentes, la synchronisation, les processus, etc [9].
- ◆ **Vue logique (conceptuel)** : c'est la définition du système vu de l'intérieur. Elle explique comment peuvent être satisfaits les besoins des acteurs [9].

- ◆ **Vue de déploiement** : cette vue décrit la position géographique et l'architecture physique de chaque élément du système [9].

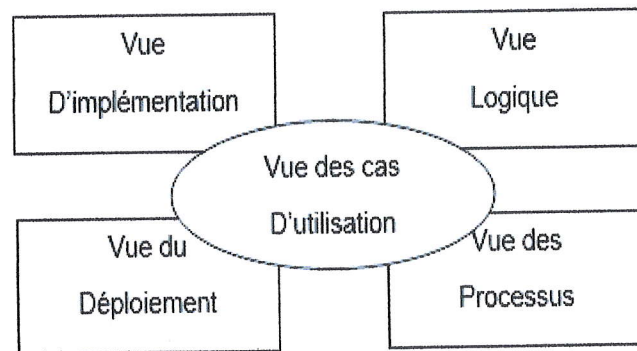


FIGURE 1.3 – Modélisation de l'architecture d'un système [9].

1.7 Les diagrammes UML

Un diagramme donne à l'utilisateur un moyen de visualiser et de manipuler des éléments de modélisation. Les diagrammes incluent des éléments graphiques qui décrivent le contenu des vues [7]. UML modélise le système sur deux aspects :

- L'un concerne la structure du système.
- L'autre concerne sa dynamique de fonctionnement [10].

La figure 1.4 montre la hiérarchie des diagrammes d'UML 2.0.

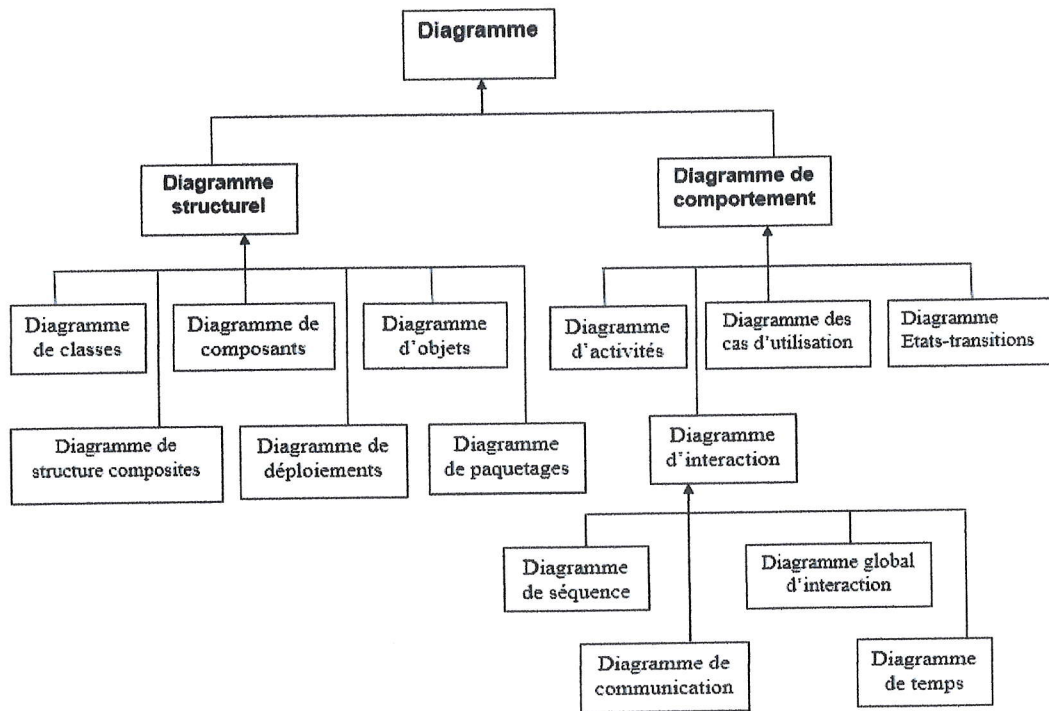


FIGURE 1.4 – La hiérarchie des diagrammes d'UML 2.0 [11].

1.7.1 Diagrammes structurels ou diagrammes statiques

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

✓ *Diagramme de classes*

Le diagramme de classes permet d'exprimer de manière générale la structure statique d'un système, en termes de classes et de relations entre ces classes qui contiennent des attributs, des opérations et des relations entre eux.

✓ *Diagramme d'objets*

Le diagramme d'objet illustrant par des exemples un diagramme de classe. Il montre des objets et des liens entre ces objets (les objets sont des instances de classes dans un état particulier).

✓ *Diagramme de composants*

Il montre les composants du système d'un point de vue physique, tels qu'ils sont mis en oeuvre (fichiers, bibliothèques, bases de données...). Il montre la mise en oeuvre physique des modèles de la vue logique avec l'environnement de développement.

✓ *Diagramme de déploiement*

Le diagramme de déploiement 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 noeuds, connectés par un support de communication.

✓ *Diagramme des paquetages*

Le paquetage est un conteneur logique permettant de regrouper et d'organiser les éléments dans le modèle UML (classe, association ...), il sert à représenter les dépendances entre paquetages.

✓ *Diagramme de structure composite*

Le diagramme de structure composite de présenter la décomposition d'une classe, en un ensemble de structures internes. Ces structures sont des ensembles d'éléments interconnectés avec leurs relations [11].

1.7.2 Diagrammes comportementaux ou diagrammes dynamiques

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

✓ *Diagramme des cas d'utilisation*

Les cas d'utilisation sont une description du comportement du système étudié selon le point de vue de l'utilisateur, sous forme d'actions et de réactions. Donc, le diagramme de cas d'utilisation, permet d'identifier l'interaction entre le système et les acteurs.

✓ *Diagramme d'activité*

Un diagramme d'activité est un diagramme graphique qui permet de représenter le comportement d'une méthode ou le déroulement d'un cas d'utilisation comme il est une variante des diagrammes d'états-transitions.

✓ *Diagramme états-transitions*

Le diagramme d'états-transitions est le seul diagramme, de la norme UML, à offrir une vision complète de l'ensemble des comportements des éléments du système. Un diagramme état de transition est composé d'un ensemble d'états, reliés par des arcs orientés qui décrivent les transitions. Il permet de décrire le comportement du système ou de ses composants sous forme de machine à états finis.

✓ *Diagramme de séquence*

Le diagramme de séquence représente le déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs. Il peut être utilisé dans la clarification d'un cas d'utilisation.

✓ *Diagramme de communication*

Le diagramme de communication c'est une représentation simplifiée d'un diagramme de séquence, Il se concentre également sur les échanges de messages entre les objets.

✓ *Diagramme global d'interaction*

C'est une variante d'un diagramme d'activités il permet de décrire les enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences.

✓ *Diagramme de temps*

Le diagramme de temps sont utilisés pour explorer le comportement des objets d'un système à travers une période déterminée [11].

1.8 Diagramme d'activité

1.8.1 Définition

UML représente l'aspect dynamique de système à l'aide de plusieurs diagrammes comportementaux, parmi eux les diagrammes d'activités qui sont conçus pour la modélisation du cheminement de flot de contrôle (toutes les instructions, branches, chemins...etc.). Et de flot de données (toutes les définitions de variable, les utilisations...). Le diagramme d'activité représente graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation, il considéré aussi comme un outil de modélisation des systèmes workflows, des modèles orientés service et des processus métiers. Par exemple, on peut utiliser un diagramme d'activités pour modéliser les étapes de la création ou de gestion d'un compte [6].

1.8.2 Intérêts des diagrammes d'activité

Les intérêts des diagrammes d'activités sont : [12] [3] [7]

- ✓ Pour avoir un changement d'état pourront s'effectuer simplement à la fin des actions donc ne pas nécessiter l'existence d'événements de transition [12].
- ✓ Modéliser les comportements internes, d'une classe, d'un cas d'utilisation ou d'une opération sous forme d'une succession d'actions [3].
- ✓ Utilisée pour représenter les successions d'états synchrones, alors que les diagrammes d'états-transitions sont utilisés principalement pour représenter les suites d'états asynchrones [3].
- ✓ Ce diagramme permet la représentation du parallélisme, qui offre un excellent outil de modélisation de workflows [7].

1.8.3 Composants d'un diagramme d'activité

Nous présentons dans cette partie les éléments principales de modélisation de base des diagrammes d'activité.

1- Les noeuds

• Les activités

Les activités donnent une description complète des traitements associés à des comportements au sens interaction d'UML. 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 noeuds reliés par des arcs [3].




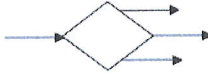
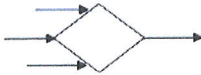
• Définition de noeud d'activité

Un noeud d'activité est un type d'élément abstrait permettant de représenter les étapes le long du flot d'une activité [3].

Il y'a trois types de noeud d'activité : Noeud exécutable, Noeud de contrôle, Noeud d'objet.

1-1- Noeuds de contrôle

Il existe plusieurs types de noeuds, le tableau ci-dessous montre l'essentiel de ces éléments :

Elément	Notation	Description
Noeud initial		Un noeud initial est un noeud de contrôle possède un arc sortant et pas d'arc entrant, à partir de laquelle le flux commence. Dans une activité on peut avoir plusieurs noeuds initiaux.
Noeud d'activité final		Lorsqu'un arc entrant à un noeud final d'activité est activé, l'exécution de l'activité en cours s'achève, et tout noeud ou flux actif au sein de cette activité est abandonné.
Noeud de flux final		L'arrivée du flux d'exécution à un noeud final de flux, implique la terminaison du flux de ce dernier. Mais cette fin n'a aucun effet sur les autres flux actifs de l'activité.
décision		Un noeud de décision est un noeud de contrôle permet de faire un choix entre plusieurs flux sortants en fonction de la condition de garde qui est associée à chaque arc sortant. Le modèle est mal formé si aucun arc en sortie n'est franchissable, et l'utilisation d'une garde [else] est recommandée car elle garantit un modèle bien formé. Dans le cas où plusieurs arcs sont franchissables (plusieurs conditions de garde sont vraies), seul l'un d'entre eux est retenu et ce choix est non déterministe.
Fusion (merge)		Un noeud de fusion est un noeud de contrôle pour accepter un flot en sortie parmi plusieurs flots en entrée n'est pas pour synchroniser les flots.

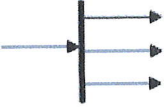

Noeud de bifurcation (fork)		Le noeud de contrôle qui joue le rôle de séparation de flux d'entrée en plusieurs flots en sortie concurrents c'est le noeud de bifurcation.
Noeud d'union (join)		Un noeud d'union est un noeud de contrôle qui utilisé pour synchroniser des flots concurrents. Lorsque tous les arcs entrants sont activés, donc l'arc sortant l'est également.

TABLE 1.1 – Les noeuds de contrôle [11] [13] [3].

1-2- Noeud objet

Un noeud d'objet est une méta-classe abstraite permettant de définir les flux d'objets dans les diagrammes d'activité. Un objet est généré par une action dans une activité et utilisé par d'autres actions, Ils sont représentés par des rectangles [3].

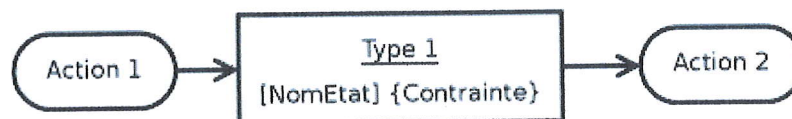


FIGURE 1.5 – Notation de noeud d'objet [7].

1-3- Noeud exécutable

Un noeud exécutable est un noeud d'activité qui peut être exécuté. Il possède un ou plusieurs gestionnaires d'exception qui peut capturer les exceptions levées par le noeud. Le noeud exécutable sera exécuté dans le cas où une exception se produite lors de l'exécution d'un autre noeud exécutable [14].

La figure ci-dessous montre un exemple d'exception.

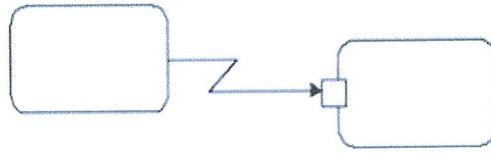


FIGURE 1.6 – Notation d'exception [7].

1-3-1- Action

Parmi les noeuds exécutables on trouve les actions qui est l'unité fondamentale de la spécification de comportement. elle possède un ensemble d'entrées qui convertit en un ensemble de résultats [14].

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,...) [11].

Dans le langage UML il y a différents types d'action parmi elles ont cité :

➤ Action appeler

L'action appeler correspond à des appels de procédure ou de méthode. Dans les deux cas, il est possible de spécifier des arguments et d'obtenir des valeurs en retour. L'appelant attend la réponse de l'appelé avant de continuer son exécution [1].

➤ Action de comportement

Action de comportement correspond à l'invocation d'un comportement spécifié à l'aide d'un diagramme UML, Par exemple vous pouvez utiliser un diagramme de séquence imbriqué dans un diagramme d'activités pour illustrer le comportement d'une activité, et inversement [1].

➤ Action envoyer

Cette action crée un message et l'envoie à un objet cible, où elle peut déclencher un comportement. Il correspond à des appels asynchrones, l'appelant poursuit son exécution sans attendre que l'entité cible ait bien reçu le message. Bien adapté à l'envoi de signaux (envoyer signal) [1].

➤ Action accepter événement

L'exécution de l'action accepter événement interrompt l'exécution en cours jusqu'à la réception du type d'événement spécifié, qui est généralement un signal [15].

➤ Action accepter appel

L'action accepter appel est une variante de accepter événement pour les appels synchrones [15].

➤ Action répondre

L'action répondre correspond au return des langages de programmation. Il permet de transmettre un message en réponse d'une action de type accepter appel [15].

➤ **Action créer**

Action créer permet d'instancier un objet [13].

➤ **Action détruire**

Action détruire permet de détruire un objet [13].

➤ **Action lever exception**

L'action lever exception permet de lever une exception au cours d'un traitement. Une exception est traitée des cas particuliers [1].

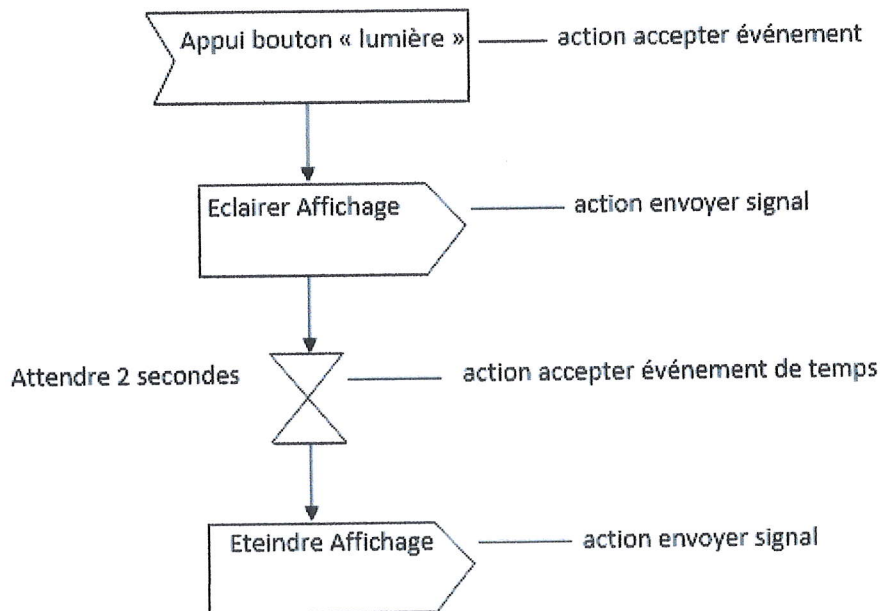


FIGURE 1.7 – Les types d'action en UML [1].

2- Les arcs

➤ Définition d'Arc d'activité

Un arc d'activité est une connexion entre deux noeuds d'activités [3].

2-1- Flot d'objet

Un flux d'objets est un arc qui permet de transmettre des données entre des noeuds d'objet [1].

2-2- Flot de contrôle

Un flot de contrôle est un arc qui décrit le séquençage de deux noeuds d'activité, ne transmet pas des données [12].

3- Partition (ligne d'eau)

Les activités peuvent impliquer de différents acteurs, tels que les différents groupes dans une organisation ou un système. Pour cela, on utilise des partitions, pour montrer à chaque action qui en est responsable. Les Partitions divisent le diagramme en colonnes comme contenu des actions qui sont menées par le groupe responsable [14].

1.9 Exemple

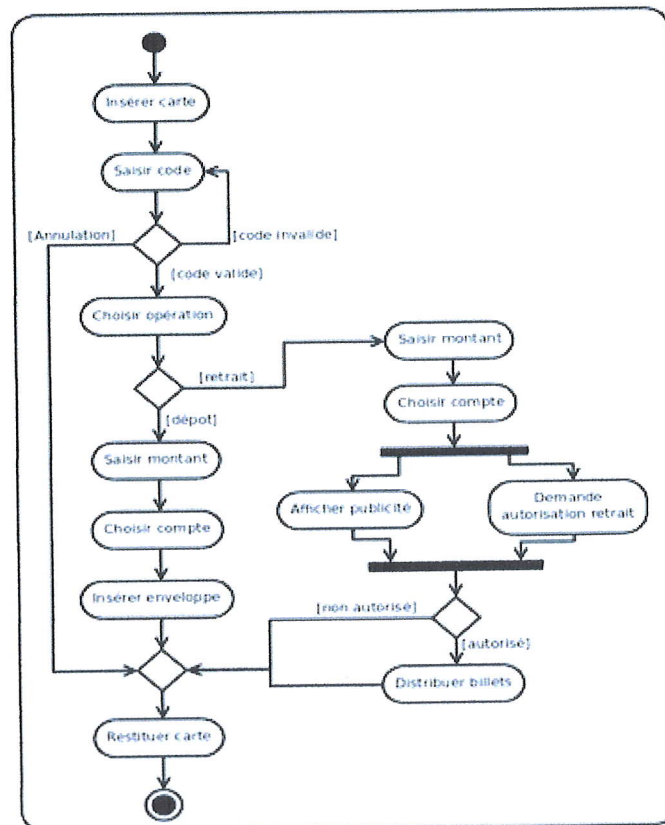


FIGURE 1.8 – Exemple d'un diagramme d'activité [13].

Dans cet exemple on présente un diagramme d'activité qui explique le fonctionnement d'une borne bancaire.

Premièrement, insérer la carte, ensuite après avoir saisi le code trois activités sont déclenchées : on choisit l'opération si le code est valide sinon si le code invalide nous ré-entrons le code, ou restituons la carte si annule l'opération. Après avoir choisit opération on trouve deux activités saisir le montant quelque soit dépot ou retrait. Dans les deux cas, nous choisissons le compte. Si nous avons choisit déposer billet nous devons insérer une enveloppe qui donne le droit de restituer la carte à la fin.

Si effectuer un retrait de billet, ont diffusé le choix de compte en deux options : afficher publicité ou demander une autorisation de retrait. Ensuite faire un choix entre les deux activités qui suit selon la garde, si la valeur de garde est autorisée retrait alors distribuer les billets et dans le cas contraire (non autorisé), nous avons restitué la carte.

1.10 Conclusion

Dans ce chapitre nous avons exposé brièvement le langage de modélisation UML. Nous avons commencé par une introduction aux modélisations des systèmes et l'adoption d'UML comme un standard de modélisation, puis nous avons montré ses différentes vues et diagrammes. Ensuite nous avons abordé en détail les diagrammes d'activités d'UML 2.0 et leur composition puisqu'il représente la base de notre travail.

Dans le chapitre suivant nous allons aborder les processus métiers et leur langage de modélisation YAWL qui représente le modèle cible de notre approche.

Le langage YAWL

2.1 Introduction

Avec la naissance des nouvelles technologies de l'information, les entreprises sont obligées d'informatiser l'ensemble des activités au tour de leur processus métier, pour un fonctionnement efficace des organisations nous adoptons sur des processus métiers robustes, et adaptés à leurs activités. La définition et l'exécution de ces processus nécessitent un modèle et des outils pour la collaboration, le déploiement et le contrôle des processus.

Dans ce chapitre, nous allons définir la notion de processus métier, les aspects de ces processus, et aussi définir la gestion de ces processus (BPM) en concentrerons sur les étapes de cycle de vie d'un processus. Par la suite nous exposons quelques langages de modélisation de processus métier et après nous allons présenter YAWL le langage de modélisation de workflow qui est le noyon de notre travail.

2.2 Le processus métier

Un processus métier représente un ensemble d'activités exécutées et coordonnées dans le cadre d'un environnement organisationnel et technique, pour réaliser un objectif prédéterminé. Généralement il est affecté à une seule organisation, mais possible d'interagir avec d'autres processus métiers qui appartient à d'autres organisations appelée partenaires [16].

2.2.1 Aspect des processus métier

Dans une entreprise un processus métier regroupe plusieurs aspects de base qui peuvent être modélisés et traités indépendamment [17].

- **Aspect Organisationnel** : Il décrit la structure organisationnelle de l'entreprise (départements, services ...) qui est invoquée pour la réalisation du processus métier.

- **Aspect Informationnel** : Il décrit les informations nécessaires pour exécuter l'activité ou le processus métier, ces informations sont manipulées par l'utilisateur du processus.
- **Aspect Comportemental** : c'est la modélisation de dynamique du processus métier, autrement dit comment les activités sont chronologiquement exécutées et les conditions qui déclenchent ces activités.
- **Aspect Opérationnel** : Il décrit l'ensemble des activités participant à un processus métier.
- **Aspect Fonctionnel** : Il décrit l'objectif à atteindre par le processus métier.

2.2.2 Gestion de processus métier (BPM)

Il existe plusieurs définitions de gestion de processus métier parmi elles nous donnons la définition de l'auteur van der Aalst et al. [18] : « BPM est l'utilisation des méthodes, des techniques et des logiciels pour modéliser, exécuter, contrôler et analyser les processus opérationnels en s'appuyant sur des acteurs qui peuvent être des êtres humains, organisations, applications, documents et autres sources d'information ».

Les processus d'entreprise démarrent le cycle de vie du BPM (la figure ??), lorsqu'ils sont créés, soit à partir de zéro, soit via la configuration d'un modèle existant. Ceci correspond à la phase de (re) **conception du processus**. La deuxième phase c'est la **configuration du système d'information** qui peut nécessiter des efforts de mise en oeuvre importants ou peut constituer une simple étape de sélection. Tout cela dépend de la technologie sous-jacente et du degré de personnalisation nécessaire. Ensuite, dans la phase de **définition et de surveillance** peut être exécuté le processus et surveillé. Enfin, lors de la phase de **diagnostic**, On peut tirer des enseignements du processus en cours pour améliorer le processus métier [19].

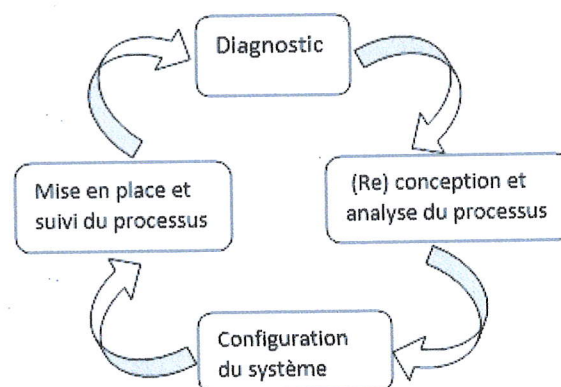


FIGURE 2.1 – Cycle de vie d'un processus métier [20].

2.2.2.1 Les niveaux d'abstraction de BPM

Généralement le processus métier est modélisé en trois niveaux d'abstractions sont : [20]

- **Niveau entreprise** : c'est les processus qui relient une entreprise à son environnement (partenaires ...). Ils utilisent pour décrire les entrées et les sorties des processus et relient les processus entre eux.
- **Niveau opérationnel** : c'est les processus qui décrivent les activités et leurs relations nécessaires à réaliser les fonctions du métier. ces processus sont modélisés de façon plus détaillés mais sans la prise en compte de leur implémentation.
- **Niveau implémentation** : Ces processus sont les spécifications techniques nécessaires à la réalisation des activités des processus.

2.3 Langages de modélisation de processus métier

Il existe plusieurs langages de modélisation de processus métier parmi eux on peut citer les suivants sont :

1. **UML** qui est déjà défini dans le chapitre 1.
2. **BPMN** (business process modeling notation) c'est un standard proposé par le consortium BPMI (Business Process Management Initiative) permettant de définir une notation graphique pour la modélisation de processus métier commune à tous les outils de modélisation.
3. **EPC** (Event-Process Chains) est un langage pour modéliser et gérer les processus par des événements, EPC représente un processus métier comme une succession de fonctions et d'événements [20].
4. Et aussi le langage **YAWL** qui sera présenté dans la prochaine section.

2.4 Langage YAWL

2.4.1 Définition de workflow

Un workflow (ou flux de travail) est l'automatisation de processus métier de laquelle les documents, informations et tâches sont transmis d'un acteur à l'autre pour action. L'objectif de workflow est la coordination automatisée de tâches réalisées par des intervenants humains [17].

- ❖ **Système de gestion de workflow**

Un système de gestion de workflow est un système pour définir les processus workflow, créer et gérer l'exécution des instances de ces processus. Un moteur de workflow pilote cette exécution, qui pouvant interpréter la définition du processus, interagir avec les différents participants et déclencher l'exécution d'un ou plusieurs programmes ou applications [20].

2.4.2 présentation de langage

YAWL (YetAnother Workflow Language) est un langage de modélisation de processus métier basé sur des modèles de workflow, il est apparu pour résoudre les limites d'expressivité des réseaux de Petri en termes de flux de contrôle. Aussi YAWL a une syntaxe XML et spécifié en termes de schéma XML. L'architecture de ce langage comprend plusieurs composants, les principaux composants comprennent un concepteur YAWL, un moteur YAWL et un référentiel YAWL [22].

Une nouvelle version de ce langage appelée newYAWL tente de couvrir tous les patrons des données et les patrons des ressources. Cette version est encore dans sa phase de recherche et elle a besoin un peu de plus de temps pour se stabiliser [23].

2.4.3 les éléments de YAWL

Un modèle de workflow est composé de tâche, de conditions et d'une relation de flux entre les tâches et les conditions sous forme de flèches unidirectionnelles [24] [25] [22].

- ✓ **Les tâches** : c'est la description d'une unité de travail qui peut être exécutée dans le cadre d'un flux de travail. Une tâche peut être effectuée manuellement par une personne ou automatiquement par un logiciel [25].

Il y a deux types de tâche : soit des tâches atomiques qui forme les feuilles de la structure de type graphe, soit des tâches composites qui fait référence à une définition de processus située à un niveau inférieur de la hiérarchie (c'est la décomposition) [22].

- ✓ Le modèle YAWL à une condition d'entrée unique (point d'entrée) et une condition de sortie unique (point d'arrivée) [26].
- ✓ Pour exprimer les situations de branchements il existe : AND-split, AND-join utilisé pour modéliser le comportement parallèle et XOR-split, XOR-join utilisés pour modéliser un routage exclusif comme illustré dans le tableau suivant : [22]

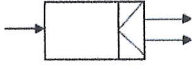
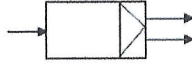
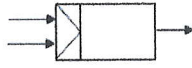
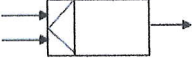
Elément	Notation	Description
AND-split		AND-split, qui divise le flux de contrôle entrant d'une branche donnée en plusieurs flux sortants exécutés simultanément.
XOR-split		XOR-Split est utilisé pour déclencher un seul flux sortant. Une fois la tâche terminée, le choix sera automatique entre plusieurs alternatives possibles.
AND-join		Une tâche avec une jointure AND attendra de recevoir tous les flux entrants avant de s'exécuter.
XOR-join		Une tâche avec une jointure XOR-Join sera capable de s'exécuter, une fois le travail terminé sur un flux entrant.

TABLE 2.1 – les situations de branchement [25] [19].

- ✓ Le langage YAWL a été enrichi par des fonctionnalités qui n'existent pas dans les réseaux de Petri telque : le modèle de synchronisation avancés, les instances multiples et l'annulation [23].

1. Modèles de synchronisation avancés :

Il existe deux types de transition OR-split et OR-join :

❖ OR-split

OR-split est utilisée pour déclencher certains flux, mais pas nécessairement tous les flux sortants vers d'autres tâches [25].

❖ OR-join

La jointure OR-Join garantit qu'une tâche attend jusqu'à ce que tous les flux entrants soient terminés ou ne se terminent jamais. La jointure OR-join c'est rarement utilisé [25].

⇒ Exemple :

Cet exemple illustre les modèles de synchronisation avancés, le registre des tâches est un "OR-split" il y a une possibilité que les trois tâches de réservation (vol, hôtel et voiture) soient

exécutées. Mais il est également possible qu'une ou deux tâches de réservation seulement soient exécutées.

La tâche pay est un "OR-join" se synchronise uniquement si nécessaire, c'est-à-dire qu'elle synchronisera uniquement les tâches de réservation réellement sélectionnées [22].

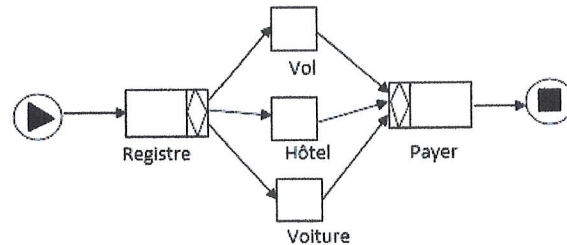


FIGURE 2.2 – Exemple de modèles de synchronisation avancés [22].

2. Modèles impliquant plusieurs instances :

Les tâches composites et les tâches atomiques peuvent avoir plusieurs instances [22].

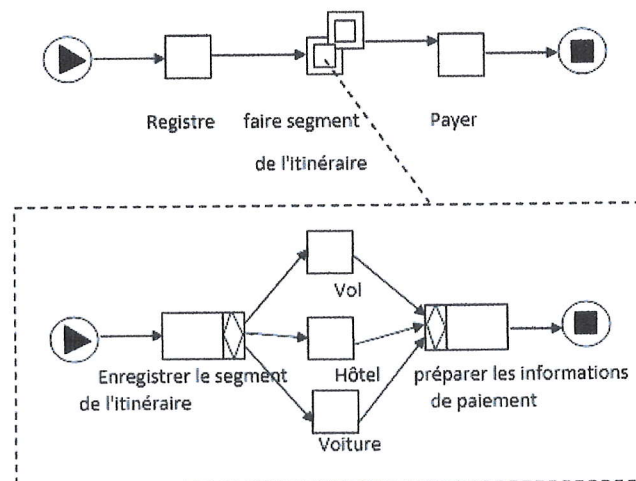


FIGURE 2.3 – Exemple de modèles impliquant plusieurs instances [22].

➡ Exemple :

Pour illustrer cette nouvelle fonctionnalité on va voir l'exemple de la figure 2.3. Un itinéraire d'un voyage peut inclure plusieurs segments. Par exemple, un voyage peut aller d'Amsterdam à Singapour, de Singapour à Los Angeles et enfin de Los Angeles à Amsterdam, entraînant ainsi trois segments d'itinéraire. Et chaque segment peut inclure une réservation d'hôtel, vol ou une réservation de voiture. Les plusieurs segments sont modélisés par plusieurs instances de la tâche [22].

3. Modèles d'annulations :

Modèle d'annulation identifie divers mécanismes permettant de déclencher l'annulation ou l'achèvement de tâches [19].

⇒ Exemple :

Prenant l'exemple précédent avec le symbole « supprimer les jetons », lorsque l'exécution d'une tâche d'annulation, tout ce qui se trouve à l'intérieur de rectangle en pointillé est désactivé autrement dit elle est possible de retirer des réservations [22].

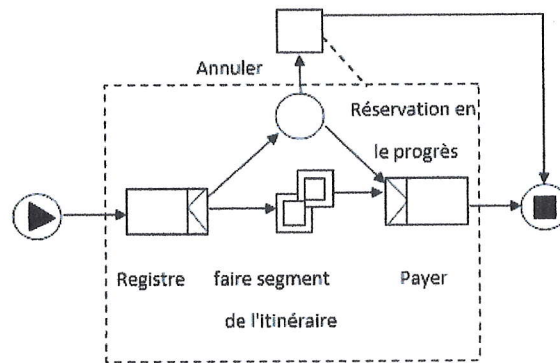


FIGURE 2.4 – Exemple de modèles d'annulations [22].

La figure ci-dessous résume les éléments de modélisation de YAWL.

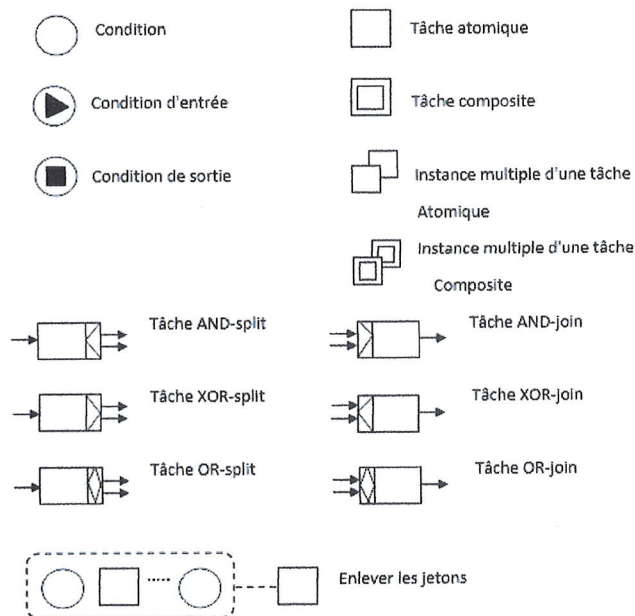


FIGURE 2.5 – Les éléments utilisés dans YAWL [24].

2.4.4 les avantages et les inconvénients

2.4.4.1 Les avantages

➤ **Plus grande expressivité :**

YAWL offre un support complet pour les modèles de flux de contrôle. C'est le langage de spécification de processus le plus puissant pour la capture de dépendances de flux de contrôle [27].

➤ **Déploiement et exécution faciles :**

YAWL est un langage qui exécute le flux de travail avec une syntaxe XML. Les spécifications de flux de travail peuvent être directement déployées et exécutées par le moteur YAWL [22].

➤ **Hériter des avantages des réseaux de Petri :**

Comme les réseaux de Petri, YAWL a une sémantique formelle appropriée. Cela rend ses spécifications sans ambiguïté et la vérification automatisée devient possible. Aussi, il offre une représentation graphique, et un environnement de conception graphique intuitif fourni par l'éditeur open source YAWL [27].

➤ **Prise en charge de la manipulation de données :**

Au début, YAWL se concentre sur le flux de contrôle, mais ensuite il a été développé pour offrir une prise en charge complète de la manipulation de données [22].

2.4.4.2 Les inconvénients

➤ **La complexité :**

Pour maîtriser le langage YAWL et utiliser le système YAWL les utilisateurs ont besoin d'une connaissance technique considérable en matière de flux de travail [22].

➤ **Interaction avec des composants logiciels limités :**

YAWL coordonne principalement les activités intégrées à un service Web. Bien que les services web sont très communs dans un scénario d'entreprise, cette solution est assez restrictive, car elle limite l'utilisation d'un langage à des contextes spécifiques [22].

2.5 Exemple

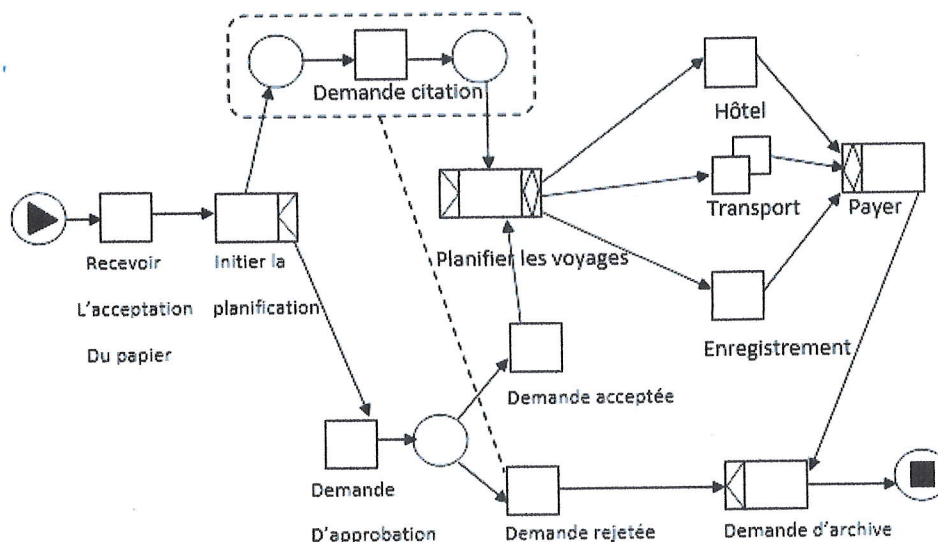


FIGURE 2.6 – Exemple de diagramme YAWL [23].

Nous représentons les éléments de langage YAWL dans un exemple qui montre comment les voyages de conférence sont planifiés dans un université.

Premièrement, on va commencer par la réception d'une acceptation de papier suivi par une planification initiale, et après cela la demande d'un devis et de l'approbation simultanément (parce que c'est and-split). La demande d'approbation est capturée par un choix qui fait par le chef d'établissement. Si l'approbation est refusée, la demande de devis est annulée (modèle d'annulation), en conséquence les documents sont classés et fermé. Si l'approbation est acceptée, et le devis a également été reçu alors le voyage peut être planifié (AND-join : attendra de recevoir les deux avant de planifier le voyage). Cette planification peut être suivie de la réservation d'un hôtel, organisation du transport pour une ou plusieurs destinations (Pour cette raison on utilise une tâche avec instances multiples) et inscription à la conférence. Lorsque toutes ces réservations sont complétées, le paiement est effectué. À la fin les documents liés au dossier peuvent être archivés [23].

2.6 Outils d'analyse des modèles YAWL

Il existe plusieurs outils de vérification des modèles YAWL, dans notre travail nous utilisons l'outil YAWL 4.2.

2.6.1 L'outil YAWL 4.2

Les spécifications de flux de travail YAWL sont conçues et vérifiées dans un environnement de conception appelé YAWL. L'éditeur est distribué sous forme d'une application Java, mais d'autres services système YAWL sont distribués sous forme de services Web.

L'outil YAWL est un outil qui vous permet de concevoir graphiquement une spécification de processus métier pouvant être enregistrée dans un fichier (.yawl) puis chargée dans le moteur YAWL pour l'exécution [38].

Les fichiers < .yawl > sont des fichiers XML qui permettent la sérialisation de n'importe quel modèle YAWL graphique.

2.7 Conclusion

Dans ce chapitre nous avons défini le processus métier, les aspects, la gestion de ces processus et aussi définit brièvement quelques langages de modélisation de processus métier. Ensuite nous avons présenté le langage de modélisation de workflowle YAWL.

Dans le chapitre suivant, nous allons présenter l'approche de l'ingénierie dirigé par le modèle (IDM) qui basée sur la notion de modèle, ensuite l'MDA la variante de cette approche.

3.1 Introduction

Depuis les années 1980, le monde de l'ingénierie informatique a connue l'émergence de l'approche objet et son principe du « tout est objet ». Ensuite, une nouvelle tendance est apparue, qui s'oriente vers l'ingénierie dirigée par les modèles (IDM) et le principe du « tout est modèle ». La possibilité d'utiliser la technologie la mieux adaptée à chacune des étapes du développement du logiciel, tout en ayant un processus global de développement unifié dans un paradigme unique. Cette unification est faite par L'IDM avec l'utilisation importante des modèles qui permet un développement souple et itératif, grâce aux raffinements et enrichissements par transformations successives. En plus, les transformations entre les modèles permettent de choisir l'espace technique et le formalisme le plus adapté à chaque activité.

Dans ce chapitre, nous présentons le principe de l'ingénierie dirigée par les modèles, et sa variante pour le génie logiciel proposée par OMG qui est l'architecture dirigée par modèles MDA. Nous exposons aussi les standards liés à cette approche, le concept de transformation des modèles MDA, ensuite on définit une classification des approches de transformation de modèles.

3.2 Ingénierie Dirigée par les Modèles

L'IDM (Ingénierie Dirigée par les Modèles) est une approche de développement logiciel, se basé à l'utilisation des modèles comme des éléments centraux tout au long du cycle de vie du logiciel [28].

L'idée de l'IDM est d'exploiter des modèles comme l'entrée du processus de développement. Il est nécessaire de formaliser les modèles pour les rendre exploitables par les machines et de réaliser des programmes permettant de traiter des modèles. Dans l'IDM, ces programmes sont regroupés sous le terme de transformations de modèles. Les deux concepts principaux de l'IDM : le méta-modélisation, et la transformation de modèles. Pour faire cela il y a des

outils comme les langages de modélisation et les langages de transformation [12].

3.2.1 Définition de Modèle

Un modèle est une abstraction simplifiée d'un système exprimé par des diagrammes, construite dans une intention particulière. Il utilise pour répondre à des questions qui pose sur le système modélisé. Mais il est encore difficile de répondre à la question « Qu'est ce qu'un bon modèle ? », Néanmoins un modèle doit être suffisant pour répondre à certaines questions du système qu'il représente [29].

Dans l'IDM la notion de modèle fait explicitement référence à la notion de formalisme ou de langage de modélisation bien défini. Un langage de modélisation est défini par une syntaxe abstraite, une syntaxe concrète et une sémantique [29].

1- La syntaxe abstraite

La syntaxe abstraite permet de décrire les concepts du langage modélisé et comment ils peuvent être combinés pour créer des modèles [14].

2- La syntaxe concrète

La syntaxe concrète définit pour chaque concept abstrait le type de notation qui sera utilisé. Il y a deux types de syntaxe concrète sont : La syntaxe textuelle permet de décrire les modèles sous forme textuelle et la syntaxe visuelle permet de décrire les modèles sous forme de diagramme [14].

3- La sémantique

La sémantique définit comment les concepts du langage doivent être interprétés par les concepteurs mais surtout par les machines [29].

3.3 L'approche MDA

3.3.1 Présentation de l'approche MDA

L'architecture dirigée par les modèles est une approche de développement de logiciels dirigée par l'OMG. Elle est basée principalement sur des modèles qui sont exprimés dans un langage de modélisation dont le méta-modèle est exprimé en MOF [2].

L'idée de base du MDA est l'élaboration de modèle, indépendants des détails techniques des plates-formes d'exécution, afin de permettre la génération automatique de la totalité du code des applications. Cette approche permet de réaliser le même modèle sur plusieurs plates-formes grâce à des projections [29].

3.3.2 l'architecture de l'approche MDA

L'approche MDA (model driven architecture) a proposé une architecture à quatre niveaux comme indiqué dans la figure suivante :

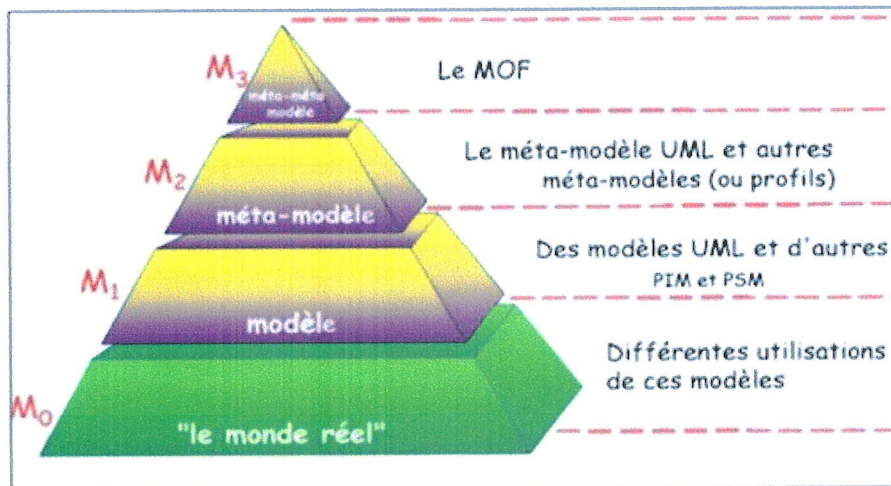


FIGURE 3.1 – Pyramide de modélisation à quatre niveaux [12].

Le niveau M0 : (ou instance)

Le niveau M0 correspond au monde réel, il est composé des informations que l'on souhaite modéliser, Instance du modèle du niveau M1 [3].

Le niveau M1 : (ou modèle)

Dans le niveau M1 construire un modèle UML comme diagramme de classe ou état de transition pour décrire les informations appartenant au niveau M0. Ces modèles sont conforme aux méta-modèles définit au niveau M2 [2].

Le niveau M2 : (ou méta-modèle)

Définit le langage de modélisation des modèles de niveau M1. Ce niveau contient le méta-modèle UML qui définit la structure interne des modèles UML. Les concepts définis par un méta-modèle sont des instances des concepts du MOF [7].

Le niveau M3 : (ou méta méta-modèle)

Il définit le langage de spécification du méta-modèle. Comme exemple Le MOF (Meta Object Facility) qui décrire lui-même [7].

➤ Donc il existe deux relations sont :

La première relation est modèle *représente* un système et la deuxième est modèle *conforme à* un méta-modèle si tous les éléments du modèle sont instance du méta-modèle,

et les contraintes exprimées sur le méta-modèle sont respectées [5]. Les relations entre modèle, méta-modèle et méta méta-modèle est illustré dans la figure 3.2.

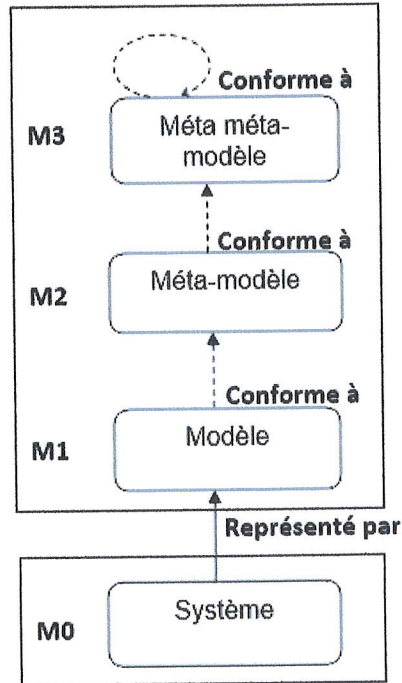


FIGURE 3.2 – Les relations dans l'IDM [2].

3.3.3 Standards liée à l'MDA

L'approche MDA basée sur un ensemble de standards de l'OMG pour résoudre les problèmes d'interopérabilité entre les systèmes d'information, Nous allons les expliquer dans la figure ci-dessous [11].

UML

Est un langage visuel semi-formel pour la modélisation des systèmes. Il permet de décrire l'architecture, les solutions et les points des vues à l'aide des diagrammes et des textes [30].

MOF

Un standard de méta-modélisation constitué d'un ensemble d'interfaces standards pour définir la syntaxe et la sémantique d'un langage de modélisation [29].

XMI

Est un standard pour l'échange de métadonnées UML basé sur XML, qui donne une représentation concrète des modèles sous forme de documents XML [11].

OCL

Est un langage informatique d'expression des contraintes intégré à UML [30].

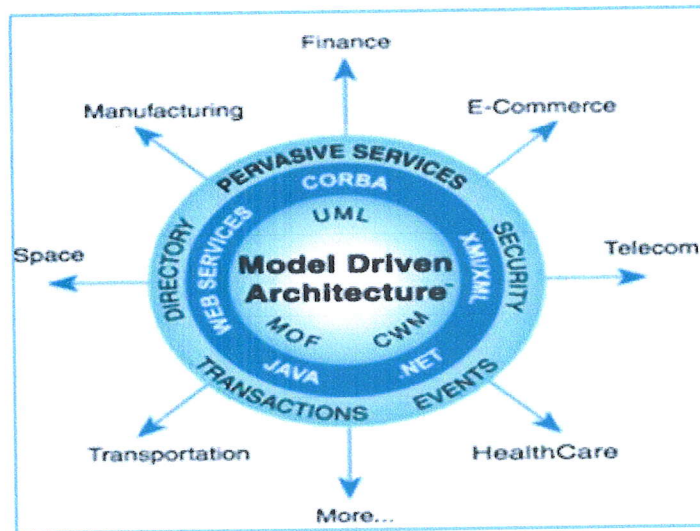


FIGURE 3.3 – Les standards de l'Architecture Dirigée par les Modèles [12].

3.3.4 Typologie des modèles dans l'approche MDA

L'OMG a défini plusieurs modèles qui ont pour rôle à modéliser l'application, en suit réaliser des transformations successives sur ces modèles jusqu'à générer le code de l'application. Il y a quatre types principaux de modèles dans l'approche MDA sont : [29]

❖ CIM (Computation Independant Model)

La première étape dans le développement d'un système est la réalisation de modèle CIM qui permet de modéliser toutes les exigences du client et définir les différentes interactions qui impliqueront le système dans ses environnements interne ou externe. Les CIM peuvent servir de référence pour s'assurer que l'application finale correspond aux demandes des clients [30].

❖ PIM (Platform Independant Model)

Les modèles PIM sont des modèles d'analyse et de conception de l'application qui décrivent le système indépendamment de plate-forme cible sur laquelle il s'exécutera [29].

Le rôle des PIM est de donner une vision structurelle et dynamique de l'application, sans détail technique [11].

❖ PDM (Platform Description Model)

Le PDM est le modèle qui décrit une plate-forme d'exécution. Il fournit un ensemble de concepts techniques représentant la plateforme d'exécution et ses services [29].

❖ PSM (Platform Specific Model)

À partir de la combinaison des modèles d'analyse et conception PIM et les modèles de plateforme PDM, facilite la génération du code. Il exprime par exemple, les événements, les composants, les instructions, les conditions, etc [30].

3.3.5 Transformation de modèle

3.3.5.1 Définition de transformation

La transformation de modèles c'est le processus essentiel dans l'ingénierie dirigée par les modèles. Elle est utilisée pour l'optimisation des modèles et d'autres formes d'évolutions des modèles.

Une transformation est la génération automatique d'un modèle cible à partir d'un modèle source, en respectant une définition de transformation. Une définition de transformation est un ensemble de règles qui décrivent comment un modèle source peut être transformée en un modèle cible [14].

3.3.5.2 Principe de transformation

La transformation est exécutée à l'aide d'un moteur de transformation qui lit le modèle source qui conforme à un méta-modèle source et produit en sortie un modèle cible conforme à un méta-modèle cible [28].

Le moteur de transformation se compose d'un ensemble des règles qui s'appliquent au méta-modèle source pour décrire comment les concepts de méta-modèle source peut être transformés en des concepts de méta-modèle cible [28].

La figure 3.4 résume le principe de transformation des modèles.

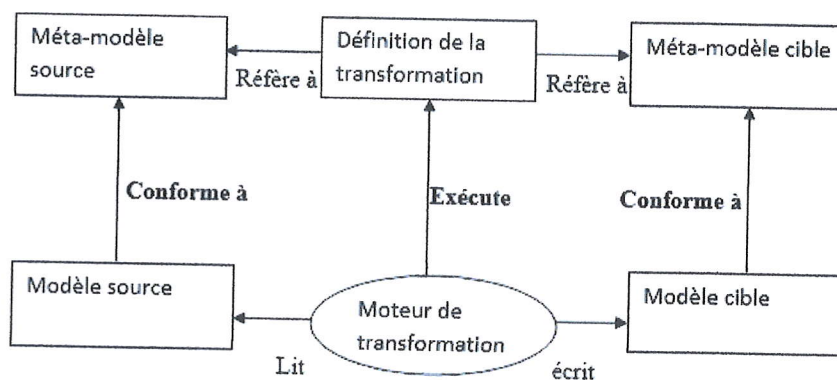


FIGURE 3.4 – Principe des transformations de modèles [7].

3.3.5.3 les types de transformation

Dans l'IDM il existe trois types de transformations de modèles : les transformations verticales, les transformations horizontales et les transformations obliques.

□ *Transformations verticales*

Le modèle source et cible d'une transformation verticale sont définis à des différents niveaux d'abstraction. Lorsque cette transformation descend le niveau d'abstraction (PIM vers PSM) c'est le raffinement, mais lorsqu'on élève le niveau d'abstraction, la transformation est dite : abstraction (PSM vers PIM) [31].

□ *Transformations horizontales*

Ces transformations gardent le même niveau d'abstraction en modifiant les représentations d'informations du modèle source (PIM vers PIM) ou (PSM vers PSM). La modification peut être un ajout, une modification, une suppression ou une restauration [31].

□ *Transformations obliques*

C'est une combinaison entre les deux types précédents de transformation. Ce type de transformation est utilisée par les compilateurs, qui effectuent des optimisations du code source avant de générer le code exécutable [31].

- ✓ Selon la nature des méta-modèles sources et cibles, nous distinguons deux types de transformations :

Une transformation *endogène* est une transformation dont les modèles sources et cibles sont conformes au même méta-modèle sinon elle se dit *exogène* si les deux méta-modèles sont différents [5].

La figure suivante illustre ces types de transformations de modèles et leur principale utilisation.

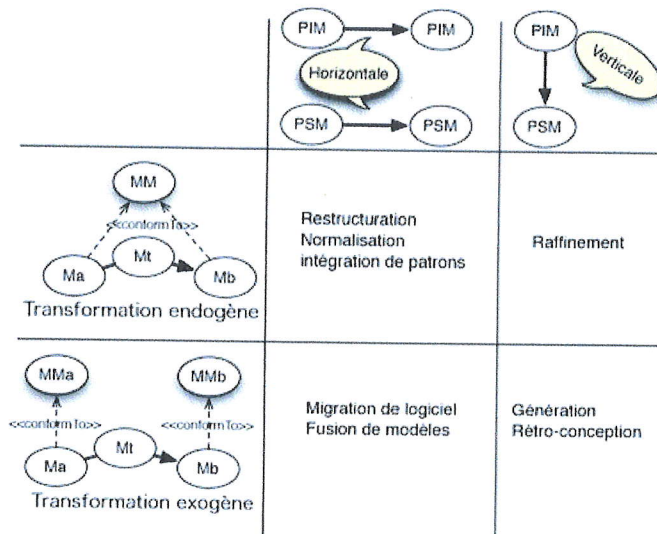


FIGURE 3.5 – les types de transformation et leur principale utilisation [2].

3.3.6 Transformation de modèles dans MDA

Plusieurs types de transformations de modèles sont définis dans l’approche MDA, nous illustrerons ces types dans la figure 3.6.

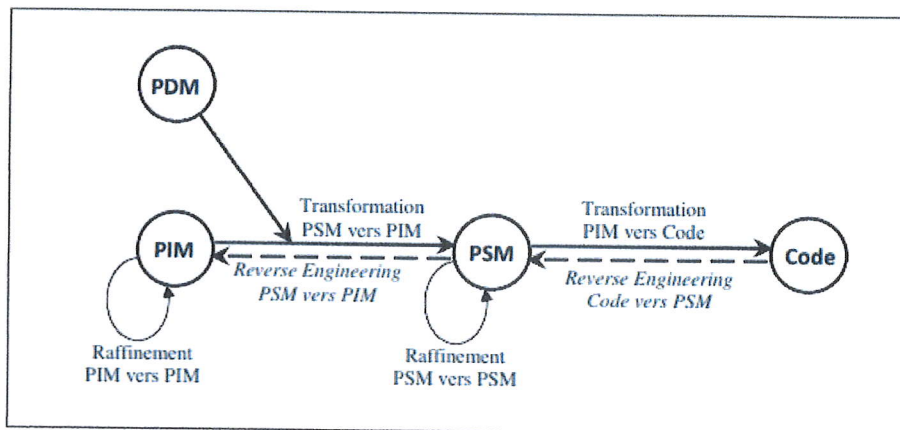


FIGURE 3.6 – Les modèles et les transformations dans l’approche MDA [12].

❑ Transformations PIM vers PIM et PSM vers PSM

Les transformations de type PIM vers PIM et PSM vers PSM visent à enrichir, filtrer ou spécialiser le modèle. Il s’agit de transformations de modèle à modèle.

❑ Transformations PIM vers PSM

La transformation de PIM vers PSM permet d'ajouter au PIM des informations spécifiques à la plate-forme d'exécution ciblée en s'appuyant sur les informations fournies par le modèle PDM. Elle n'est effectuée qu'une fois le PIM suffisamment raffiné.

□ Transformations PSM vers Code

La transformation de PSM vers le code (transformation de type modèle à texte) est la génération du code. Parfois le code est assimilé à un PSM exécutable, généralement n'est pas possible d'obtenir la totalité du code à partir du modèle, alors il est nécessaire de le compléter manuellement.

□ Transformations PSM vers PIM et Code vers PSM

Ces transformations sont des opérations de rétro-ingénierie, est une opération très difficile et complexe à réaliser. Si le code n'a pas été conçu dans la démarche du MDA, il faut faire un appel aux techniques traditionnelles de rétro-ingénierie pour pouvoir effectuer telles opérations [29].

3.3.7 Propriété de transformation des modèles

Une transformation des modèles est caractérisée par des propriétés, nous allons définir certaines propriétés :

❖ *Réversibilité* :

Une transformation est dite réversible si elle peut se faire dans les deux sens, c'est-à-dire capable de retrouver le modèle source à partir du modèle cible. Par exemple transformation modèle au texte et texte au modèle [2].

❖ *Traçabilité* :

La traçabilité est la propriété d'avoir un dossier des liens entre les éléments des modèles de transformation ainsi que les différentes étapes du processus de transformation. Ces liens peuvent être stockés soit dans le modèle source, soit le modèle cible ou dans un modèle à part [2].

❖ *Modularité* :

Une transformation modulaire permet de mieux modéliser les règles de transformation en faisant un découpage du problème. Un langage de transformation de modèles supportant la modularité facilite la réutilisation des règles de transformation [2].

❖ *Ordonnement des règles* :

C'est la suite des règles à exécuter lors d'une transformation. Il y a deux types d'ordonnement : implicite si l'algorithme d'ordonnement est défini par l'outil de transformation, sinon est dite explicite si d'autres mécanismes spécifient l'ordre d'exécution des règles [30].

❖ L'organisation des règles :

C'est une organisation qui définit la stratégie selon laquelle les règles seront appliquées. Ces règles peuvent être organisées de façon modulaire avec importation. Et aussi selon une structure dépendante du modèle source ou du modèle cible [30].

❖ L'incrémentalité :

Cette propriété est liée à l'aptitude des modèles cibles à s'adapter aux changements des modèles sources [30].

❖ Réutilisabilité :

La réutilisabilité permet de réutiliser des règles de transformation dans d'autres transformation de modèles. L'identification de patrons de transformation est un moyen pour mettre en oeuvre cette réutilisabilité [2].

3.3.8 Classification des approches de transformation de modèles

Czarnecki et Helsen ont proposé une classification, contenant deux grandes classes de transformations de modèles : les transformations modèle vers modèle et les transformations modèle vers code comme illustré dans la figure ci-dessous [2].

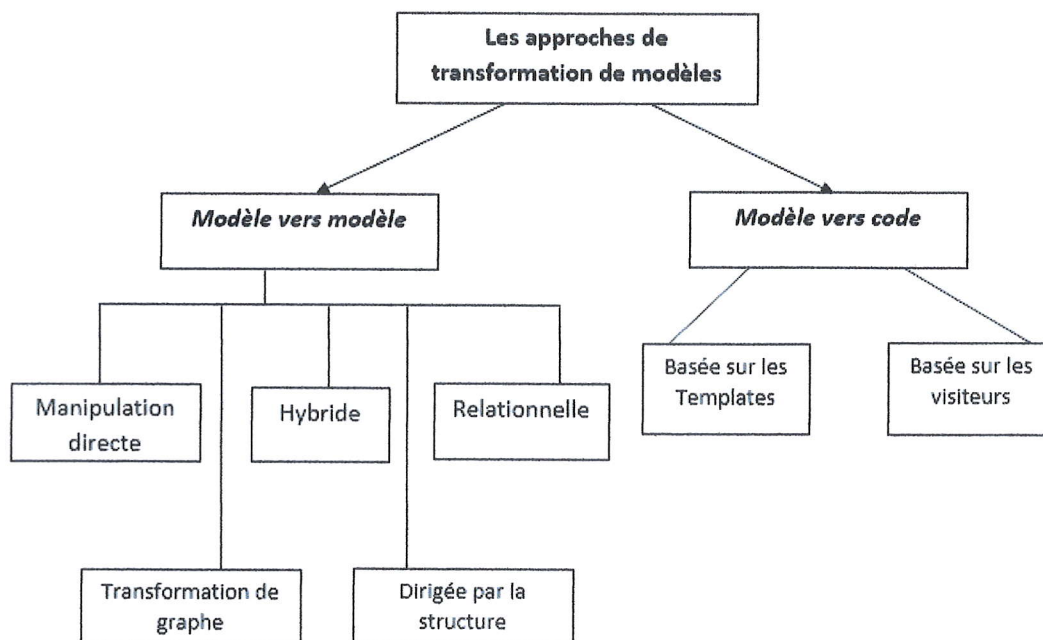


FIGURE 3.7 – Les approches de transformations de modèles [2].

1- Transformations de type Modèle vers Modèle

La transformation de type modèle vers modèle consiste à transformer un modèle source à un modèle cible, ces modèles peuvent être conformé de différents méta-modèles [11].

On peut distinguer plusieurs catégories dans ce type de transformation :

➤ **Approche par manipulation directe :**

Pour manipuler la représentation interne des modèles source et cible, cette approche est basée sur l'utilisation des API (Application Programming Interface) [11].

➤ **Approche relationnelle :**

Le principe de base de ces approches consiste à établir les relations entre les éléments de modèle source et cible à l'aide des contraintes en utilisant une logique déclarative reposant sur des relations mathématiques [11].

➤ **Approche dirigée par la structure :**

Dans ces approches la transformation est divisée en deux phases : la première c'est la création d'une structure hiérarchique du modèle cible, et le deuxième consiste à définir les valeurs des attributs et des références pour compléter le modèle [11].

➤ **Approche basée sur les transformations des graphes :**

Ces approches se basent sur les grammaires de graphes. Les règles de transformation sont définies pour des fragments de modèles, qui peuvent être exprimés dans les syntaxes concrètes des modèles sources et cibles respectivement, ou dans leur syntaxe abstraite. Chaque règle est composée d'un graphe source (LHS : Left Hand Side) et d'un graphe cible (RHS : Right Hand Side) [3].

➤ **Approche hybride :**

C'est la combinaison de différentes techniques. On peut citer des approches utilisant des règles à logique déclarative et des règles à logique impérative à la fois comme XDE et ATL. [30].

2- Transformations de type Modèle vers Code

Il existe deux approches de transformations de type modèle vers code la première basée sur le principe de visiteur, ou celle basée sur les patrons (templates).

➤ **Approche basée sur le visiteur :**

Transformer le modèle en code en lui ajoutant des mécanismes de visiteur pour traverser la représentation interne d'un modèle et créer le code [3].







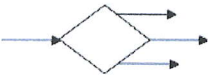

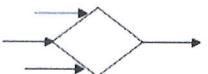
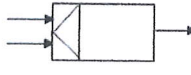

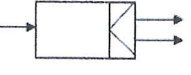
➤ **Approche basée sur les templates :**

Dans les outils MDA, ces approches sont actuellement très utilisées. En utilisant les fragments de méta-code du code cible pour accéder aux informations de modèle source [12].

3.4 Transformation de diagramme d'activité vers le langage Yawl

Le diagramme d'activité est un diagramme pour la représentation de l'enchaînement des activités et la modélisation du flux de contrôle entre ces activités. YAWL est un langage de workflow et aussi offre un support complet pour les modèles de flux de contrôle. Mais tous les deux sont des langages de modélisation des processus métiers.

Dans le tableau ci-dessous ont présente les règles théoriques pour le passage entre le diagramme activité et YAWL.

Diagramme d'activité	YAWL	La description
<p>Noeud initiale</p> 	<p>Condition d'entrée</p> 	<p>Un noeud initial est transformé en condition d'entrée parce que les deux composants lancent le flux de contrôle.</p>
<p>Noeud d'activité Final</p> 	<p>Condition de sortie</p> 	<p>Un noeud d'activité final est transformé en condition de sortie.</p>
<p>Noeud de flux Final</p> 	<p>Condition de sortie</p> 	<p>Un noeud de flux Final est transformé en condition de sortie.</p>
<p>Décision</p> 	<p>XOR-Split</p> 	<p>Noeud de décision est transformé en XOR-split parce que les deux éléments permettent de faire un choix entre plusieurs flux sortants.</p>
<p>Fusion (Merge)</p> 	<p>XOR-Join</p> 	<p>Noeud de fusion est transformé en XOR-join parce que les deux éléments permettent d'accepter parmi plusieurs flots en entrée, un flot en sortie.</p>
<p>Noeud de bifurcation (fork)</p> 	<p>AND-Split</p> 	<p>Noeud de bifurcation est transformé en AND-split parce que les deux ont une sémantique équivalente qui divise le flux de contrôle entrant en plusieurs flux sortant.</p>


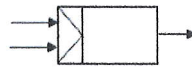




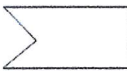



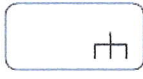

<p>Noeud d'union (join)</p> 	<p>AND-Join</p> 	<p>Noeud d'union est transformé en AND-join parce que les deux éléments font la synchronisation des flots entrants concurrents.</p>
<p>Action</p> 	<p>Tâche atomique</p> 	<p>Tâche atomique correspond à une action dans le diagramme d'activité parce que les deux sont des unités fondamentales de comportement.</p>
<p>Action envoyer signal</p> 	<p>Tâche atomique</p> 	<p>Une action signal ne peut être transformée que vers une tâche atomique.</p>
<p>Action accepter événement</p> 	<p>Tâche d'événement</p> 	<p>Une action accepter événement ne peut être transformée que vers une tâche atomique.</p>
<p>Action accepter l'événement de temps</p> 	<p>Tâche de temps</p> 	<p>Une action accepter événement de temps ne peut être transformé que vers une tâche atomique.</p>
<p>Action de comportement</p> 	<p>Tâche composite</p> 	<p>Une action de comportement est transformée en une tâche composite.</p>

TABLE 3.1 – Transformation des éléments de diagramme d'activité vers les éléments de YAWL.

3.5 Conclusion

Dans ce chapitre, nous avons consacré une bonne partie à la présentation des concepts fondamentaux de l'ingénierie dirigée par les modèles et basée sur la transformation des modèles, son principe et les types de transformation. Ainsi, nous avons fait une description sur l'approche MDA, les standards liés à cette approche, ses différents modèles existants, son principe de transformation, et certaines propriétés de transformation. Nous avons présenté aussi des règles théoriques pour le passage entre le diagramme d'activité et YAWL.

Dans le chapitre suivant nous allons utiliser la transformation de graphe pour l'implémentation des règles théoriques proposée dans ce chapitre afin de transformer le diagramme d'activité vers YAWL.

Une Approche de transformation des diagrammes d'activités vers YAWL

4.1 Introduction

Pour la modélisation des systèmes complexes, les graphes sont considérés comme moyens pratiques. Ainsi, les différents formalismes de modélisation existants (comme les diagrammes UML, réseau de pétri etc) sont des graphes. Si les graphes servent à visualiser les structures complexes des modèles d'une manière simple et intuitive, les transformations de graphes peuvent donc être exploitées pour spécifier comment ces modèles peuvent évoluer.

Dans ce chapitre, nous allons commencer par les concepts de base de transformation de graphes, suivi par l'exposition de quelques outils de transformation. Par la suite, nous exposerons en détail notre approche automatique de transformation de diagrammes d'activité UML vers le langage YAWL basée sur la transformation de graphe et en utilisant l'outil AToM³. Nous allons créer deux méta-modèles, un pour les diagrammes d'activité UML et l'autre pour le YAWL, puis nous allons développer deux grammaires de graphes, la première pour faire le passage UML vers YAWL, et la deuxième pour dériver le code XML de YAWL, prêt pour l'analyse.

4.2 Transformation de graphe

Pour l'expression de transformation de modèles on utilise la transformation de graphe. Particulièrement, les transformations de modèles visuels peuvent être naturellement formulées par des transformations de graphes, parce que les graphes sont bien adaptés pour décrire les structures fondamentales des modèles [12].

4.2.1 Principe de transformation de graphe

La transformation de graphes consiste en l'application itérative d'une règle à un graphe. Chaque application de règle remplace une partie du graphe par une autre suivant la définition

de la règle. La transformation de graphe fonctionne comme suit : la Sélection d'une règle applicable de l'ensemble des règles, l'application de cette règle au graphe d'entrée et la répétition de processus jusqu'à ce qu'aucune règle ne puisse être appliquée [30].

La transformation de graphe est basée sur un ensemble de règles respectant une syntaxe particulière, appelé modèle de grammaire de graphe. Ce modèle est composé des règles où chaque règle possède deux parties : une partie gauche (LHS : Left Hand Side) et une partie droite (RHS : Right Hand Side). La partie gauche correspond au graphe d'entrée (appelé aussi Host Graph). La partie droite de la règle, décrit la modification qui sera effectuée sur la partie gauche. La règle remplace l'équivalent de son LHS dans le graphe à transformer par sa partie droite RHS [30].

La figure 4.1 illustre le principe de l'application d'une règle sur un graphe.

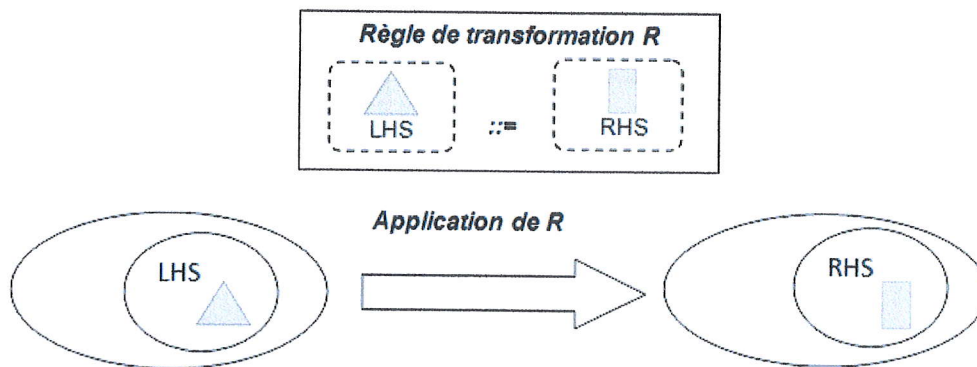


FIGURE 4.1 – Le principe de l'application d'une règle [2].

4.2.2 notion de graphe

Il existe deux types de graphe : les graphes orientés et les graphes non orientés [2].

4.2.2.1 Graphe non orienté

- Un graphe contient un ensemble de sommets reliés entre eux par des arêtes. Si deux sommets reliés par une arête, ils sont adjacents.
- Le nombre de sommets dans un graphe est appelé l'ordre du graphe.
- Le nombre d'arêtes dont le sommet est une extrémité est appelé le degré d'un sommet.

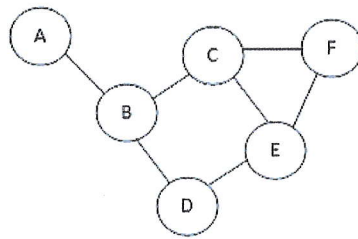
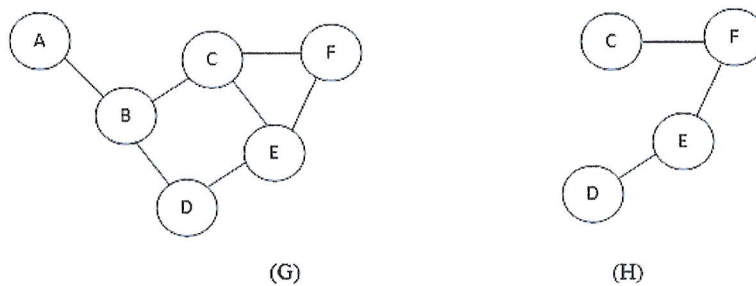


FIGURE 4.2 – Graphe non orienté.

- Un sous-graphe d'un graphe G est un graphe H composé de certains sommets de G , et toutes les arêtes qui relient ces sommets.

FIGURE 4.3 – Graphe G et Sous-graphe H .

4.2.2.2 Graphe orienté

Un graphe orienté est un graphe dont ses arêtes sont orientées alors on parle de l'origine et extrémité d'une arête. Dans ce type de graphe, une arête est appelée arc.

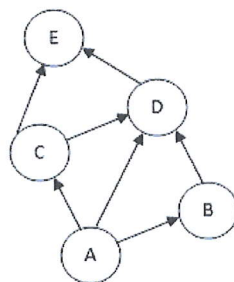


FIGURE 4.4 – Graphe orienté.

Un graphe étiqueté est un graphe orienté, dont les arcs sont associés d'étiquettes. Si toutes les étiquettes sont des nombres positifs, on parle de graphe pondéré.

Un graphe attribué est un graphe qui peut contenir un ensemble prédéfini d'attributs.

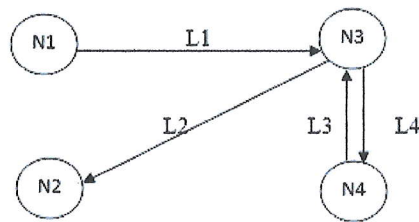


FIGURE 4.5 – Graphe orienté étiqueté.

4.2.3 Outils de transformation de graphes

Pour faire des transformations de modèles basées sur transformation de graphe de manière efficace il y a plusieurs outils, parmi ces outils on peut citer :

- ❖ **AGG** [33] : The Attributed Graph Grammar System.
- ❖ **FUJABA** [34] : From UML to Java and back again.
- ❖ **AToM³** [35] : A Tool for Multi-formalism and Meta-Modelling.
- ❖ **VIATRA** [36] : Visual Automated model TRAnsformations.
- ❖ **GreAT** [37] : The Graph Rewrite and Transformation tool suite [11].

Dans notre travail, nous avons opté pour l'outil AToM³ pour développer notre grammaire de graphe parce qu'il offre de nombreux avantages Parmi lesquels on peut citer : la disponibilité, la simplicité, c'est un outil de modélisation multi paradigme [31].

4.2.4 Présentation de l'outil AToM³

AToM³(A ToolForMulti-formalism and Meta-Modeling) est un outil visuel de modélisation et de méta-modélisation multi-formalismes, développé dans le laboratoire MSDL (Modeling Simulation and Design Lab) de l'institut d'informatique à l'université de McGill. Il est écrit en python et peut être exécuté sur toutes les plateformes où un interpréteur python est disponible (Linux, Windows et MacOS).

AToM³ est développé pour satisfaire deux fonctionnalités à savoir la méta-modélisation et la transformation de modèles [29].

1. Pour la méta-modélisation, AToM³ supporte la modélisation visuelle on utilise le formalisme Entité-Relation ou le formalisme de diagrammes de classes d'UML. Autrement dit pour méta-modéliser de nouveaux formalismes, on peut utiliser l'un des deux derniers formalismes. Dans notre approche on utilise le formalisme de diagramme de classe UML.
2. Pour la transformation de modèles, AToM³ supporte la réécriture de graphes qui utilise les règles de grammaire de graphes pour guider visuellement la procédure de transformation. Ces règles sont spécifiées par la syntaxe concrète des modèles [31].

AToM³ offre la possibilité à l'utilisateur de créer une nouvelle grammaire, de charger, de modifier et d'exécuter une grammaire. L'exécution de la grammaire sur un modèle en entrée produit un modèle en sortie [32].

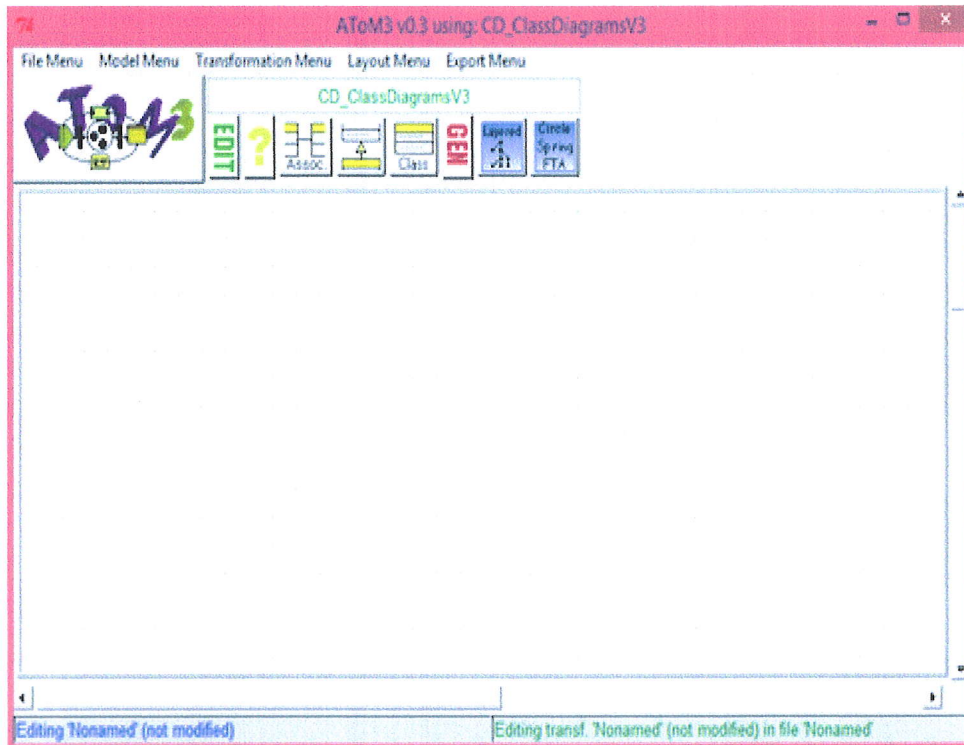


FIGURE 4.6 – L'interface de l'outil AToM³.

4.3 Transformation du diagramme d'activité UML vers le langage YAWL

Afin d'obtenir un diagramme YAWL à partir d'un diagramme d'activité UML, une méta-modélisation des formalismes de diagramme d'activité et celui de langage YAWL a été faite suivie de la création de 45 règles de grammaires, Pour ce faire, Nous avons utilisé l'outil de transformation de graphe AToM³.

4.3.1 Scénario de la transformation

Notre approche de transformation est appuyée sur les étapes suivantes :

1. Construction de méta-modèle des diagrammes d'activité.
2. Construction de méta-modèle des modèles YAWL.
3. Appliquer la grammaire de graphe (dgrmactv2yawl) sur le diagramme d'activité.

1- La classe "Action"

Cette classe représente une Action de diagramme d'activité, Elle est représentée graphiquement par un ovale. Elle possède les attributs suivants : NAME, de type String pour donner un nom à l'action ; SWIM_NAME, de type string pour donner un nom à la partition qui appartient l'action. Et elle possède le contrainte swim valid pour vérifier l'existence du Swimlane. La classe action peut être reliée par l'une des associations suivantes : Appartient (avec la classe Swimlane), AC2AC (avec la classe action), AC2ND (avec les classes Decision, Fork, Merge, Join), AC2FN (avec la classe Final_Noeud).

2- La classe "Initial_Noeud"

La classe Initiale_Noeud représente le début d'un diagramme d'activité. Graphiquement, elle est représentée par un petit cercle plein. Elle possède les deux attributs : NAME, de type String pour donner le nom ; SWIM_NAME, de type string pour donner un nom à la partition qui appartient, aussi une contrainte swimvalid pour vérifier l'existence du Swimlane.

Cette classe peut être reliée par l'une des associations suivantes : Appartient (avec la classe Swimlane), AC2ND (avec les classes Decision, Fork, Merge, Join), ND2AC (avec la classe Action).

3- La classe "Fork"

Cette classe représente un noeud de séparation de flux d'entrée en plusieurs flux en sortie concurrents. Elle est représentée graphiquement par un trait plein. La classe Fork possède les attributs : NAME, de type String pour donner le nom ; SWIM_NAME, de type string pour donner un nom à la partition qui appartient le fork. Et aussi elle possède deux contraintes : swim pour vérifier l'existence du Swimlane et input pour limiter le nombre des arcs entrants à 1.

La classe Fork peut être reliée par l'une des associations suivantes : Appartient (avec la classe Swimlane), ND2AC (avec la classe Action), ND2FN (avec la classe Final_Noeud) et ND2ND (avec les classes Fork, Join, Merge, Decision).

4- La classe "Merge"

La classe Merge pour accepter un seul flux en sortie parmi plusieurs flux en entrée, elle est représentée graphiquement par un losange, elle possède les attributs : NAME, de type String pour donner le nom ; SWIM_NAME, de type string pour donner un nom à la partition qui appartient le Merge. Et aussi elle possède deux contraintes : swimvalid pour vérifier l'existence du Swimlane et output pour limiter le nombre des arcs sortants à 1.

Cette classe peut être reliée par l'une des associations suivantes : Appartient (avec la classe Swimlane), ND2AC (avec la classe Action), ND2FN (avec la classe Final_Noeud) et ND2ND (avec les classes Fork, Join, Merge, Decision).

5- La classe "Decision"

Cette classe permet de faire un choix entre plusieurs flux sortants en fonction de la condition de garde qui est associée à chaque arc sortant. Représentée graphiquement par un losange, elle possède les attributs : NAME, de type String pour donner le nom ; SWIM_NAME, de type string pour donner un nom à la partition qui y appartient Decision. Et aussi elle possède deux contraintes : swimvalid pour vérifier l'existence du Swimlane et input pour limiter le nombre des arcs entrants à 1.

La classe Decision peut être reliée par l'une des associations suivantes : Appartient (avec la classe Swimlane), ND2AC (avec la classe Action), ND2FN (avec la classe Final_Noeud) et ND2ND (avec les classes Fork, Join, Merge, Decision).

6- La classe "Join"

Cette classe représente un noeud de synchronisation qui ne peut être franchit que lorsque toute les arcs entrants sont activés. Elle est reliée par plusieurs arcs en entrée et une seule arc en sortie. Représentée graphiquement par un trait plein, elle possède les attributs : NAME, de type String pour donner le nom ; SWIM_NAME, de type string pour donner un nom à la partition ou appartient Join. Et aussi elle possède les contraintes suivants : swim pour vérifier l'existence du Swimlane et output pour limiter le nombre des arcs sortants à 1.

La classe Join peut être reliée par l'une des associations suivantes : Appartient (avec la classe Swimlane), ND2AC (avec la classe Action), ND2FN (avec la classe Final_Noeud) et ND2ND (avec les classes Fork, Join, Merge, Decision).

7- La classe "Swimlane"

La classe Swimlane représente les partitions dans le diagramme d'activité, graphiquement, Elle représentée par un rectangle. Elle possède un attribut NAME de type string pour donner un nom à la partition.

8- La classe "Final_Noeud"

La classe Final_Noeud représente la fin de flux de contrôle d'un diagramme d'activité. Graphiquement, elle est représentée par un petit cercle plein. Elle possède les deux attributs : NAME de type String pour donner le nom ; SWIM_NAME de type string pour donner un nom à la partition qui appartient et type de type Enum pour choisir le type de noeud final, Et aussi elle possède les contraintes suivants : swim valid pour vérifier l'existence du Swimlane et choix Type pour permet de changer l'apparence graphique entre un noeud de flux final et un noeud d'activité final. Le noeud d'activité final est représenté graphiquement par un cercle vide contenant un petit cercle plein, et l'autre représenté par un cercle vide contenant une croix.

4.3.3 Méta-modèle de langage YAWL

Ce méta-modèle est composé de 10 classes et 8 associations développé par le méta-formalisme CD.classDiagramsV3, dans le but d'avoir un outil intégrant AToM³ qui offre les outils nécessaires pour modéliser les modèles YAWL (voir la figure 4.8).

de type String pour donner un nom à la tâche.

La classe Task_atomic peut être reliée par l'association task2task avec lui-même, par tasjo2output avec la classe output-condition, par task2spjo avec les classes AND-join, XOR-join, OR-join, AND-split, XOR-split, OR-split.

2. La classe "Task_composite"

Cette classe représente une tâche composite du langage YAWL, elle possède l'attribut : name, de type String pour donner un nom à la tâche.

La classe Task_composite peut être reliée par l'association taskcomp2taskcomp avec lui-même, par tasjo2output avec la classe output_condition, par task2spjo avec les classes AND-join, XOR-join, OR-join, AND-split, XOR-split, OR-split.

3. La classe "Input_condition"

La classe Input_condition représente le point d'entrée du langage YAWL.

La classe Input_condition peut être reliée par l'association input2spltas avec les classes task_atomic, task_composite, AND-split, XOR-split, OR-split.

4. La classe "Output_condition"

La classe output-condition représente le point d'arrivée du langage YAWL.

5. La classe "AND-split"

La classe AND-split représente un noeud qui divise le flux de contrôle entrant en plusieurs flux sortants exécutés simultanément, elle possède un seul attribut : name, de type String pour indiquer le nom. Ainsi, elle possède une contrainte : input, pour spécifier le nombre de flux entrants à 1.

La classe AND-split peut être reliée par l'association spjo2task avec les deux classes Task_atomic et Task_composite, par split2spjo avec les classes AND-join, XOR-join, OR-join, AND-split, XOR-split, OR-split.

6. La classe "XOR-split"

La classe XOR-split représente un noeud pour déclencher un seul flux sortant, elle possède l'attribut : name, de type String pour indiquer le nom. Ainsi, elle possède une contrainte : input, pour spécifier le nombre de flux entrants à 1.

La classe XOR-split peut être reliée par l'association spjo2task avec les deux classes Task_atomic et Task_composite, par split2spjo avec les classes AND-join, XOR-join, OR-join, AND-split, XOR-split, OR-split.

7. La classe "OR-split"

Cette classe représente un noeud pour déclencher certains flux, mais pas nécessairement tous les flux sortants vers d'autres tâches, elle possède l'attribut : name, de type String pour

indiquer le nom. Ainsi, elle possède une contrainte : input, pour spécifier le nombre de flux entrants à 1.

La classe OR- split peut être reliée par l'association spjo2task avec les deux classes Task_atomic et Task_composite, par split2spjo avec les classes AND-join, XOR-join, OR-join, AND-split, XOR- split, OR- split.

8. La classe "AND- join"

Cette classe représente un noeud qu'attendre de recevoir tous les flux entrants avant de s'exécuter. Elle possède l'attribut : name, de type String pour indiquer le nom. Ainsi, elle possède une contrainte : Output, qui Permet de spécifier le nombre de flux sortants à 1.

La classe AND- join peut être reliée par l'association spjo2task avec les deux classes Task_atomic et Task_composite, par join2spjo avec les classes AND-join, XOR-join, OR-join, AND-split, XOR- split, OR- split, par tasjo2output avec la classe output-condition.

9. La classe "XOR- join"

La classe XOR-join représente un noeud capable de s'exécuter le travail, une fois terminé sur un flux entrant, elle possède l'attribut : name, de type String pour indiquer le nom. Ainsi, elle possède une contrainte : Output, pour spécifier le nombre de flux sortants à 1.

La classe XOR- join peut être reliée par l'association spjo2task avec les deux classes Task_atomic et Task_composite, par join2spjo avec les classes AND-join, XOR-join, OR-join, AND-split, XOR- split, OR- split, par tasjo2output avec la classe output-condition.

10. La classe "OR- join"

La classe OR-join représente un noeud qui permet de garantir qu'une tâche attend jusqu'à ce que tous les flux entrants soient terminés ou ne se terminent jamais. Elle possède l'attribut : name, de type String pour indiquer le nom. Ainsi, elle possède une contrainte : Output, pour spécifier le nombre de flux sortants à 1.

La classe OR- join peut être reliée par l'association spjo2task avec les deux classes Task_atomic et Task_composite, par join2spjo avec les classes AND-join, XOR-join, OR-join, AND-split, XOR- split, OR- split, par tasjo2output avec la classe output-condition.

4.3.4 Génération de l'environnement

Après la modélisation de deux méta-modèles, nous procédons à la génération des environnements des deux méta-modèles. Chaque méta modèle généré comporte l'ensemble des classes modélisées sous forme de boutons qui sont prêts à être utiliser pour une modélisation d'un diagramme d'activité ou d'un modèle YAWL.

La figure 4.9 montre l'environnement généré pour les diagrammes d'activité.

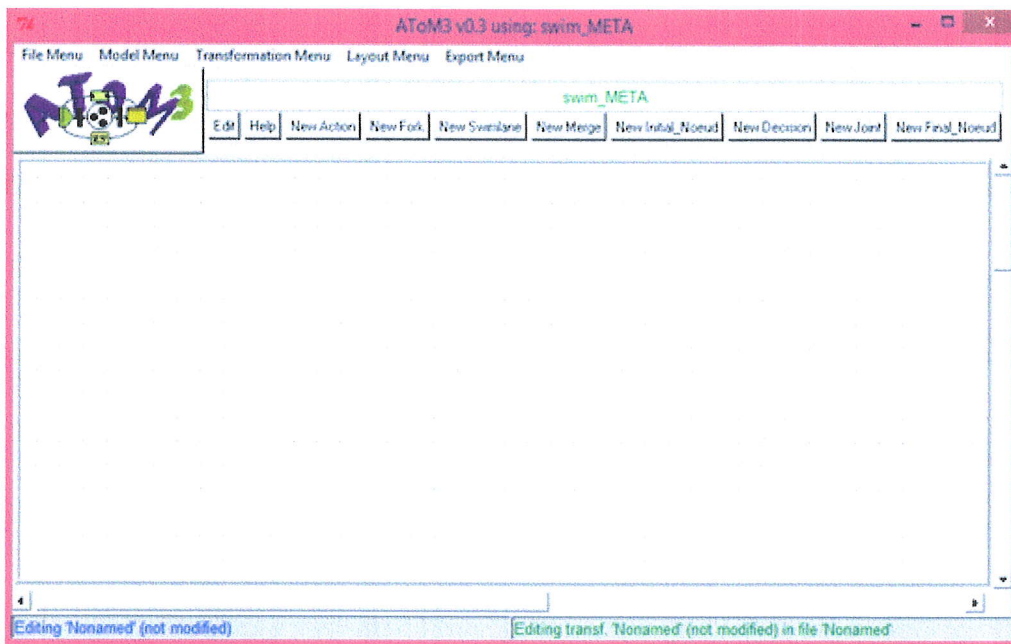


FIGURE 4.9 – Environnement de diagramme d'activité sous ATOM³.

La figure 4.10 montre l'environnement généré pour les modèles YAWL.



FIGURE 4.10 – Environnement de modèle YAWL sous ATOM³.

4.3.5 La grammaire de graphe proposée

Dans la transformation de graphe sur l'AToM³ le processus le plus important est la construction des règles de grammaire pour faire cela, il faut comprendre les deux modèles diagramme d'activité et YAWL.

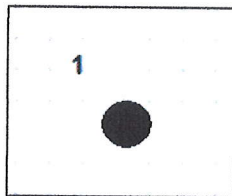
❖ Ensemble des règles :

Notre grammaire de graphes (`dgrmactv2yawl`) est composée de 45 règles. Chaque règle est caractérisée par un nom et une priorité.

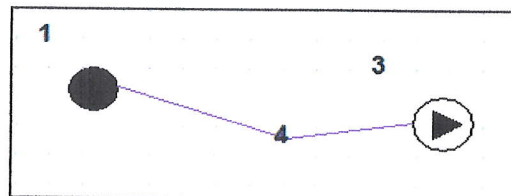
☛ Règle 1 : IN2IC (priorité : 1)

Description : un état initial dans un diagramme d'activité est transformé vers une condition d'entrée dans YAWL.

LHS :



RHS :



Condition :

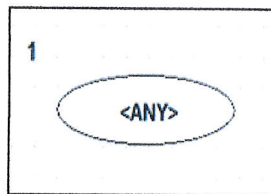
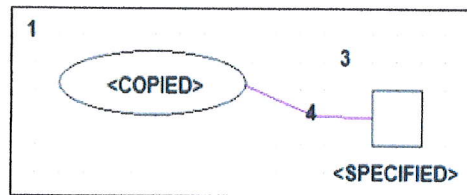
```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not hasattr(node, "IN2IC_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node.IN2IC_executed = True
```

☛ Règle 2 : AC2AT (priorité : 2)

Description : une action dans un diagramme d'activité est transformée vers une tâche atomique dans YAWL, le nom de l'action devient le nom de tâche atomique.

LHS :**RHS :****Condition :**

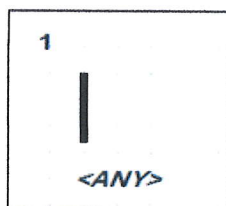
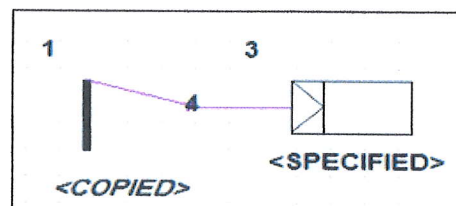
```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not hasattr(node, "AC2AT_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node.AC2AT_executed = True
```

☛ Règle 3 : JN2AJ (priorité : 3)

Description : un noeud d'union (join) dans un diagramme d'activité est transformé vers un noeud AND-join dans YAWL.

LHS :**RHS :****Condition :**

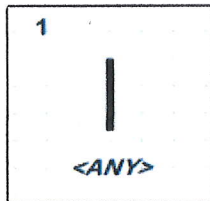
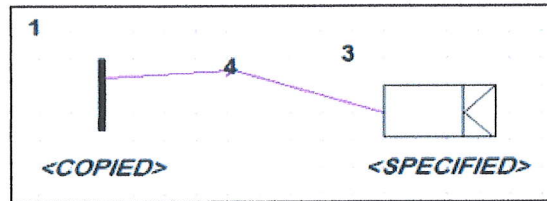
```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not hasattr(node, "JN2AJ_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node.JN2AJ_executed = True
```

☛ Règle 4 : FR2AS (priorité : 4)

Description : un noeud de bifurcation (fork) dans un diagramme d'activité est transformé vers un noeud AND-split dans YAWL.

LHS :**RHS :****Condition :**

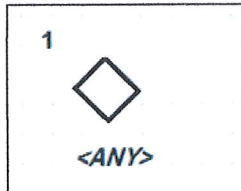
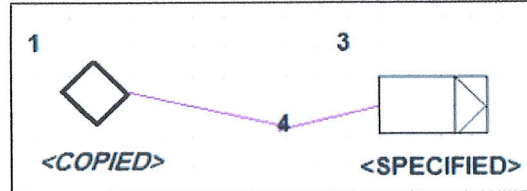
```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not hasattr(node, "FR2AS_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node.FR2AS_executed = True
```

☛ Règle 5 : DC2XS (priorité : 6)

Description : un noeud de décision dans un diagramme deactivité est transformé vers un noeud XOR-split dans YAWL.

LHS :**RHS :****Condition :**

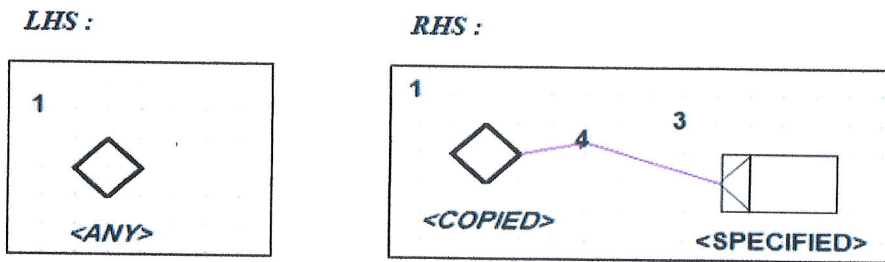
```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not hasattr(node, "DC2XS_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node.DC2XS_executed = True
```

☛ Règle 6 : MR2XJ (priorité : 8)

Description : un noeud de fusion (merge) dans un diagramme d'activité est transformé vers un noeud XOR-join dans YAWL.



Condition :

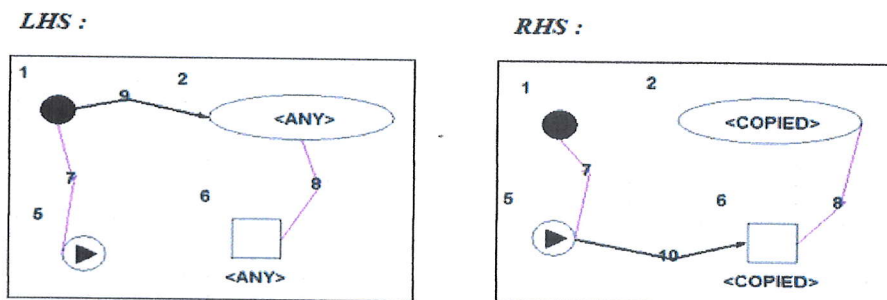
```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not hasattr(node, "MR2XJ_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node.MR2XJ_executed = True
```

☛ Règle 7 : IA2IT (priorité : 9)

Description : un état initial relié par une action est transformée vers une condition d'entrée reliée par une tâche atomique dans YAWL. Il existe des règles similaires à cette règle à savoir : l'état initial relié par fork (IF2CA) et l'état initial relié par une décision : (ID2CX).



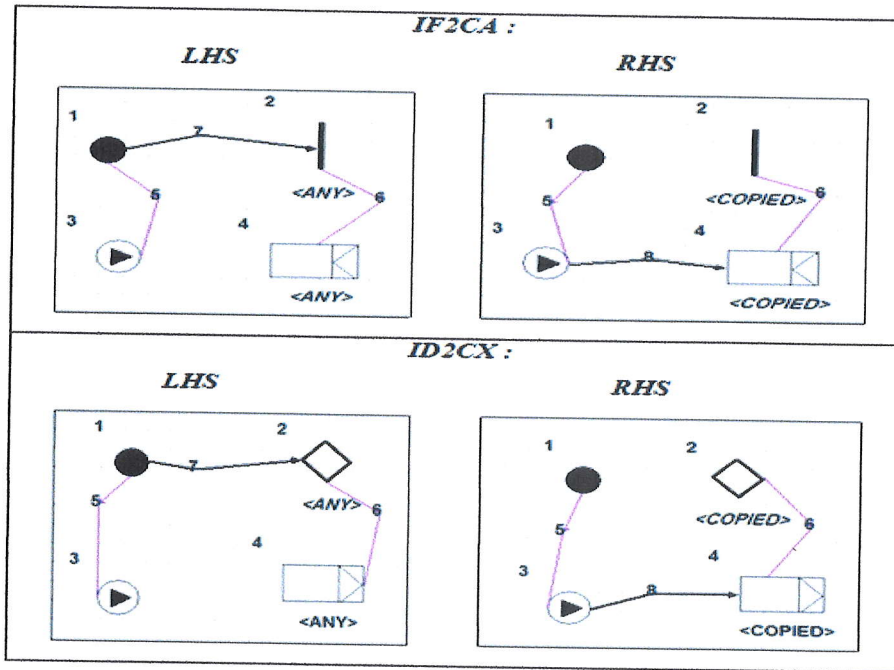
Condition :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not hasattr(node, "IA2IT_executed")
```

Action :

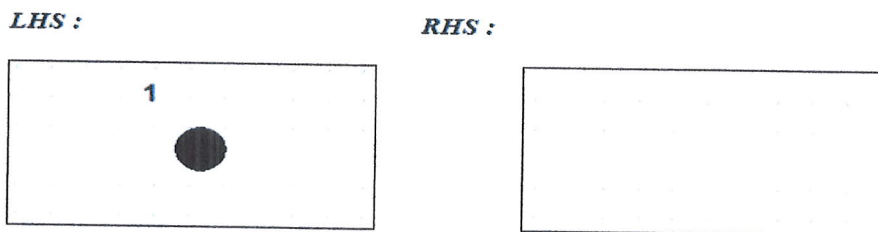
```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node.IA2IT_executed = True
```

Règles similaire :



➤ Règle 8 : deleteIA (priorité : 10)

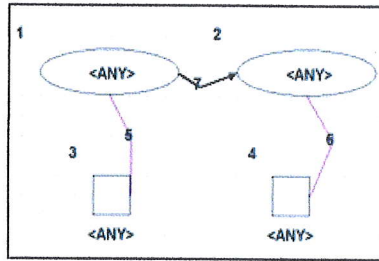
Description : pour la suppression d'un état initial dans un diagramme d'activité on va laisser la partie RHS vide.



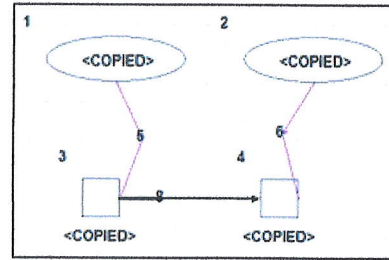
➤ Règle 9 : AA2TT (priorité : 11)

Description : une action reliée par une autre action est transformée vers une tâche atomique reliée par une autre tâche atomique dans YAWL. Les autres règles similaires à cette règle sont : une action reliée par une décision (AD2TX), une action reliée par un fork (AF2TA), une action reliée par un merge (AM2TX) et une action reliée par un join (AJ2TA).

LHS :



RSH :



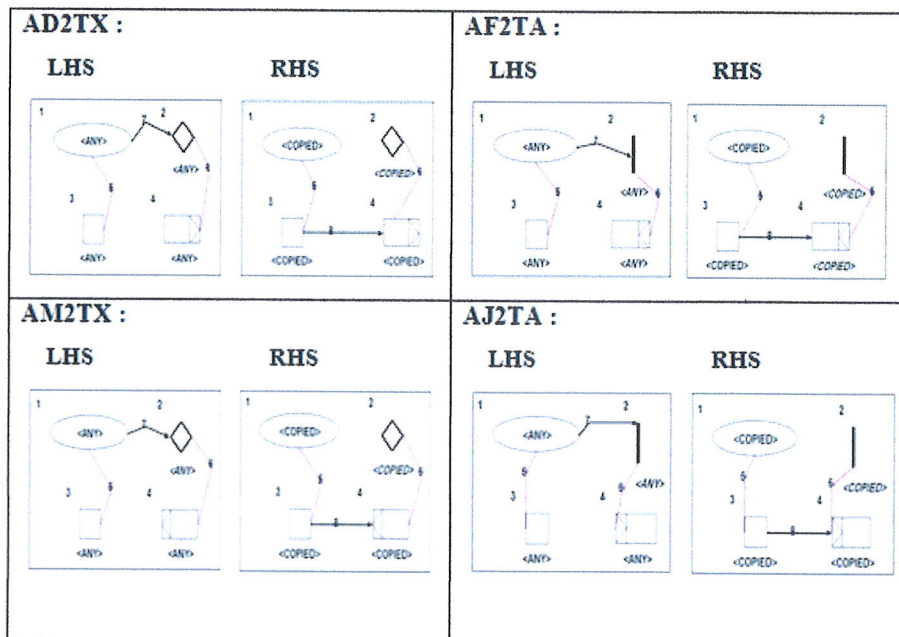
Condition :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(7))
return not hasattr(node, "AA2TT_executed")
```

Action :

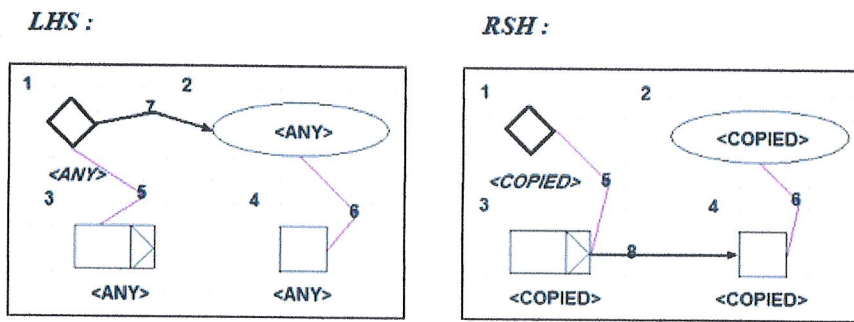
```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(7))
node.AA2TT_executed = True
```

Règles similaire :



☛ Règle 10 : DA2XT (priorité : 16)

Description : une décision reliée par une action est transformée vers un XOR-split reliée par une tâche atomique dans YAWL. Les autres règles similaires à cette règle sont : un merge relié par une action (MA2TX), un fork relié par une action (FA2AT) et un join relié par une action (JA2AT).



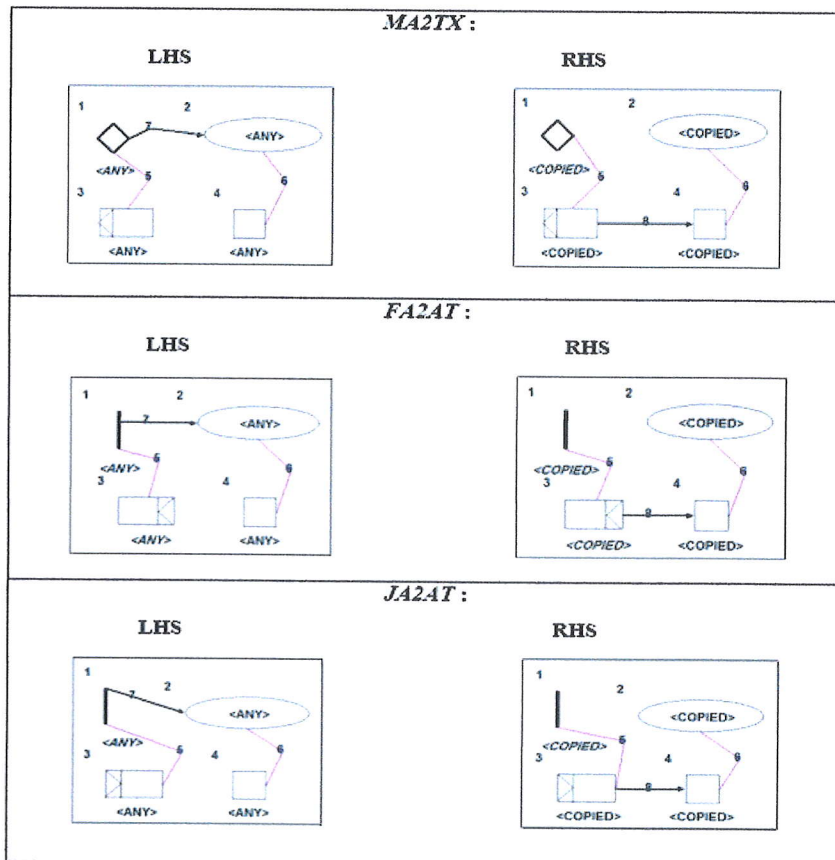
Condition :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(7))
return not hasattr(node, "DA2XT_executed")
```

Action :

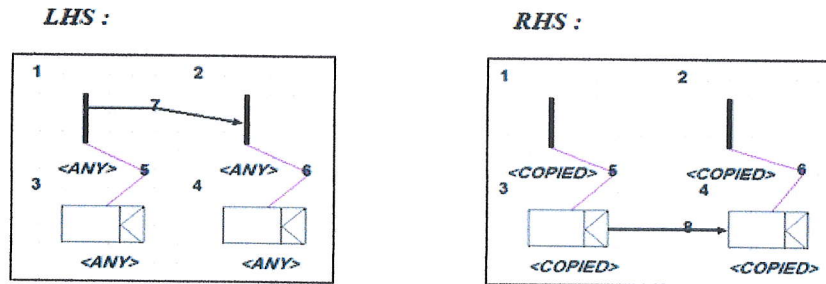
```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(7))
node.DA2XT_executed = True
```

Règles similaire :



➡ Règle 11 : FF2AA (priorité : 21)

Description : un fork relié à un fork dans un diagramme d'activité est transformée vers un noeud AND-split reliée par un AND-split dans YAWL. Les autres règles similaires à cette règle sont : un fork relié par une décision (FD2AX), un fork relié par un join (FJ2AA), un fork relié par un merge (FM2AX).



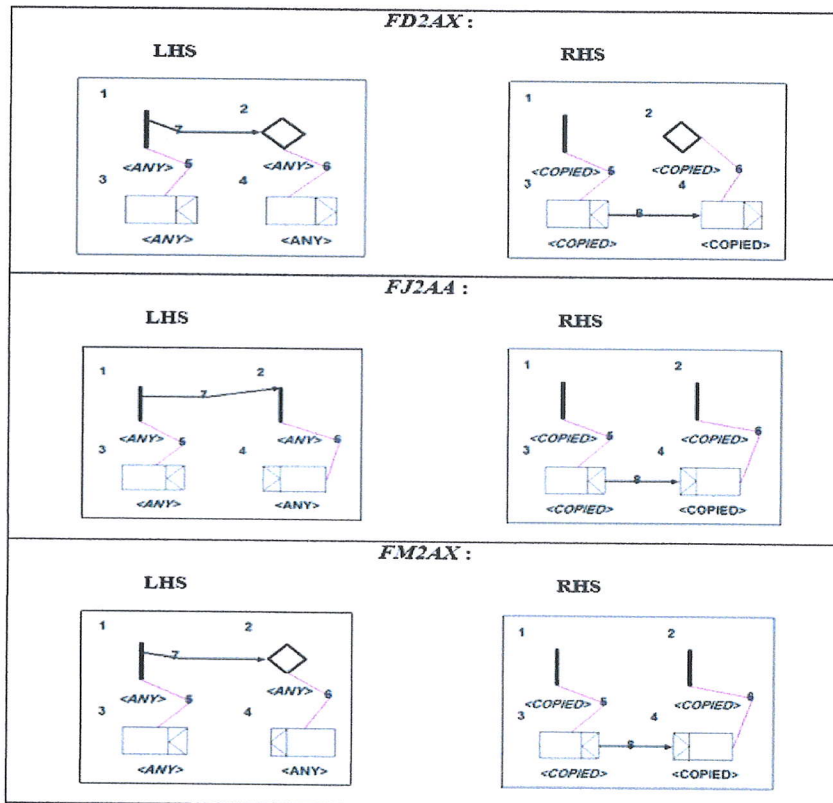
Condition :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
return not hasattr(node, "FF2AA_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
node.FF2AA_executed = True
```

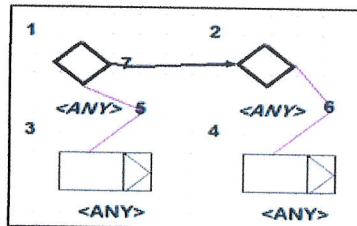
Règles similaire :



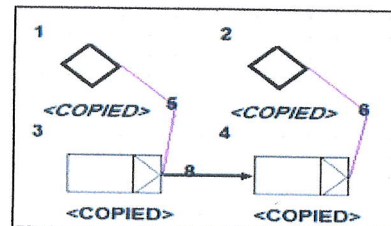
☛ Règle 12 : DD2XX (priorité : 20)

Description : une décision reliée par une décision dans un diagramme d'activité est transformée vers un XOR-split reliée à un XOR-split dans YAWL. Les autres règles similaires à cette règle sont : une décision relié par un fork (DF2XA), une décision reliée par un join (DJ2XA), une décision reliée par un merge (DM2XX).

LHS :



RHS :



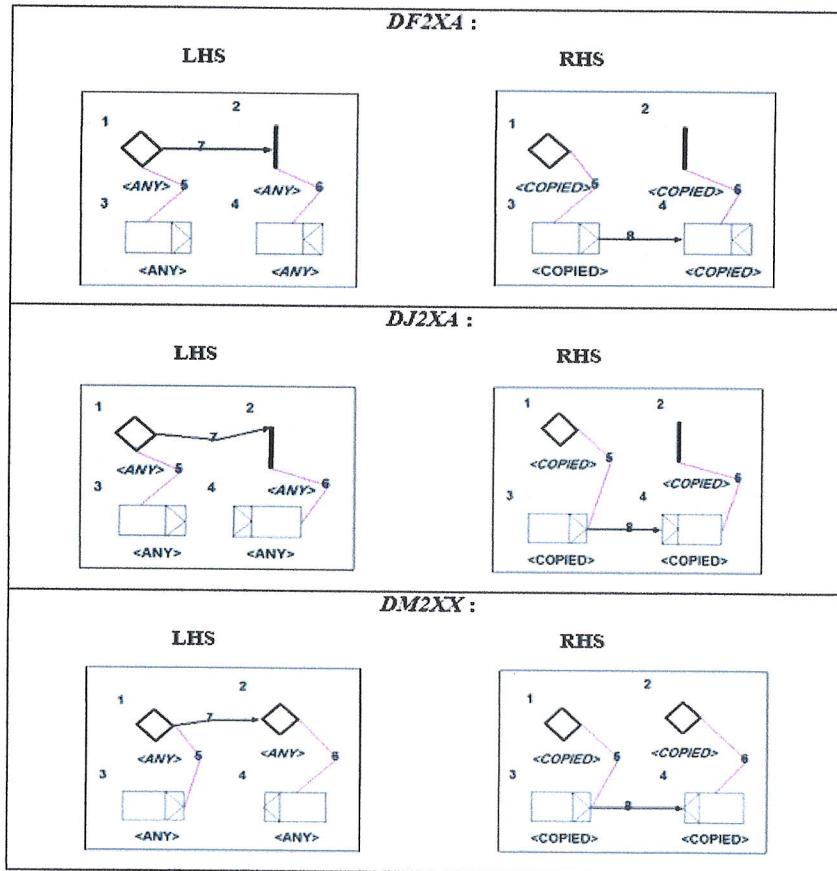
Condition :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
return not hasattr(node, "DD2XX_executed")
```

Action :

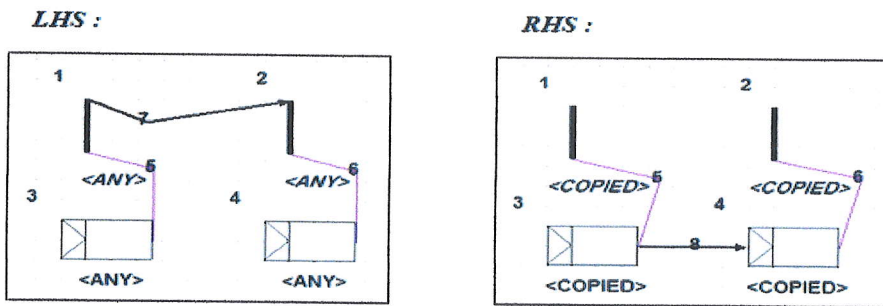
```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
node.DD2XX_executed = True
```

Règles similaire :



☛ Règle 13 : JJ2AA (priorité : 30)

Description : un join relié par un join dans un diagramme d'activité est transformée vers un AND-join relié à un AND-join dans le modèle YAWL. Les autres règles similaires à cette règle sont : un join relié par un fork (JF2AA), un join est par un join (JD2AX), un join relié par un merge (JM2AX).



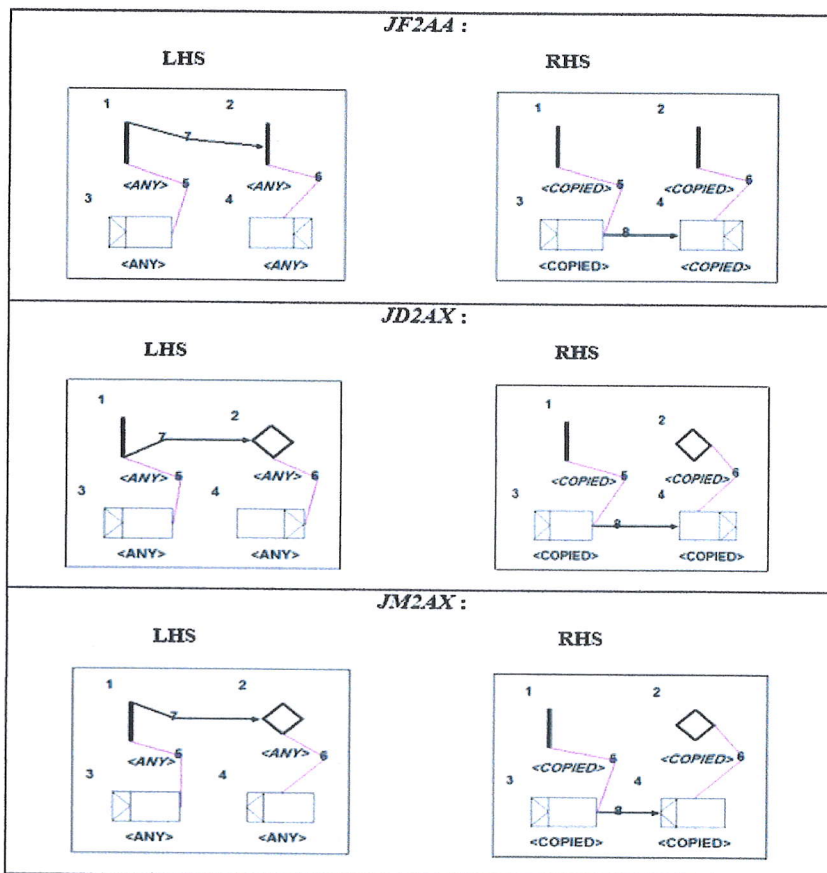
Condition :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not hasattr(node, "JJ2AA_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node.JJ2AA_executed = True
```

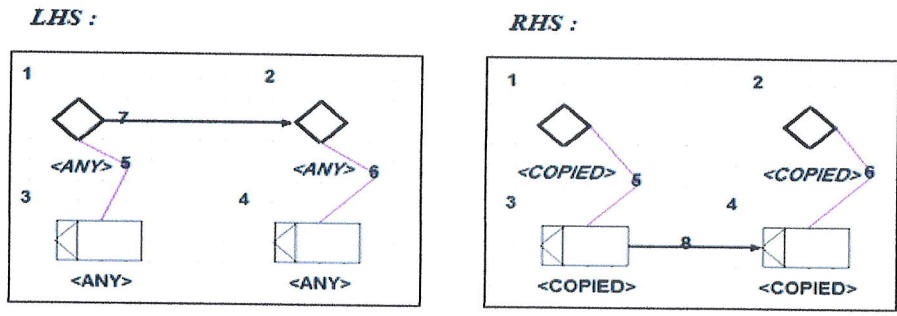
Règles similaire :



➤ Règle 14 : MM2XX (priorité : 35)

Description : un merge relié par un merge dans un diagramme d'activité est transformée vers un XOR-join relié à un XOR-join dans YAWL. Les autres règles similaires à cette règle

sont : un merge relié par un fork (MF2XA), un merge relié par un join (MJ2XA), un merge relié par une décision (MD2XX).



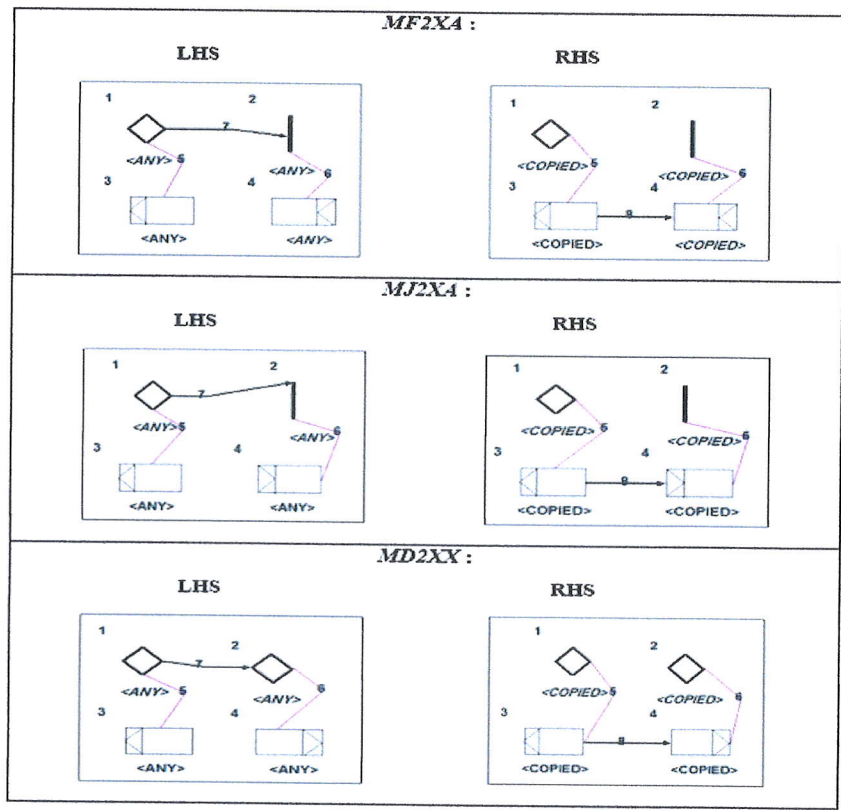
Condition :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not hasattr(node, "MM2XX_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node.MM2XX_executed = True
```

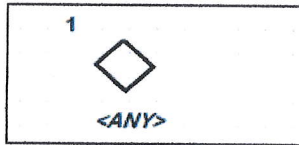
Règles similaire :



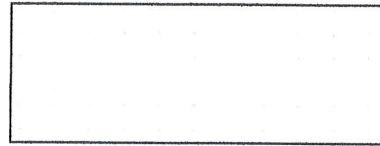
➤ Règle 15 : deleteDC (priorité : 36)

Description : pour la suppression d'une décision dans un diagramme d'activité on va laisser la partie RHS vide.

LHS :



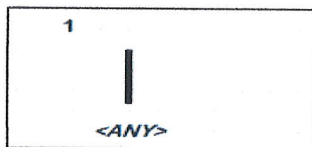
RHS :



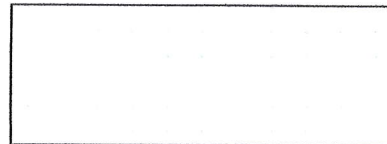
☛ Règle 16 : deleteFR (priorité : 37)

Description : pour la suppression d'un noeud de bifurcation dans un diagramme d'activité on va laisser la partie RHS vide.

LHS :



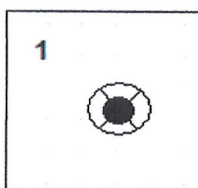
RHS :



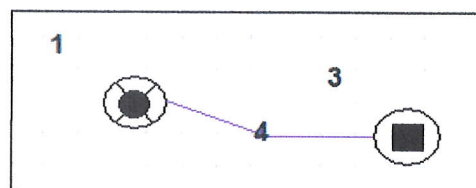
☛ Règle 17 : EF2OC (priorité : 38)

Description : un noeud d'activité final ou un noeud de flux final dans un diagramme d'activité est transformé vers une condition de sortie dans YAWL.

LHS :



RHS :



Condition :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not hasattr(node, "EF2OC_executed")
```

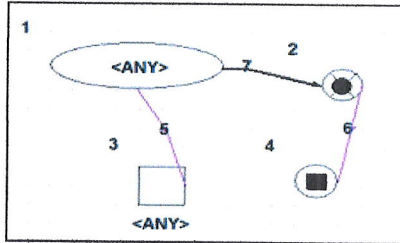
Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node.EF2OC_executed = True
```

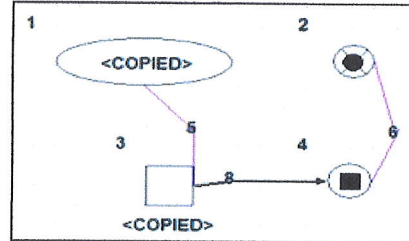
☛ Règle 18 : AF2TO (priorité : 39)

Description : une action reliée par un noeud final est transformée vers une tâche atomique reliée à une condition de sortie dans YAWL. D'autres règles similaires à cette règle sont : un join relié par un noeud final (JF2AO) et un merge relié par un noeud final (MF2XO).

LHS :



RHS :



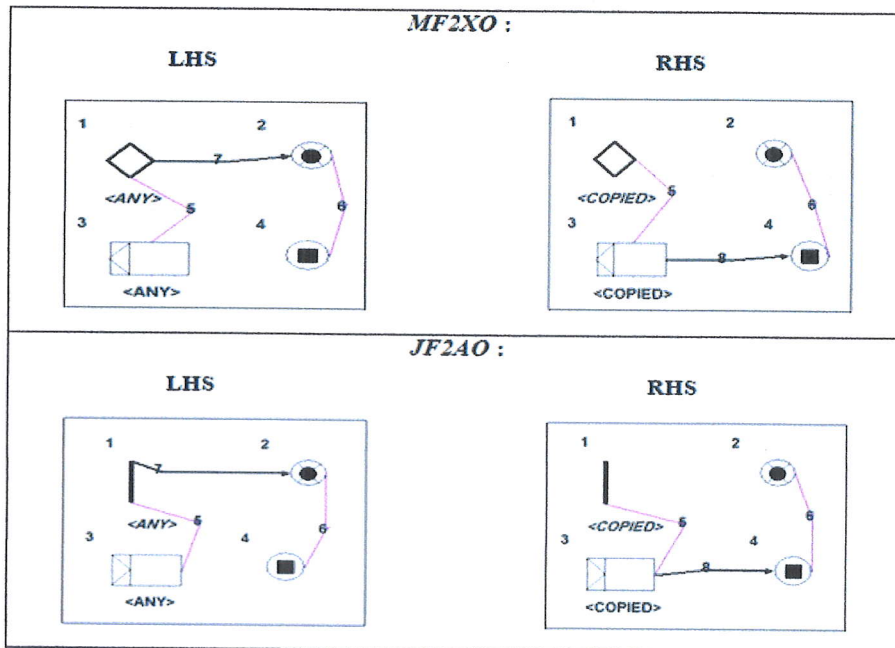
Condition :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not hasattr(node, "AF2TO_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node.AF2TO_executed = True
```

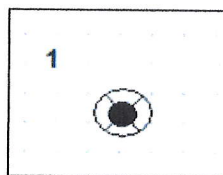
Règles similaire :



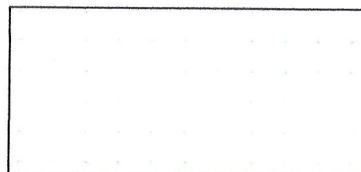
➤ Règle 19 : deleteFE (priorité : 42)

Description : pour la suppression d'un noeud final dans un diagramme d'activité on laisse la partie RHS vide.

LHS :



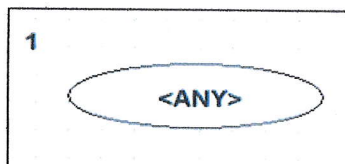
RHS :



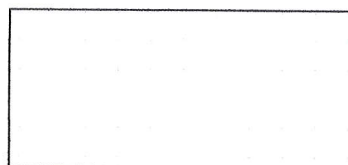
☛ Règle 20 : deleteAC (priorité : 43)

Description : pour la suppression d'une action dans un diagramme d'activité on laisse la partie RHS vide.

LHS :



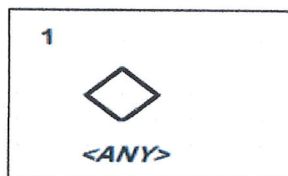
RHS :



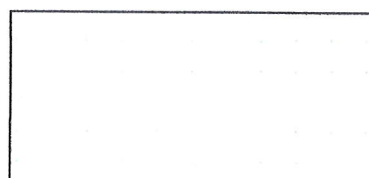
☛ Règle 21 : deleteMR (priorité : 44)

Description : pour la suppression d'un merge dans un diagramme d'activité on laisse la partie RHS vide.

LHS :



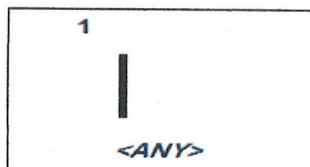
RHS :



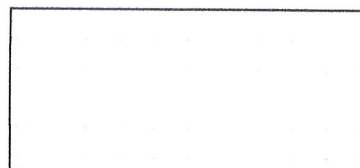
☛ Règle 22 : deleteJN (priorité : 45)

Description : pour la suppression d'un noeud d'union (join) dans un diagramme d'activité on laisse la partie RHS vide.

LHS :



RHS :



4.3.6 Exemple de transformation d'un diagramme d'activité vers le YAWL

Afin d'illustrer l'approche de transformation définie précédemment, nous allons l'appliquer sur un exemple de diagramme d'activité présenté dans le premier chapitre.

Tout d'abord, nous devons utiliser notre outil intégré AToM³ pour décrire le diagramme d'activité, la figure ci-dessous montre sa spécification.

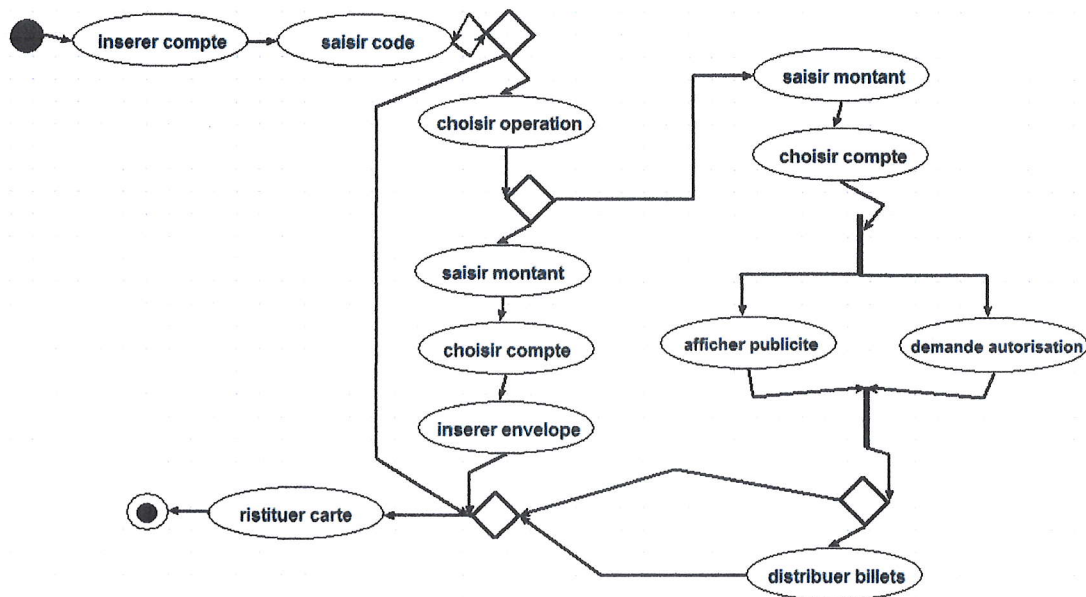


FIGURE 4.11 – Exemple de diagramme d'activité sous AToM³.

Entamant l'exécution de notre grammaire de graphe, on obtient le graphe intermédiaire ci-dessous, en fait c'est un mélange entre les deux graphes : diagramme d'activité et modèle YAWL :

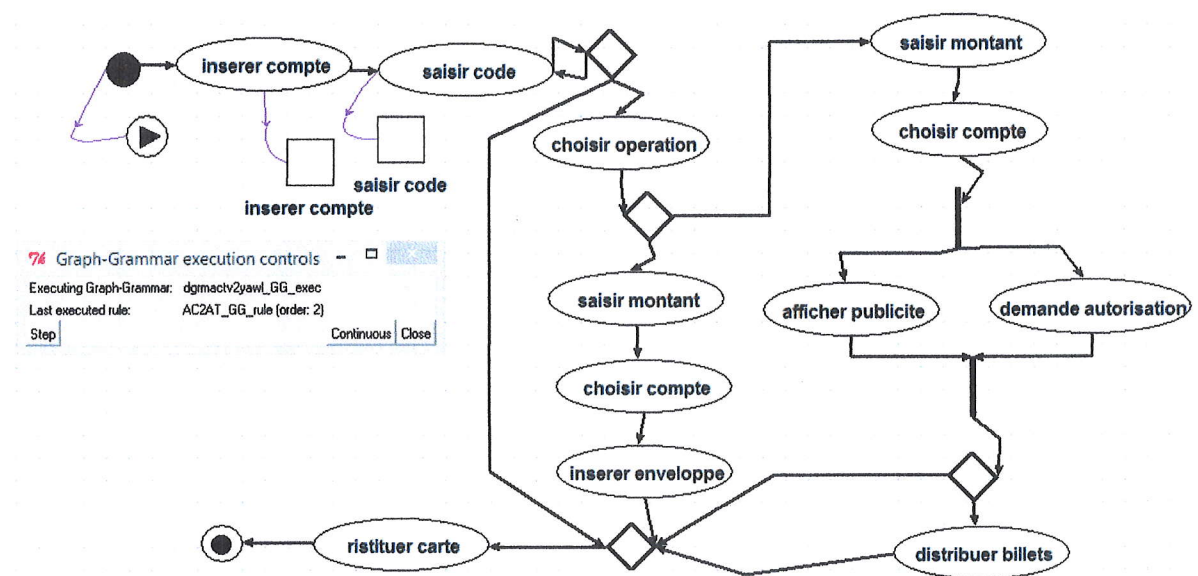


FIGURE 4.12 – Graphe intermédiaire de l'exemple.

Après l'exécution de la grammaire de graphe, on obtient le modèle YAWL représenté dans la figure ci-dessous :

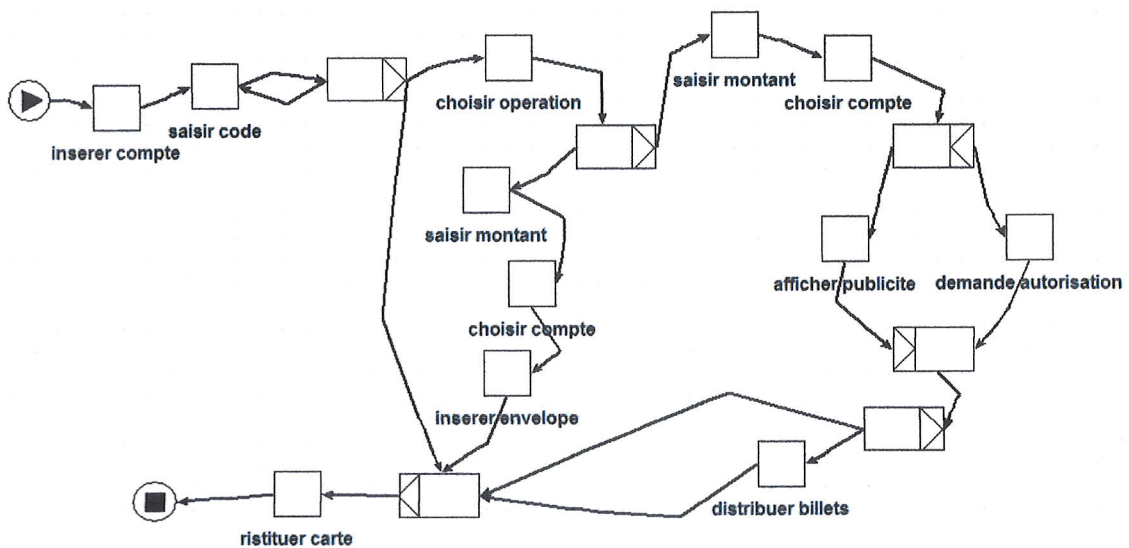


FIGURE 4.13 – Le modèle YAWL.

4.4 La transformation des modèles YAWL vers une description textuelle

4.4.1 la grammaire de graphe proposée

Afin de produire des spécifications textuelles des modèles YAWL, nous avons proposé une grammaire de graphes (yawl2xml.GG_exec) avec une action initiale, 12 règles, et une action finale. L'application de cette grammaire à un modèle YAWL conduit à la génération d'un fichier qui contient sa description XML. Un schéma XML est fourni dans [39] pour montrer comment passer du graphique YAWL vers sa représentation XML « .yawl ». Les règles sont similaires et on va présenter seulement quelques règles notamment celles liées à l'exemple précédent.

- ✓ **L'action initial** : Dans l'action initiale de la grammaire de graphe nous avons créé un fichier nommé (fichyawl.yawl) à accès séquentiel (c.-à-d. les données sont enregistrées dans le fichier les unes à la suite des autres) pour stocker le code de fichier générée. Sous Python, l'accès aux fichiers est assuré par l'intermédiaire d'un objet-fichier appelé « obFichier » dans notre cas, que l'on crée à l'aide de la fonction interne "open()". Après avoir appelé cette fonction, nous pouvons écrire dans le fichier en utilisant les méthodes spécifiques de cet objet-fichier telque "write()" puis le refermer "close()".

```

obFichier = open('fichyawl.yawl','w')
obFichier.write('\n')
obFichier.write('<?xml version="1.0" encoding="UTF-8"?>')
obFichier.write('\n')
obFichier.write('<specificationSet
xmlns="http://www.yawlfoundation.org/yawlschema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="4.0"
xsi:schemaLocation="http://www.yawlfoundation.org/yawlschema
http://www.yawlfoundation.org/yawlschema/YAWL_Schema4.0.xsd">')
obFichier.write('\n')
obFichier.write('<specification uri="New_Specification">')
obFichier.write('\n')
obFichier.write('<documentation>No description provided</documentation>')
obFichier.write('\n')
obFichier.write('<metaData>')

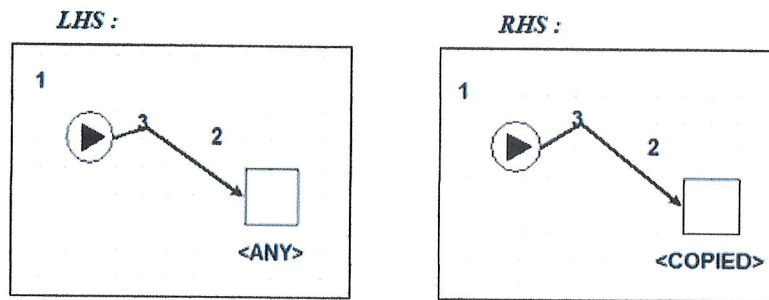
obFichier.write('<metaData>')
obFichier.write('\n')
obFichier.write('<creator>Chamou</creator>')
obFichier.write('\n')
obFichier.write('<description>No description provided</description>')
obFichier.write('\n')
obFichier.write('<coverage>4.2.744</coverage>')
obFichier.write('\n')
obFichier.write('<version>0.6</version>')
obFichier.write('\n')
obFichier.write('<persistent>>false</persistent>')
obFichier.write('\n')
obFichier.write('<identifier>UID_2717ae96-bf87-4bf5-a9d5-a2ea56b9a7a9</identifie
r>')
obFichier.write('\n')

obFichier.write('\n')
obFichier.write('</metaData>')
obFichier.write('\n')
obFichier.write('<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" />')
obFichier.write('\n')
obFichier.write('<decomposition id="Net" isRootNet="true"
xsi:type="NetFactsType">')
obFichier.write('\n')
obFichier.write('<processControlElements>')
obFichier.write('\n')
obFichier.close()

```

FIGURE 4.14 – l'action initiale sous ATOM³.

- ✓ Ensemble des règles : Notre grammaire graphes est composée de 12 règles. Chaque règle est caractérisée par un nom et une propriété.
- ☛ Règle 1 : ICT2ICT (priorité : 1)



Condition :

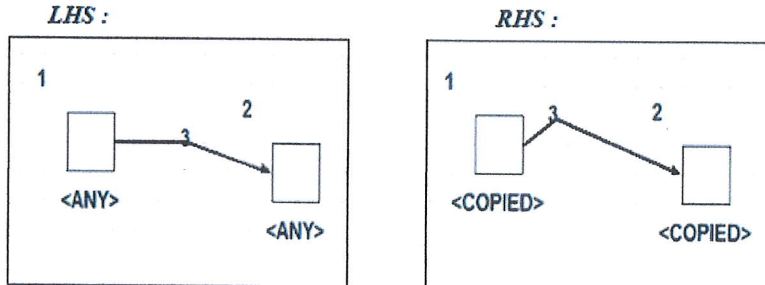
```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
return not hasattr(node, "ICT2ICT_executed")
```

Action :

```
node1 = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
taskname = node.name.getValue()
node.ICT2ICT_executed = True

obFichier = open('fichyawl.yawl','a')
obFichier.write('<inputCondition id="InputCondition"> \n <flowsInto> \n
<nextElementRef id= "' +taskname+' "/> \n </flowsInto> \n </inputCondition>')
```

➤ Règle 2 : TA2TA (priorité : 2)



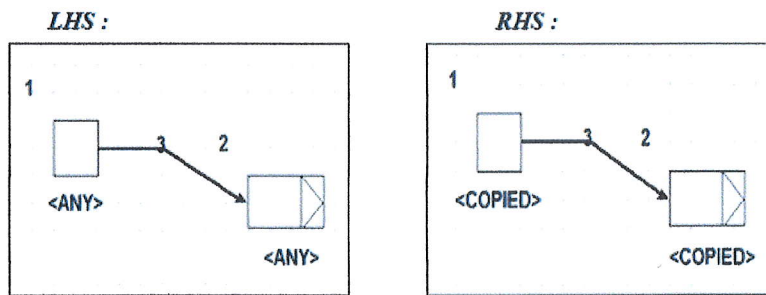
Condition :

```
node2 = self.getMatched(graphID, self.LHS.nodeWithLabel(3))
return not hasattr(node2, "TA2TA_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
taskname = node.name.getValue()
node1 = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
taskname1 = node1.name.getValue()
node2 = self.getMatched(graphID, self.LHS.nodeWithLabel(3))
node2.TA2TA_executed = True
obFichier = open('fichyawl.yawl','a')
obFichier.write('<task id="'+taskname+'"> \n <name> '+taskname+' </name>\n
<flowsInto> \n <nextElementRef id= "' +taskname1+' "/> \n </flowsInto> \n <join
code="xor" /> \n <split code="and" /> \n <resourcing> \n <offer
initiator="user" /> \n <allocate initiator="user" /> \n <start initiator="user"
/> \n </resourcing> \n </task>'+'\n')
```

➤ Règle 3 : TA2XS (priorité : 3)

**Condition :**

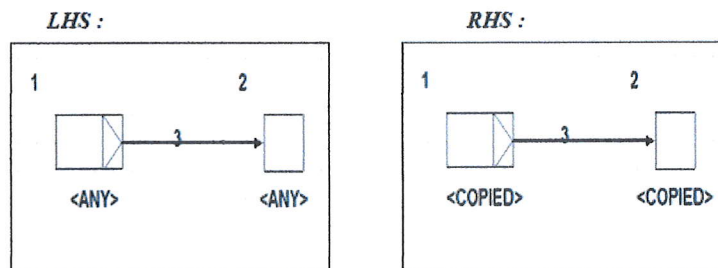
```
node2 = self.getMatched(graphID, self.LHS.nodeWithLabel(3))
return not hasattr(node2, "TA2XS_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
taskname = node.name.getValue()
node1 = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
taskname1 = node1.name.getValue()

node2 = self.getMatched(graphID, self.LHS.nodeWithLabel(3))
node2.TA2XS_executed = True
obFichier = open('fichyawl.yawl', 'a')
obFichier.write('<task id="'+taskname+' "> \n <name>'+taskname+'</name>\n
<flowsInto> \n <nextElementRef id= "' +taskname1+'"/> \n </flowsInto> \n <join
code="xor" /> \n <split code="and" /> \n <resourcing> \n <offer
initiator="user" /> \n <allocate initiator="user" /> \n <start initiator="user"
/> \n </resourcing> \n </task>'+'\n')
```

➤ Règle 4 : XJ2TA (priorité : 6)

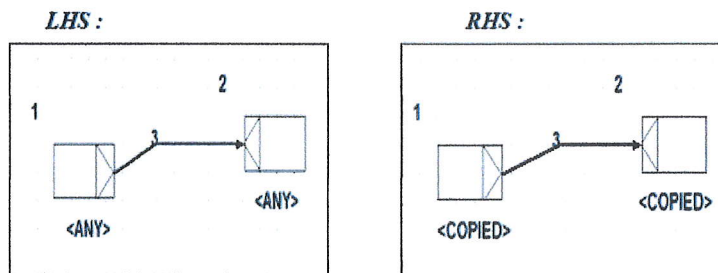
**Condition :**

```
node2 = self.getMatched(graphID, self.LHS.nodeWithLabel(3))
return not hasattr(node2, "XS2TA_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
taskname = node.name.getValue()
node1 = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
taskname1 = node1.name.getValue()
node2 = self.getMatched(graphID, self.LHS.nodeWithLabel(3))
node2.XS2TA_executed = True
obFichier = open('fichyawl.yawl', 'a')
obFichier.write('<task id= "' +taskname+' "> \n <flowsInto> \n <nextElementRef
id= "' +taskname1+'"/> \n </flowsInto> \n <join code="xor" /> \n <split
code="xor" /> \n <resourcing> \n <offer initiator="user" /> \n <allocate
initiator="user" /> \n <start initiator="user" /> \n </resourcing> \n
</task>'+'\n')
```


☛ Règle 5 :XS2XJ (priorité :5)



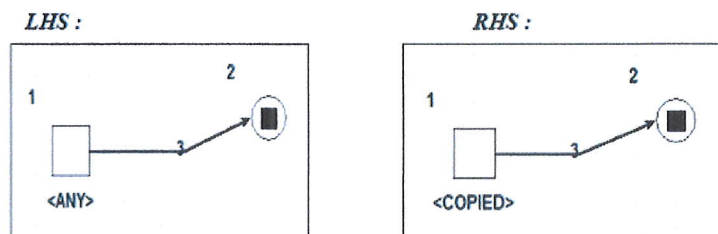
Condition :

```
node2 = self.getMatched(graphID, self.LHS.nodeWithLabel(3))
return not hasattr(node2, "XS2XJ_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
taskname = node.name.getValue()
node1 = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
taskname1 = node1.name.getValue()
node2 = self.getMatched(graphID, self.LHS.nodeWithLabel(3))
node2.XS2XJ_executed = True
obFichier = open('fichyawl.yawl', 'a')
obFichier.write('<task id= "' + taskname + '"> \n <flowsInto> \n <nextElementRef
id= "' + taskname1 + '" /> \n </flowsInto> \n <join code="xor" /> \n <split
code="xor" /> \n <resourcing> \n <offer initiator="user" /> \n <allocate
initiator="user" /> \n <start initiator="user" /> \n </resourcing> \n
</task>'+'\n')
```

☛ Règle 6 : TA2OC (priorité : 7)



Condition :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
return not hasattr(node, "TA2OC_executed")
```

Action :

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
node1 = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
taskname = node1.name.getValue()
node.TA2OC_executed = True

obFichier = open('fichyawl.yawl', 'a')
obFichier.write('<task id= "' + taskname + '"> \n <name> '+taskname+' </name>\n
<flowsInto> \n <nextElementRef id= "OutputCondition"/> \n </flowsInto> \n
<join code="xor" /> \n <split code="and" /> \n <resourcing> \n <offer
initiator="user" /> \n <allocate initiator="user" /> \n <start initiator="user"
/> \n </resourcing> \n </task>'+'\n')
```

✓ L'action finale :

```

obFichier = open('fichyawl.yawl','a')
obFichier.write('<outputCondition id="OutputCondition" />')
obFichier.write('\n')
obFichier.write('</processControlElements>')
obFichier.write('\n')
obFichier.write('</decomposition>')
obFichier.write('\n')
obFichier.write('</specification>')
obFichier.write('\n')
obFichier.write('</specificationSet> ')
obFichier.close()

```

FIGURE 4.15 – l'action finale sous ATOM³.

4.4.2 Exemple

Nous avons appliqué la grammaire sur le modèle YAWL précédent (voir la figure 4.13) résultat de la première grammaire.

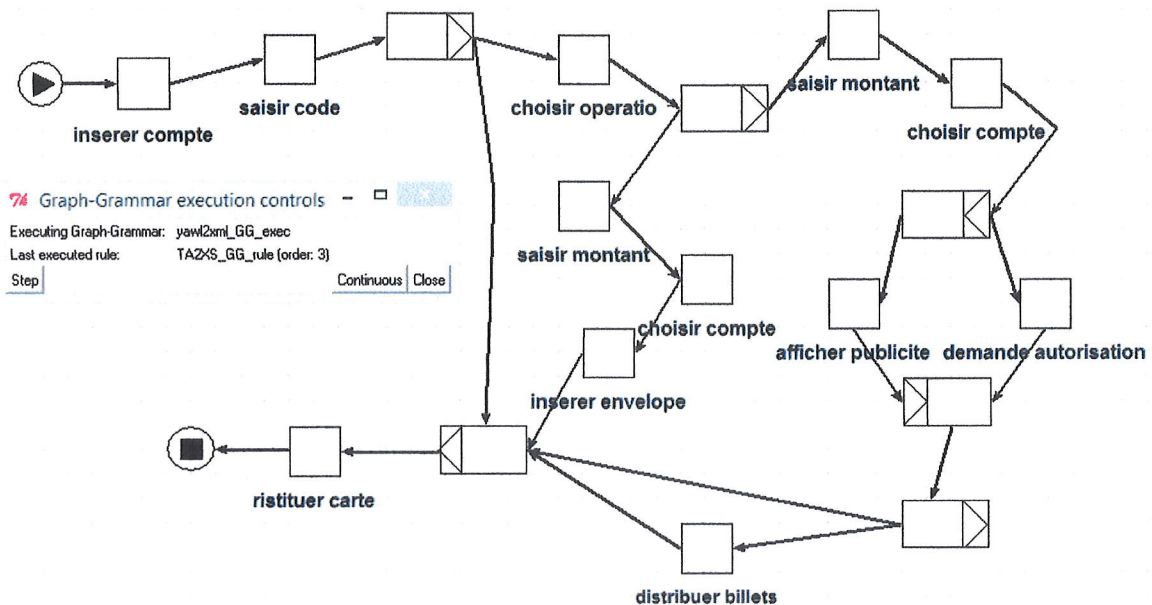


FIGURE 4.16 – modèle YAWL durant l'exécution.

Le résultat est un fichier XML « .yawl » qui contient toutes les informations existantes dans le modèle graphique. Ce fichier est prêt pour être chargé dans un outil de vérification. la figure ci-dessous montre un extrait de ce fichier.

```
<?xml version="1.0" encoding="UTF-8"?>
<specificationSet xmlns="http://www.yawlfoundation.org/yawlschema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" version="4.0" xsi:schemaLocation="http://www.yawlfoundation.org/yawlschema
http://www.yawlfoundation.org/yawlschema/YAWL_Schema4.0.xsd">
<specification uri="New_Specification">
<documentation>No description provided</documentation>
<metaData>
<creator>Chamou</creator>
<description>No description provided</description>
<coverage>4.2.744</coverage>
<version>0.6</version>
<persistent>>false</persistent>
<identifier>UID_2717ae36-bf87-4bf5-a9d5-a2ea56b9a7a9</identifier>
</metaData>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" />
<decomposition id="Net" isRootNet="true" xsi:type="NetFactsType">
<processControlElements>
<inputCondition id="InputCondition">
  <flowsInto>
    <nextElementRef id= inserer compte/>
  </flowsInto>
</inputCondition><task id="inserer compte">
  <name> inserer compte </name>
  <flowsInto>
    <nextElementRef id= "saisir code"/>
  </flowsInto>
  <join code="xor" />
  <split code="and" />
  <resourcing>
    <offer initiator="user" />
    <allocate initiator="user" />
    <start initiator="user" />
  </resourcing>
</task>
<task id="saisir montant">
  <name> saisir montant </name>
  .
</processControlElements>
</decomposition>
</specification>
</specificationSet>
```

FIGURE 4.17 – le fichier « .yawl » généré.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by Lachlan Aldred (Queensland University of Technology) -->
3 <!--
4 <xs:schema targetNamespace="http://www.citi.qut.edu.au/yawl" xmlns:yawl="http://www.citi.qut.edu.au/yawl" xmlns:xs="http://www.w3.org/2001
5 -->
6 <xs:schema targetNamespace="http://www.citi.qut.edu.au/yawl" xmlns:yawl="http://www.citi.qut.edu.au/yawl" xmlns:xs="http://www.w3.org/2001
7 <!--
8 #####
9 DECLARE ROOT ELEMENT - YAWL_Specification
10 #####
11 -->
12 <xs:element name="specificationSet" type="yawl:SpecificationSetFactsType">
13 <xs:unique name="SpecificationUnique">
14 <xs:selector xpath="specification"/>
15 <xs:field xpath="@uri"/>
16 </xs:unique>
17 <xs:key name="DecompositionKey">
18 <xs:selector xpath="yawl:specification/yawl:decomposition"/>
19 <xs:field xpath="@id"/>
20 </xs:key>
21 <xs:key name="NetElementKey">
22 <xs:selector xpath="yawl:specification/*/yawl:processControlElements/*"/>
23 <xs:field xpath="@id"/>
24 </xs:key>
25 <xs:keyref name="FlowsIntoForeignKey" refer="yawl:NetElementKey">
26 <xs:selector xpath="yawl:specification/*/yawl:processControlElements/*/yawl:flowsInto/nextElementRef"/>
27 <xs:field xpath="@id"/>
28 </xs:keyref>
29 <xs:keyref name="RemovesTokensForeignKey" refer="yawl:NetElementKey">
30 <xs:selector xpath="yawl:specification/*/yawl:processControlElements/*/yawl:removesTokens"/>
```

FIGURE 4.18 – le fichier XML Schema .

4.5 vérification

La vérification répond à la question « Construisons-nous correctement le modèle ? » La vérification est l'ensemble des actions de revue, inspection, test, preuve automatique, ou autres techniques appropriées permettant d'établir et de documenter la conformité des artefacts du développement vis-à-vis des critères préétablis [32].

Le but de transformer les diagrammes d'activités vers les modèles YAWL est de pouvoir les vérifier. Pour ce faire, nous avons choisi l'outil YAWL présenté dans le chapitre 2.

nous avons charger le code ".yawl" générer précédemment (voir la figure) pour procéder la vérification. la figure 4-18 montre notre modèle charger dans l'outil YAWL 4.2.

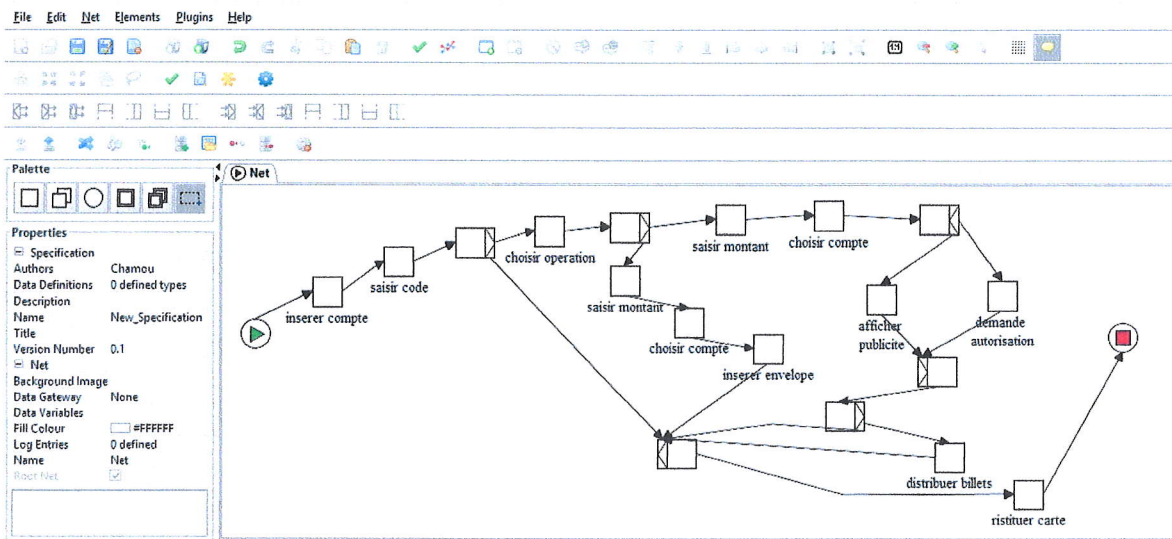


FIGURE 4.19 – exemple de modèle YAWL charger sur l’outil.

Nous avons essayé de déstabilisé notre diagramme d’activité exemple par la suppression de l’arc qui relie AND-split et la tâche demande autorisation. Après nous avons régénéré et recharger notre code "yawl" et nous avons utilisé un des techniques d’analyse inclut dans l’outil YAWL 4.2 (soundness vérification).

☛ **Propriété Soundness (solidité) :**

Chaque processus métier doit présenter certaines caractéristiques souhaitables. Tout d’abord, il est important de savoir qu’un processus, une fois démarré, peut toujours se terminer (option à compléter). Deuxièmement, aucune autre tâche ne devrait encore être en cours d’exécution pour ce processus à la fin du processus (bon achèvement). Troisièmement, le processus ne doit pas contenir de tâches qui ne seront jamais exécutées (Pas de transitions mortes). La combinaison de ces trois propriétés souhaitables est connue sous le nom de propriété de solidité [38].

la figure ci-dessous montre le modèle YAWL résultat après la régénération.

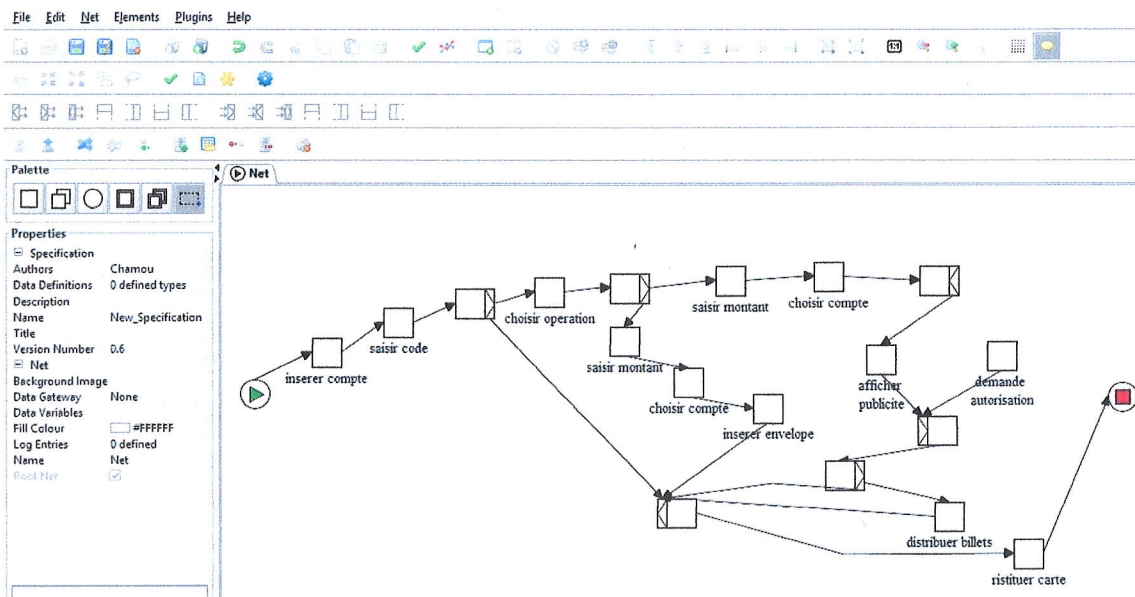


FIGURE 4.20 – exemple de modèle YAWL avec un problème de solidité.

Les résultats sont indiqués dans la figure ci-dessous :

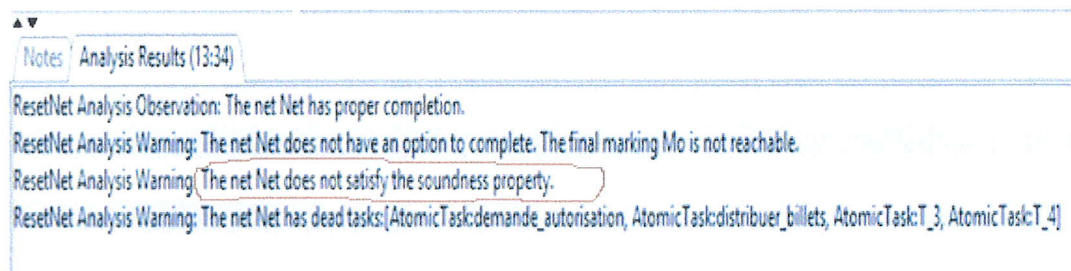


FIGURE 4.21 – Les résultats de vérification.

4.6 Conclusion

Dans ce chapitre, nous avons abordé la notion de graphe, les types de graphe, le principe de transformation de graphe, ainsi que les outils de transformation de graphe. Nous avons adopté et présenté AToM³ l'un des outils pour implémenter cette approche.

Après nous avons proposé l'approche de transformation de diagramme d'activité vers YAWL, premièrement nous allons présenter le méta-modèle source et cible, les environnements générés, ensuite l'ensemble des règles de transformation qui permettent de transformer n'importe quel diagramme d'activité UML décrit dans le canevas de l'AToM³ vers un modèle YAWL. Après, nous avons développé une autre grammaire de graphe qui fait le codage de

Conclusion générale

Le travail présenté dans ce mémoire s'inscrit dans le domaine de l'ingénierie dirigée par les modèles. Nous avons commencé par avec une présentation des deux langages, UML comme langage de modélisation orientée objet et YAWL, comme langage de modélisation de processus métier. Nous avons proposé des règles théoriques de transformation des diagrammes d'activités d'UML 2.0 vers les modèles YAWL, ensuite nous avons présenté notre approche pour implémenter ces règles en se basant sur la transformation de graphe à l'aide de l'outil AToM³.

Cette approche a été réalisée dans deux étapes essentielles :

- La première étape consiste à proposer un méta-modèle pour les diagrammes d'activité, et un autre pour les modèles YAWL, afin de générer un outil visuel permettant la modélisation de diagramme d'activité et de modèle YAWL.
- La deuxième étape contient une grammaire de graphe permettant de générer automatiquement les modèles YAWL à partir des diagrammes d'activité. Ensuite, et dans le but de valider notre approche, nous avons dû proposer une deuxième grammaire qui transforme les modèles YAWL obtenus par la première grammaire en format textuelle, afin de les charger dans un outil d'analyse et de vérification, pour commencer la tâche de vérification.

Dans un futur travail, nous voulons continuer l'implémentation des règles qui transforment les types d'action dans un diagramme d'activité ainsi que les noeuds qu'on n'a pas présenté dans ce mémoire à savoir les noeuds exécutables et les noeuds d'objets, afin d'arriver à une transformation complète de la totalité des éléments des diagrammes d'activité. Nous planifions aussi étendre notre approche en transformant d'autres diagrammes UML afin de collecter plus d'informations fournies par ces modèles.

Un autre défi est d'essayer de réaliser une implémentation des règles de transformation dans d'autres outils de transformation afin de comparer les performances et choisir la bonne technique.

Bibliographie

- [1] Benoît Charroux, Aomar Osmani, and Yann Thierry-Mieg. *UML 2 Pratique de la modélisation*. 2ème édition.
- [2] Bendiaf Messaoud. Spécification vérification des systèmes embarqués temps réel en utilisant la logique de réécriture, 15.05.2018.
- [3] Houda Hamrouche. Une approche de transformation des diagrammes d'activité d'uml vers csp basée sur la transformation de graphes. Master's thesis, Université 20 Août-1955-SKIKDA.
- [4] <https://www.lucidchart.com/pages/fr/quest-ce-que-le-langage-de-modelisation-unifie>.
- [5] Zahoui Anissa Amel. Développement d'une chaîne d'outil en fonction du nouveau standard fondationnel uml(fuml). Master's thesis, Université Badji Mokhtar, 2014.
- [6] Lahouaoula Manel. Une approche de modélisation et de vérification des systèmes complexes en utilisant uml et rdp. Master's thesis, Université 20 Août-1955-Skikda, 2017.
- [7] Boudia Malika. Transformation des diagrammes d'états-transitions vers maude. Master's thesis, Université De M'sila, 2007.
- [8] Ghammit Samia. Une approche automatique de transformation de diagrammes d'état transition vers le pi-calcul. Master's thesis, Université 20 Août-1955-Skikda, 2015.
- [9] [https://fr.wikipedia.org/wiki/UML_\(informatique\)](https://fr.wikipedia.org/wiki/UML_(informatique)).
- [10] Pascal Roques and Franck Vallée. *UML2 En Action De L'analyse Des Besoins à La Conception*. 4ème édition.
- [11] Kirli Asma. Bouhini Meryem. Une approche dirigée par les modèles basée sur uml pour la mise en oeuvre de services web composés. Master's thesis, Université de Djilali Bounaâma Khemis Miliana, 2017.
- [12] Med Saad djabellah mehdia. Une approche de transformation de diagramme d'activité uml vers pi-calcul basée sur la transformation de graphe. Master's thesis, Université 20 Août-1955-SKIKDA, 2015.
- [13] <https://laurent-audibert.developpez.com/Cours-UML/?page=diagramme-activites>.

- [14] Guerrouf Fayçal. Une approche de transformation des diagrammes d'activités d'uml mobile 2.0 vers les réseaux de petri. Master's thesis, Université El Hadj Lakhdar-Batna.
- [15] https://fr.wikiversity.org/wiki/Mod%C3%A9lisation_UML/Le_diagramme_d%27activit%C3%A9.
- [16] Aymen Baouab. Gouvernance et supervision décentralisée des chorégraphies inter-organisationnelles, 27.06.2013.
- [17] Mounira Zerari. Une approche pour l'amélioration de la flexibilité des processus métier basée sur les techniques du process mining, 12.03.2012.
- [18] Mohamed Boukhebouze. Gestion de changement et vérification formelle de processus métier : une approche orientée règle, 2010.
- [19] Pascal Arthur H. M. ter Hofstede, Wil M. P. van der Aalst, Michael Adams, and Nick Russell. *Modern Business Process Automation*.
- [20] Mohammed Oussama Kherbouche. Contribution à la gestion de l'évolution des processus métiers., 2013.
- [21] Elio Goettelmann. Modélisation de processus métiers sensibilisés aux risques et déploiement en confiance dans le cloud, 21 Octobre 2015.
- [22] Zhen Chen. Workflow management theories and techniques including the data perspective, 27 August 2012.
- [23] Nick Russell and Arthur H. M. ter Hofstede. Surmounting bpm challenges : the yawl story, 6.03.2009.
- [24] Marlon Dumas Arthur H. M. ter Hofstede Wil M. P. van der Aalst, Lachlan Aldred. *Design and implementation of the YAWL system*.
- [25] Lindsay Bradford and Marlon Dumas. Getting started with yawl. 2007.
- [26] W.M.P. van der Aalst A.H.M. ter Hofstede M.T. Wynn, H.M.W. Verbeek and D. Edmond. Business process verification - finally a reality!
- [27] <http://www.yawlfoundation.org/>.
- [28] Firas Bacha. Une approche mda pour l'intégration de la personnalisation du contenu dans la conception et la génération des applications interactives, 2013.
- [29] El hillali Kerkouche. Modélisation multi-paradigme : Une approche basée sur la transformation de graphes, 2011.
- [30] Mouna Bouarioua. Une approche basée transformation de graphes pour la génération de modèles de réseaux de petri analysables à partir de diagrammes uml, 2013.
- [31] Kholadi Mohamed Naoufel. Une approche de transformation de la notation bpmn vers bpel basée sur la transformation de graphe. Master's thesis, Université Mentouri Constantine.

- [32] Raida ElMansouri. Modélisation et vérification des processus métiers dans les entreprises virtuelles : Une approche basée sur la transformation de graphes.
- [33] Agg. <http://tfs.cs.tu-berlin.de/agg/>,.
- [34] Fujaba. <http://www.fujaba.de/>.
- [35] Atom3. <http://atom3.cs.mcgill.ca/>.
- [36] Viatra. <http://dev.eclipse.org/viewcvs/indextech.cgi/gmthome/subprojects/VIA-TRA2/index.html>.
- [37] Great. <http://www.escherinstitute.org/Plone/tools/suites/mic/great/>.
- [38] Yawl - user manual. <http://www.yawlfoundation.org/manuals/YAWLUserManual4.1.pdf>.
- [39] https://github.com/yawlfoundation/yawl/blob/master/schema/YAWL_schema.xsd.



RÉSUMÉ

Actuellement, l'UML est devenu un standard largement accepté dans l'industrie de développement de logiciels orienté objet. Certains diagrammes d'UML ont été utilisés pour modéliser la structure d'un système, d'autres sont utilisés pour modéliser son comportement. Les diagrammes d'activité c'est un diagramme d'UML pour modéliser le comportement. Ils sont utilisés pour modéliser les systèmes workflow, les systèmes orientés services et les processus métiers. Cette dernière propriété la modélisation de processus métier est commune avec le modèle YAWL.

Le travail présenté dans ce manuscrit s'inscrit dans le domaine de l'Ingénierie Dirigée par les Modèles (IDM), on exploite la transformation des modèles.

Dans ce travail nous proposons une approche automatique de transformation de diagramme d'activité vers le modèle YAWL basée sur la transformation de graphe. cette approche consiste à proposer un méta-modèle pour le diagramme d'activité et un autre pour le modèle YAWL.

Après, nous proposons une première grammaire de graphe qui fait la transformation des diagrammes d'activité UML vers les modèles YAWL. Une deuxième grammaire de graphe est développée pour transformer les modèles YAWL vers un format textuel afin de les charger dans un outil qui permet l'analyse et de vérification. L'outil *AToM³* est utilisé.

Mots clés : UML, YAWL, Ingénierie Dirigée par les Modèles, Transformation de graphe, AToM³.

ABSTRACT

Currently, UML has become a widely accepted standard in the object-oriented software development industry. Some UML diagrams have been used to model the structure of a system, others are used to model its behavior. Activity diagrams is a UML diagram for modeling behavior. They are used to model workflow systems, service-oriented systems, and business processes. This last property of business process modeling is common with the YAWL model.

The work presented in this manuscript is in the field of Model Driven Engineering (IDM), we exploit the transformation of models.

In this work we propose an automatic approach of transformation of activity diagram towards the YAWL model based on the transformation of graph. this approach consists in proposing a meta-model for the activity diagram and another for the YAWL model.

Next, we propose a first graph grammar that transforms UML activity diagrams into YAWL models. A second graph grammar is developed to transform YAWL models to a textual format in order to load them into a tool that allows for analysis and verification. The *AToM³* tool is used.

Key words : UML, YAWL, Graph transformation, Model Driven Engineering, AToM³.