République Algérienne Démocratique et Populaire
Ministère de l' Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Seddik Ben Yahia 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'information et aide à la décision

Thème

Spécification Maude de Diagramme Global d'Interaction : Une Approche Automatique Basée sur la Transformation de Graphe

Élaboré et présenté par :
Bellour Farida

Encadré par :
Mlle.DJAOUI Chafika

Année Universitaire : 2018-2019

République Algérienne Démocratique et Populaire
Ministère de l' Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Seddik Ben Yahia 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'information et aide à la décision

Thème

Spécification Maude de Diagramme Global d'Interaction : Une Approche Automatique Basée sur la Transformation de Graphe

Élaboré et présenté par :
Bellour Farida

Encadré par :
Mlle.DJAOUI Chafika

Année Universitaire : 2018-2019

⁂ Remerciements ⁂

Je tiens tout d'abord à remercier Dieu tout puissant et miséricordieux, qui m'a donné la force et la patience d'accomplir ce travail.

Nous tenons aussi à exprimer ma sincère reconnaissance et ma profonde gratitude à Mon encadrante Mlle " ***DJAOUI Chafika*** *" 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.*

Mes vifs remerciements également aux ***membres du jury*** *pour leurs attentions et intérêts portés envers notre travail. Merci de nous avoir honorés de votre présence.*

*Je remercie également mon enseignante Mlle "****BOUAZIZE Hamida****" pour ses efforts et son aide précieuse.*

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

Je tiens également à remercier le corps professoral et administratif de la Faculté des Sciences Exact 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 aux personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire.

Enfin, je tiens à remercier les personnes qui m'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 :

Mon très cher ***Père Mahmoud :*** *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 et le fruit de vos sacrifices que vous avez consentis pour mon éducation et ma formation le long de ces années.*

Ma tendre ***Mère Chafia :*** *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.*

Ma ***très cher frère " Farouk "****pour leurs appuis et leurs encouragements.*

Mes ***très chères frères " Zin el Abidin, Zakariya, Abd Rahim "***
pour leurs encouragements permanents, et leurs soutiens morales.

Mes très chères amies ***Roqiya, Asma, Ahlem, Salima, Lamia, Widad, Selma.***

Mes oncles et tantes et à toute ma famille.

Tous mes enseignants depuis ma première année d'études.

Tous les membres de ma promotion.

Sans oublier tous mes amis, proffesseur, famille, connaissance que je ne pourrai citer. Je vous dis merci.

Merci d'être toujours là pour moi

FARIDA

Table des matières

Table des matières i

Liste des tableaux iv

Table des figures vi

Liste des abréviations vii

Introduction générale 1

1 Transformation de modèle et ATOM^3 4

1.1 Introduction 4

1.2 Ingénierie Dirigée par les Modèles (IDM) 5

1.2.1 Définition de l'approche IDM 5

1.2.2 Notions générales de l'IDM 5

1.3 L'Architecture Dirigée par les Modèles (MDA) 6

1.3.1 Architecture du MDA 7

1.3.2 L'architecture à quatre niveaux de MDA 7

1.4 Transformation des modèles 9

1.4.1 Définition de transformation 9

1.4.2 Principe de transformation 9

1.4.3 Typologie des modèles dans l'approche MDA 10

1.4.4 Transformation de modèles dans MDA 10

1.4.5 Type de transformation 12

1.4.6 Propriété de transformation des modèles 13

1.5 Techniques de transformation des Modèles 14

1.6 Transformation de graphe 16
1.6.1 Concepts de graphe 16
1.6.2 Grammaire de Graphes 17
1.7 Outils de transformations de graphes 18
1.8 conclusion 20

2 Le diagramme global d'interaction d'UML **21**
2.1 Introduction 21
2.2 Les diagrammes d'UML 2.0 22
2.2.1 Diagrammes structurels (statiques) 22
2.2.2 Comportementaux (dynamiques) 22
2.2.3 Diagrammes d'interaction 23
2.3 Le diagramme global d'interaction (I.O.D) 23
2.3.1 Présentation 23
2.3.2 Les composants du diagramme global d'interaction 24
2.3.3 Relations entre les éléments du diagramme global d'interaction 27
2.4 Diagramme de Communication 28
2.4.1 Les Composants du Diagramme de Communication 29
2.4.2 Exemple de diagramme de communication 30
2.5 Exemple d'un IOD 31
2.6 Conclusion 32

3 Logique de Réécriture et Maude **33**
3.1 Introduction 33
3.2 La logique de réécriture 34
3.3 Définition formelle 34
3.4 Le Langage Maude 37
3.4.1 Présentation de Maude 37
3.4.2 Les concepts de Maude 37
3.4.3 Les niveaux de Maude 42
3.4.4 La programmation paramétrée sous Maude 42
3.4.5 L'outil Maude : 43

3.4.6 Les commandes les plus utilisées dans Maude 44
3.4.7 Les caractéristiques de Maude 45
3.5 Conclusion 46

4 Une Approche de transformation de IOD vers Maude 47
4.1 Introduction 47
4.2 Scénario de la transformation 47
4.3 Transformation du diagramme global d'interaction vers le langage Maude . 48
4.3.1 méta-modèle de diagramme global d'interaction 48
4.3.2 Les classes et leurs relations 50
4.3.3 Génération de l'environnement 51
4.3.4 Les Contraintes 52
4.3.5 La grammaire de graphe proposée 55
4.4 Exemple 64
4.5 Conclusion 66

Conclusion générale 67

Bibliographie 69

Liste des tableaux

2.1 Les nœuds de contrôle [1]. 26
2.2 Les notations des nœuds d'interaction. 27

3.1 Représentation graphique des règles de déduction. 36
3.2 Les attributs des opérations dans Maude. 41
3.3 Quelques commandes dans Maude. 45

Table des figures

1.1 Concepts d'approche IDM [21]. 5
1.2 Relation entre système, modèle, méta-modèle et méta méta-modèle [10]. . . 6
1.3 Architecture du MDA [11]. 7
1.4 Pyramide de modélisation à quatre niveaux [23]. 8
1.5 Concept de base de la transformation de modèles [28]. 9
1.6 Les modèles et les transformations dans l'approche MDA [39]. 11
1.7 Les types de transformation et leur principale utilisation [23]. 13
1.8 Les approches de transformation de modèles [4]. 14
1.9 Les deux types de graphe [26]. 17
1.10 Principe de l'application d'une règle de transformation [8]. 18
1.11 L'outil d'ATOM^3. 19
1.12 Le canevas d'ATOM^3. 20

2.1 Notation arc 27
2.2 Illustration des relations entre les éléments des IODs. 28
2.3 liens entre les Objet 29
2.4 Les types des messages 29
2.5 Diagramme de communication[15]. 30
2.6 Diagramme de communication illustrant la recherche puis l'ajout, dans son panier virtuel, d'un livre lors d'une commande sur Internet[19]. 30
2.7 Exemple de diagramme d'interaction[20]. 31

3.1 Déclaration de sorte et de sous sorte. 39
3.2 Déclaration des opérations. 40
3.3 Attributs des opérations. 41
3.4 Exécution de Maude 44

4.1 Schéma de l'Approche proposée. 48
4.2 Le méta-modèle du modèle global d'interaction proposé. 49
4.3 Environnement de diagramme d'interaction sous AToM^3. 52
4.4 Modèle IOD crée avec L'outil de modélisation. 53
4.5 Violation de contrainte de classe Inital_ Node avec l'outil de modélisation. . . 54
4.6 Violation de contrainte de classe Final_ Node avec l'outil de modélisation. . . 54
4.7 Violation de contrainte de classe Interaction_ Use avec l'outil de modélisation 55
4.8 La création du fichier sous AToM^3. 56
4.9 Création de Module fonctionnel de fichier sous AToM^3. 57
4.10 Fermeture de fichier dans l'action finale 63
4.11 Exemple montre un étudiant qui a été accepté dans une université modélisé avec l'outil. 64
4.12 Exécution des règles de transformation. 65
4.13 La description textuelle sous MAUDE. 66

Liste des abréviations

ATOM^3 A Tool for Multi-formalism and Meta-Modelling

CIM Computation Independent Model

IDM l'Ingénierie Dirigée par les Modèles

IOD Interaction Overview Diagram

LHS Left Hand Side

MDA Model Driven Architecture

MDE Model Driven Engineering

MOF Méta Object Facility

OCL Object Constraint Language

OMG Object Management Group

PIM Platform Independent Model

PSM Platform Specic Model

RHS Right Hand Side

UML Unified Modeling Language

OOD Développement Orienté Objet

XMI XML Meta data Interchange

Introduction générale

Pour catégoriser le degré de formalisation d'un langage, la classification est en quatre degrés : informel, semi-informel, semi-formel et formel. 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 des modèles des systèmes.

Depuis sa standardisation par l'Object Management Group (OMG) en 1997, l'Unified Modeling Language (UML) s'impose progressivement comme un standard de fait pour la modélisation à objets de systèmes, qu'ils soient logiciels, matériels ou organisationnels. Cependant, UML présente quelques inconvénients, il n'était pas possible de prouver d'une manière formelle que certaines situations ne seront jamais atteintes (sureté), ni que certains états ne puissent toujours être atteints si cela s'avère nécessaire (vivacité), compte tenu des contraintes temporelles.

L'utilisation d'une approche de transformation permet la combinaison d'un langage de spécification formelle éprouvé et une technique de vérification de modèles qui permettra de valider formellement les modèles UML développés. Ces approches utilisent des méthodes formelles en transformant les modèles UML en des formalismes performants favorisant l'analyse mathématique par le biais de langages formels tel que MAUDE. Une telle transformation de modèles, prend en entrée un modèle généré depuis un méta-modèle source et produit en sortie le code du langage de programmation formel correspond, dans notre cas c'est le langage MAUDE.

Notre travail consiste à proposer une technique permettant de faire le passage des diagrammes globaux d'interactions depuis l'état de modèle dans l'outil ATOM^3 (qui est un outil permettant de représenter graphiquement ce genre de diagramme) vers un état textuel dans un langage de vérification formelle appelé MAUDE (qui est un langage de programmation formel et déclaratif, utilisant des descriptions mathématiques). Ce passage est basé principalement

sur le concept de transformation de graphes, et est effectif grâce à une grammaire de graphes. Cette dernière est implémentée par l'outil ATOM^3, qui supporte la méta-modélisation et les transformations de graphes.

Problématique et objectif

Dans notre mémoire, nous proposons une approche de transformation des diagrammes globaux d'interaction vers Maude. Ce travail est inscrit dans le contexte de l'IDM (Ingénierie dirigée par modèles) ou on utilise et exploite la transformation de modèles. Plus précisément, la transformation de graphe dont elle facilite la modélisation des modèles qui sont décrits comme des graphes, les transformations entre modèles sont effectuées par réécriture des graphes et peuvent être décrites aussi comme des graphes sous forme de grammaire de graphes. Chaque transformation prend des modèles en entrée et produit des modèles en sortie. Alors l'approche que nous définissons vise à proposer :

- ✓ Un outil de modélisations des diagrammes global d'interaction.
- ✓ Une grammaire de graphe pour appliquer la transformation des diagrammes global d'interaction vers Maude.

Pour cela, nous utilisons ATOM^3 (A Tool for Multi-formalism and Meta-Modelling) qui permet de réaliser ces concepts.

Organisation du mémoire

Nous avons organisé notre mémoire en quatre chapitres :

Le premier chapitre : Transformation de modèle et ATOM^3

Consacré à la présentation de l'approche IDM et la transformation de modèles.

Le deuxième chapitre : Diagramme global d'interaction

Nous définissons brièvement les diagrammes globaux d'interaction d'UML 2 avec diagramme de communication.

Le troisième chapitre : Logique de réécriture et Maude

Nous présenterons la logique de réécriture qui est un élément de base de notre travail ainsi que ses notions élémentaires. Puis nous aborderons le langage Maude qui se base sur cette

logique avec ses différents concepts.

Le quatrième chapitre : Une Approche de transformation de IOD vers Maude

Nous détaillons dans le dernier chapitre, notre Approche de transformation qui est basée sur la transformation des graphes en utilisant l'outil ATOM^3.

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.

CHAPITRE 1

Transformation de modèle et ATOM^3

1.1 Introduction

La transformation de modèle est une activité centrale dans le développement de logiciels dirigé par les modèles. Elle est utilisée aussi pour l'optimisation des modèles et d'autres formes d'évolutions des modèles. En outre, la transformation du modèle est utilisée pour le mappage des modèles entre les différents domaines pour les analyser ou pour la génération automatique de code à partir d'eux-mêmes. Le besoin d'une convention de conception et de communication est la raison principale de l'apparition d'UML qui est devenu une pratique indispensable au milieu des développeurs mais la croissance de la complexité des systèmes rend la vérification de ses diagrammes un besoin crucial.

L'approche IDM (Ingénierie Dirigée par les Modèles), est basée sur l'utilisation des modèles et méta-modèles. Ce principe innovant a apporté de nombreuses améliorations dans le développement et la conception des systèmes logiciels complexes par rapport au principe de la programmation classique. Dans ce chapitre nous allons commencer par une présentation des notions du IMD et MDA. Puis nous discuterons en détail la transformation de modèles : ses types, ses propriétés, par la suite nous s'intéressons par la transformation de graphe et la grammaire de graphe. Enfin, une présentation de l'outil ATOM^3.

1.2 Ingénierie Dirigée par les Modèles (IDM)

1.2.1 Définition de l'approche IDM

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 [38].

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 [39].

La Figure 1.1 suivante représente les notions de base en ingénierie des modèles.

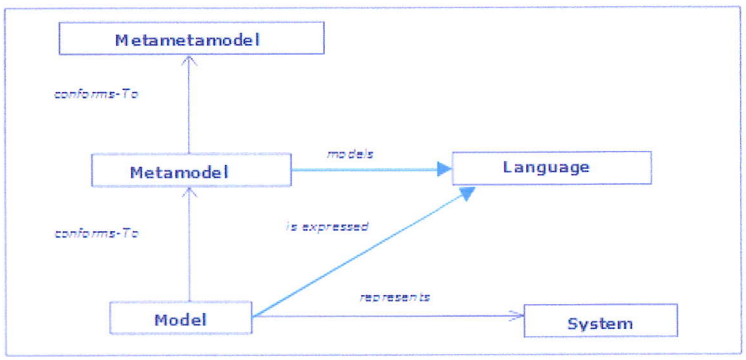


FIGURE 1.1 – Concepts d'approche IDM [21].

1.2.2 Notions générales de l'IDM

Dans cette section nous donnons les notions générales de l'approche IDM sur les Modèles, Méta-modèles, méta-modélisation, langage de modélisation et transformation de modèles.

- **Système :** Un système est une entité complexe traitée comme une totalité organisée formée d'élément et des relations entre ces éléments[40].

- **Modèle :**Un modèle est une représentation simplifiée d'un système où il est construit avec l'intention de décrire le système et répondre aux questions que l'on se pose sur lui. Donc un système est modélisé avec un ensemble de modèles où chacun capture un

aspect particulier. Par analogie avec les langages de programmation, un programme exécutable représente le système alors que le code source de ce programme représente le modèle.[4]

➢ **Méta-modélisation :** Un méta-modèle est un modèle qui définit le langage d'expression d'un modèle. Autrement dit, le méta-modèle représente (modélise) les entités d'un langage, leurs relations ainsi que leurs contraintes(une spécification de la syntaxe du langage). [8]

➢ **Méta-Méta-modèle :** Le méta-méta-modèle est un méta-modèle pour les méta-modèles utilisé tout naturellement pour désigner ce méta modèle particulier. Chaque élément du modèle est une instance d'un élément du méta-modèle.[6]

La figure suivante représente la relation entres les différents notions de l'IDM.

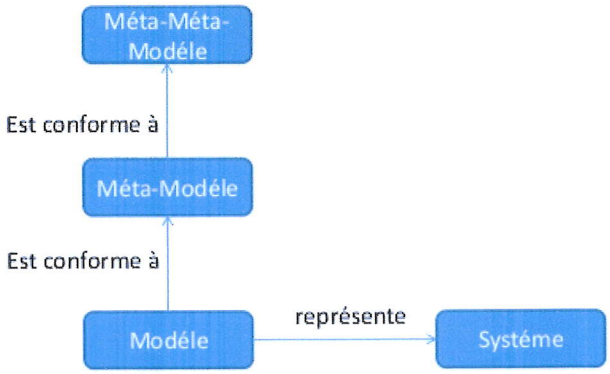


FIGURE 1.2 – Relation entre système, modèle, méta-modèle et méta méta-modèle [10].

1.3 L'Architecture Dirigée par les Modèles (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 [23].

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 [8].

1.3.1 Architecture du 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 [33].

- **UML** (Unified Modeling Language) :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 [12].
- **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 [8].
- **OCL** (Object Constraint Language) : Est un langage informatique d'expression des contraintes intégré à UML [12].
- **XMI** (XML Meta data Interchange) : Est un standard pour l'échange de métadonnées UML basé sur XML, qui donne une représentation concrète des modèles sous forme de documents XML [33].

La figure 1.3 illustres ces standards ainsi que leurs domaines d'application :

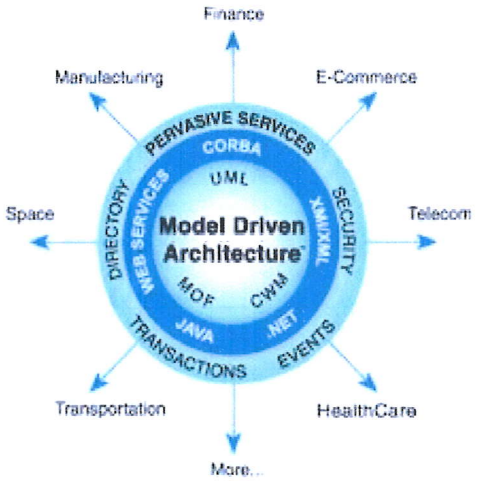


FIGURE 1.3 – Architecture du MDA [11].

1.3.2 L'architecture à quatre niveaux de MDA

Cette architecture est définie par l'OMG. Comme suit :

➢ **Le niveau M0 (instance) :** Le niveau M0 correspond au monde réel, il est composé des informations que l'on souhaite modéliser, Instance du modèle du niveau M1 [29].

➢ **Le niveau M1 (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 saux méta-modèles définit au niveau M2 [23].

➢ **Le niveau M2 (méta-modèle) :** représente toutes les instances d'un méta-méta-modèle. Il est composé de langages de spécifications ou de modélisation des modèles d'information (Le méta-modèle). Le méta-modèle UML décrit dans le standard UML définit la structure interne des modèles UML appartenant au niveau M2 [6].

➢ **Le niveau M3 (méta-méta-modèle) :** Il définit le langage de spécification du méta-modèle. Comme exemple Le MOF (Meta Object Facility) qui décrire lui-même [34].

La figure 1.4 représente ces différents niveaux de l'MDA

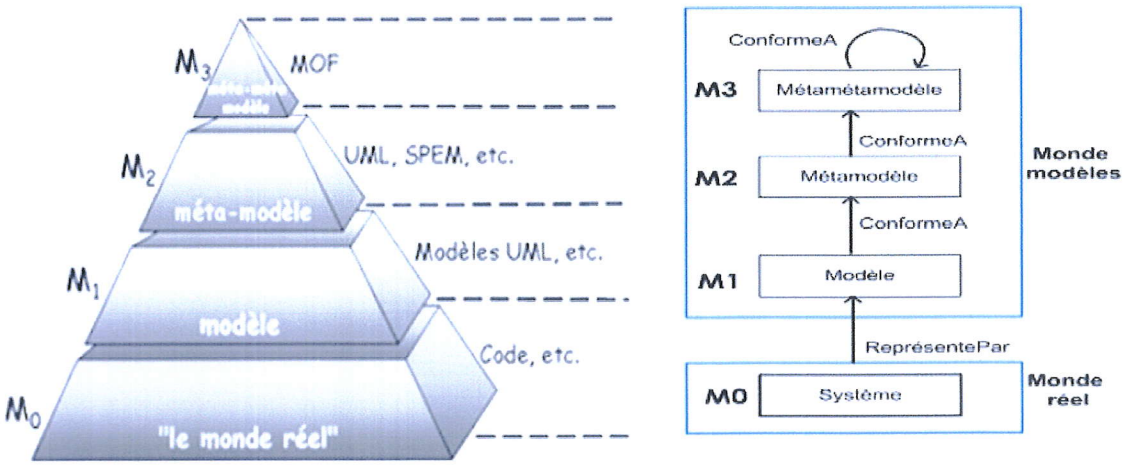


FIGURE 1.4 – Pyramide de modélisation à quatre niveaux [23].

Au niveau du **MDA** (**M**odel **D**riven **A**rchitecture) les modèles manipulés sont de natures diverses : modèles d'objets métiers, de processus, de service, de plate-forme, de transformation et de tests.

1.4 Transformation des modèles

1.4.1 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 [20].

1.4.2 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 [38].

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 [38].

La figure suivante résume le principe de transformation des modèles.

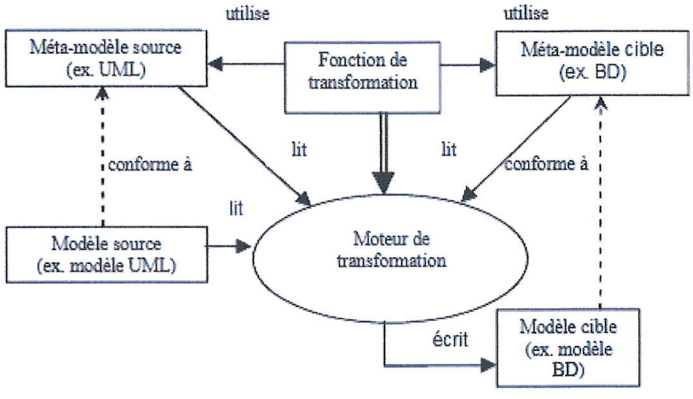


FIGURE 1.5 – Concept de base de la transformation de modèles [28].

1.4.3 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 : [8]

❖ **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 [12].

❖ **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 [8].
Le rôle des PIM est de donner une vision structurelle et dynamique de l'application, sans détail technique [33].

❖ **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 [8].

❖ **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 [12].

1.4.4 Transformation de modèles dans MDA

Plusieurs types de transformations de modèles sont définis dans l'approche MDA, nous illustrerons ces types dans la figure 1.6.

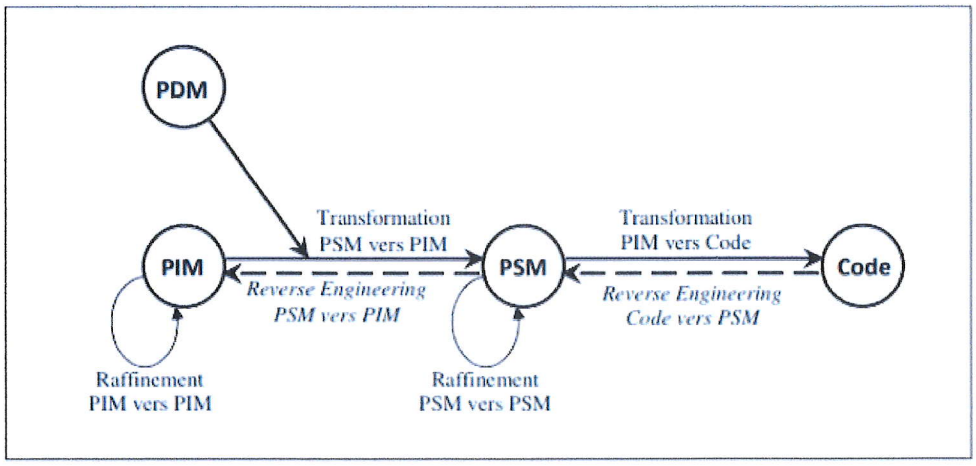


FIGURE 1.6 – Les modèles et les transformations dans l'approche MDA [39].

❑ 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 [8].

1.4.5 Type 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) [15].

❑ ***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 [15].

❑ ***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 [15].

✓ 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 [18].

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

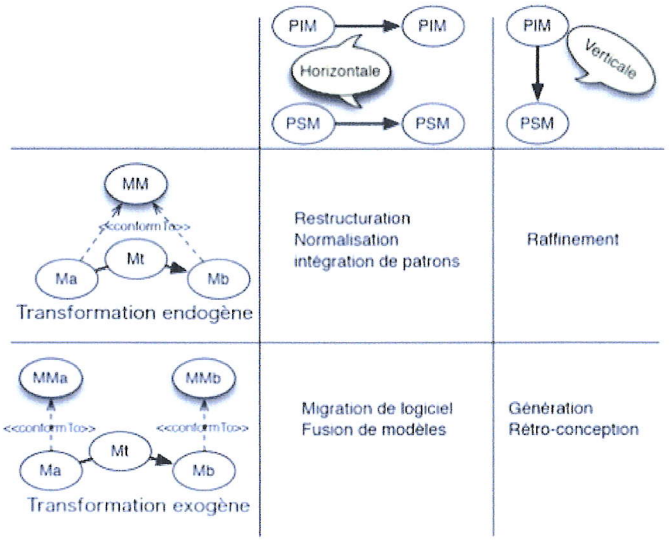


FIGURE 1.7 – Les types de transformation et leur principale utilisation [23].

1.4.6 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 [23].

❖ ***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 [23].

❖ ***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 [23].

❖ ***Ordonnancement des règles :***

C'est la suite des règles à exécuter lors d'une transformation. Il y a deux types d'ordonnancement : implicite si l'algorithme d'ordonnancement est défini par l'outil de transformation, sinon est dite explicite si d'autres mécanismes spécifient l'ordre d'exécution des règles [12].

❖ ***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 [12].

❖ ***L'incrémentalité :***

Cette propriété est liée à l'aptitude des modèles cibles à s'adapter aux changements des modèles sources [12].

❖ ***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 [23].

1.5 Techniques de transformation des Modèles

Les transformations de modèles peuvent être partagées en deux grandes classes : Les transformations Modèle vers Modèle et les transformations Modèle vers Code comme illustré dans la figure ci-dessous

<table>
<tr><th colspan="2">Approches de transformation de modèles</th></tr>
<tr><td>M2M</td><td>M2T</td></tr>
<tr><td>Dirigée par la structure</td><td rowspan="2">Parcours de modèles (programmation)</td></tr>
<tr><td>Manipulation directe</td></tr>
<tr><td>Approche relationnelle</td><td rowspan="3">Template</td></tr>
<tr><td>Transformation de graphes</td></tr>
<tr><td>Hybride</td></tr>
</table>

M2M : Model To Model.
M2T : Model To Text.

FIGURE 1.8 – Les approches de transformation de modèles [4].

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 [33].
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) [33].

➢ **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 [33].

➢ **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 [33].

➢ **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) [29].

➢ **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. [12].

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 [29].

➢ **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 [39].

1.6 Transformation de graphe

La modélisation des systèmes se heurte à la difficulté de la description de leurs structures complexes. Les modèles et méta-modèles (par exemple UML) possèdent le plus souvent une représentation sous forme de graphe. Par ailleurs, Les graphes offrent un support agréable et efficace qui permet la modélisation de systèmes. Par conséquent, les techniques de transformation et de réécriture des graphes peuvent être appliquées à la transformation de modèles. Avant d'aborder les techniques de transformation de graphes et les outils rendant de telles techniques applicables, il est nécessaire de rappeler quelques concepts de bases relatifs à la théorie de graphes[24].
La transformation de graphe est le processus de choisir une règle d'un ensemble indiqué, appliquer cette règle à un graphe et réitérer le processus jusqu'à ce qu'aucune règle ne puisse être appliquée.

1.6.1 Concepts de graphe

Le graphe est un moyen de la description des systèmes complexes, et aussi moyen de modélisation des concepts, simple à comprendre et expose meilleur visibilité. Un graphe est un schéma constitué des sommets reliés par des arêtes. Alors on appelle graphe **G** le couple (**S**,**A**) tel que **S** est un ensemble non vide des sommets et **A** ensemble non vide des arêtes.

- **Graphe non orienté :** Un graphe non orienté est un ensemble des sommets reliés par des arêtes. Chaque arête relié les deux sommets qui sont adjacentes.
  - Les arêtes ne sont pas orientées.
  - Les adjacents sont deux sommets reliés par ou moins une arête.
  - Le nombre des sommets dans le graphe est représenté l'ordre de graphe.

la figure ci-dessous illuster la présentation de graphe orienté et graphe non orienté

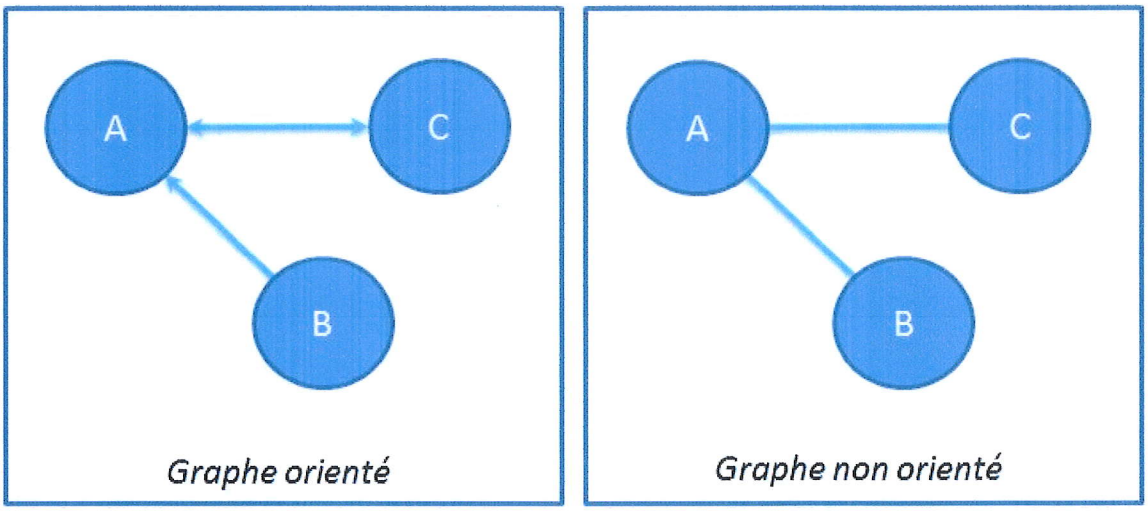


FIGURE 1.9 – Les deux types de graphe [26].

- **Graphe orienté :**En donnant un sens aux arêtes d'un graphe (fléchée) on parle alors de l'extrémité initiale et l'extrémité finale, un arc est défini par une paire ordonnée de sommets.
  - **Graphe étiqueté :**C'est un graphe orienté où chaque arc possède une étiquette.
  - **Graphe pondéré :**C'est un graphe étiqueté où les étiquettes sont des nombres positifs.
  - **Graphe attribué :**C'est un graphe qui peut comporter un ensemble prédéfini d'attributs.

1.6.2 Grammaire de Graphes

Les grammaires de graphes, d'un certain point de vue, sont une généralisation des grammaires de Chomsky appliquées aux graphes. La transformation de graphe est spécifiée sous forme d'un modèle de grammaires de graphes. Ces grammaires sont composées de règles dont chacune est constituée de deux parties : une partie gauche appelée **LHS** (Left Hand Side) qui correspond à un graphe, et une partie droite appelée **RHS** (Right Hand Side) qui correspond aussi à un graphe. Le principe est de modifier des graphes en se basant sur les règles (modification des graphes basée règle) comme le montre la figure suivante :

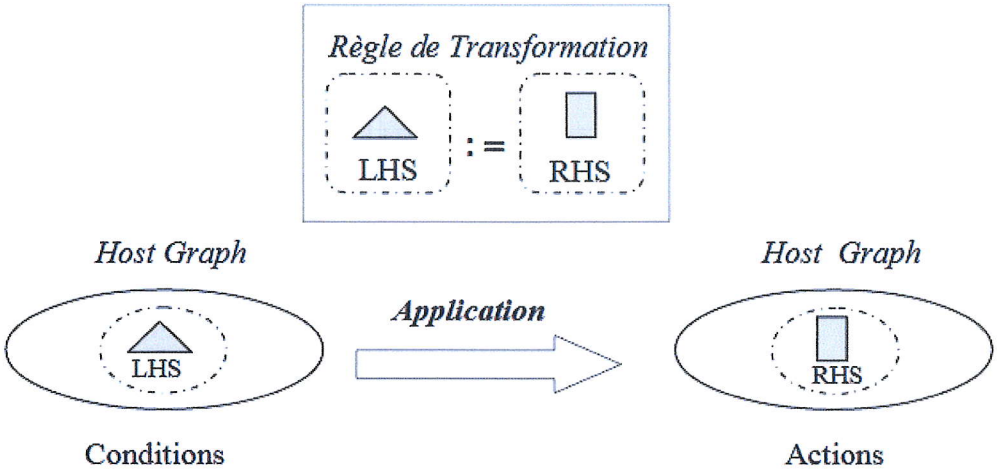


FIGURE 1.10 – Principe de l'application d'une règle de transformation [8].

L'exécution d'une grammaire de graphes est un traitement itératif. A chaque itération, le système de réécriture doit :

- Identifier la règle la plus prioritaire, c'est-à-dire la règle pour laquelle une occurrence du LHS est localisée dans le host graph.
- Evaluer les conditions d'application de cette règle.
- Si les conditions sont vérifiées :
  - Retirer l'occurrence repérée du host graph ainsi que les arcs pendillés.
  - Coller le RHS dans le host graph. Ainsi, les règles sont appliquées sur le host graph dans l'ordre spécifié jusqu'à ce qu'aucune règle ne puisse être applicable [25].

1.7 Outils de transformations de graphes

Il existe plusieurs outils implantant les systèmes de réécriture de graphes. Les plus intéressants sont :

- **Fujaba :** From UML to Java and back gain.
- **AGG :** The Attributed Graph Grammar System.
- **GreAT :** The Graph Rewrite And Transformation tool suite.

— **Viatra : VI**sual **A**utomated model **TRA**nsformations.

— **ATOM³ : A T**ool for **M**ulti-formalism and Meta-Modelling.

— **BOOGGIE : B**rings **O**bject-**O**riented **G**raph **G**rammars **I**nto **E**ngineering.

Et d'autre outils tels que : **DiaGen**, **GenGED**, **PROGRES**, **GrGen.NET**, **MoTMoT**, **GROOVE**, etc.

➢ **L'outil ATOM³ :** est un outil de modélisation multi-paradigme développé par le laboratoire MSDL (Modelling, Simulation and Design Lab) de l'école "computer science" de l'université McGill Montréal au Canada. ATOM³ est un outil visuel dédié à la transformation de graphes, implémenté en langage Python. Il possède une couche de Méta-Modélisation permettant une spécification syntaxique et graphique des formalismes utilisés. Les manipulations de modèles souhaitées sont définies sous forme de grammaires de graphes [23]. ATOM³ signifie "A Tool for Multiformalism and Meta-Modelling". Deux tâches principales d'ATOM³ : la méta-modélisation et la transformation de modèle. La méta-modalisation représente la description ou la modélisation de différents types de formalisme utilisés pour modéliser les systèmes. Malgré qu'on se soit concentré sur les formalismes pour la simulation des systèmes dynamiques, les capacités d'AToM3 ne sont pas limitées à ceux-ci[29]. La transformation de modèle désigne le processus (automatique) de conversion, traduction ou modification d'un modèle dans un formalisme donné en un autre modèle qui pourrait ou ne pas être dans le même formalisme. La figure 1.14 représente l'interface de l'éditeur ATOM³.

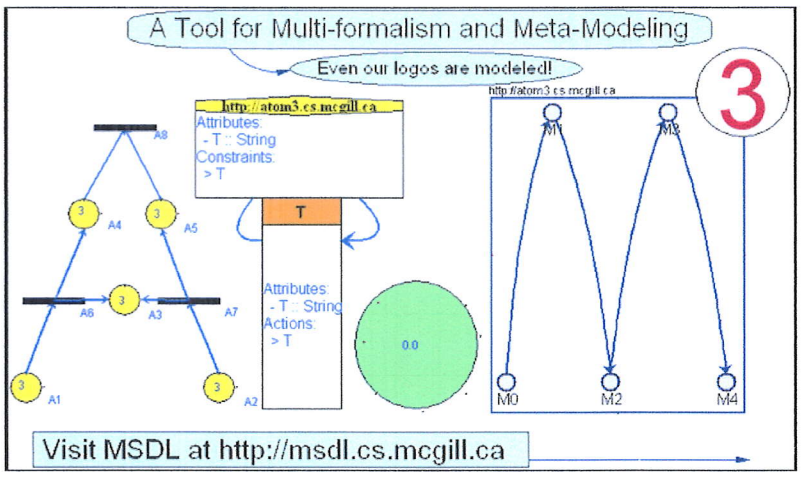


FIGURE 1.11 – L'outil d'ATOM³.

Dans notre approche nous utilisons le formalisme "CD ClassDiagramsV3", la figure suivante représente l'interface d'ATOM³ ce formalisme chargé :

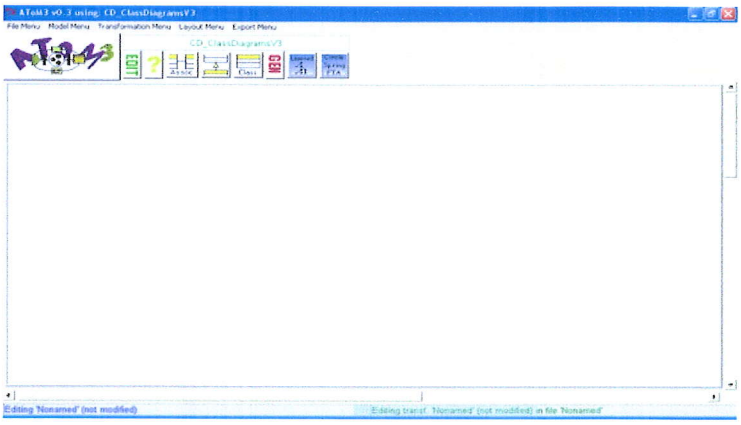


FIGURE 1.12 – Le canevas d'ATOM³.

1.8 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.
Dans le dernier chapitre nous présenterons l'approche de cette transformation.

CHAPITRE 2

Le diagramme global d'interaction d'UML

2.1 Introduction

L'approche orientée objet nécessite une conception avant la modélisation car la modélisation facilite la compréhension des logiciels, tout en offrant de la flexibilité au système et en permettant la comparaison des solutions de conception avant le développement. Les langages de modélisation sont chargés de surmonter les contraintes liées aux langages d'implémentation. UML (Unified Modeling Language) est aujourd'hui le langage de modélisation d'applications informatiques le plus important du marché. Il est supporté par la quasi-totalité des outils de développement, lesquels permettent l'édition de modèles UML et offrent des capacités telles que la génération de code, de test et de documentation, le suivi d'exigences ou encore le Reverse Engineering. Au niveau d'Unified Modeling Langage, nous notons deux termes : unified et langage. Le terme unified montre que les auteurs ont essayé de rassembler les éléments principaux des concepts objets, alors le terme langage signifie que c'est un langage de modélisation. Dans notre travail, nous avons appliqué la modélisation en utilisant les diagrammes UML 2.0.

Dans ce chapitre nous évoquons les notions fondamentales du diagramme global d'interaction et ses différentes composants, les relations entre sec composants, le diagramme de communication et enfin un exemple sur les diagramme global d'interaction.

2.2 Les diagrammes d'UML 2.0

UML est un langage de modélisation graphique à base de pictogrammes utilisé dans le développement logiciel. Ce langage de schémas qui découle d'une harmonisation des méthodes d'analyse utilisées dans les années 1990, est aussi bien prévu pour être dessiné par des logiciels graphiques, que tracé sommairement et surtout rapidement au crayon sur une feuille blanche.[2]

UM L 2.0 propose treize types de diagrammes pour représenter les différents points de vues distinctes pour modéliser des concepts particuliers d'un système.[3] Ils se répartissent en deux grandes catégories :

- ☞ Diagrammes structurels ou statiques d'un système (Structure Diagram).
- ☞ Diagrammes comportementaux ou dynamiques d'un système (Behavior Diagram).
- ☞ Diagramme d'interaction ou dynamiques (Interaction diagrams).

2.2.1 Diagrammes structurels (statiques)

Structure diagrammes définir l'architecture statique d'un modèle. Ils sont utilisés pour modéliser 'les choses' qui composent un modèle - les classes, des objets, des interfaces et des composants physiques. En outre, ils sont utilisés pour modéliser les relations et les dépendances entre les éléments.[5]

Les diagramme UML structurels sont : Diagrammes de Classes, Diagrammes d'Objets, Diagrammes de Composants, Diagrammes de Déploiement, Diagrammes de Paquetage, Diagrammes de Structures Composites Diagrammes

2.2.2 Comportementaux (dynamiques)

Les diagrammes de comportement permettent de spécifier le comportement des éléments d'un système, et leur dynamique. On les utilise notamment pour modéliser des processus et des événements chronologiques. Ces diagrammes sont :

- ✦ **Diagrammes de Cas d'Utilisation** (Use Case Diagram)

- **Diagrammes d'états-transition** (State Machine Diagram)

- **Diagrammes d'Activité** (Activity diagram)

2.2.3 Diagrammes d'interaction

Les diagrammes d'interaction permettent de montrer l'interactivité entre les différents acteurs/utilisateurs et le système. Ces diagrammes sont :

- **Diagramme de séquence** (Sequence Diagram)

- **Diagramme de communication** (Communication Diagram)

- **Diagramme de temps**(Timing Diagram)

- **Diagramme global dinteraction** (Interaction Overview Diagram)

Les définitions dans cette section peuvent être dérivées principalement de [5], [6], [7]. L'axe principal de notre travail c'est le diagramme global d'interaction (Interaction Overview Diagram). A cet effet, il est indispensable de lui consacrer plus d'espace dans ce chapitre.

2.3 Le diagramme global d'interaction (I.O.D)

2.3.1 Présentation

Le diagramme d'IOD peut intégrer des diagrammes de séquence, des diagrammes de communication ou des diagrammes de temps. Le diagramme global d'interaction apparut dans la version 2.0 d'UML, qui permet de représenter un flux de contrôle avec des nœuds pouvant contenir des diagrammes d'interaction montrant comment un ensemble de fragments peut être initié dans différents scénarios. Les diagrammes de vue d'ensemble des interactions se concentrent sur la vue d'ensemble du flux de contrôle où les nœuds sont des diagrammes de communication(cd) si nous allons nous intéresser à la communication entre les objets du

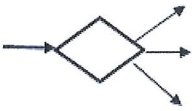
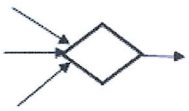
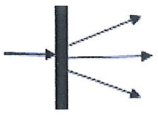
système, un diagramme de séquence(sd) dans le cas où on s'intéresse à l'ordre des messages ou bien une utilisation d'interaction (ref). Les autres éléments de notation pour les diagrammes de synthèse des interactions sont les mêmes que pour les diagrammes d'activité et de communication. Ceux-ci incluent les nœuds initiaux, final, décision, fusion (merge), fork et join. Le diagramme global d'interaction est un diagramme comportemental qui permet de spécifier d'une manière hiérarchique le comportement du système.

2.3.2 Les composants du diagramme global d'interaction

L'IOD est un type particulier de diagramme d'activités où les nœuds sont des interactions et les liens entre nœuds se réduisent à un flot de contrôle. [9]
Les éléments d'un diagramme global d'interaction sont :

- ➢ **Les Nœuds (Nodes) :** Il existe deux types de nœuds :
  - ✓ **Nœuds de contrôle :** Un nœud de contrôle est un nœud d'activité abstrait utilisé pour coordonner les flots entre les nœuds d'une activité. Il existe plusieurs types de nœuds de contrôle.

Le tableau suivant illustre les différents types des nœuds de contrôle :

Elément	Notation	Description
Nœud initial (initial node)		C'est un nœud de contrôle à partir duquel le flot débute lorsque l'activité enveloppante est invoquée. Il ne possède aucun arc entrant mais il dispose d'un arc sortant. Dans un diagramme global d'interaction, il existe un seul état initial.
Nœud final (final node)		C'est un nœud de contrôle dans lequel le flux de contrôle s'arrête. Un nœud de contrôle possédant un ou plusieurs arcs entrants et aucun arc sortant. Un nœud final représenté par un cercle contenant un petit cercle plein.
Nœud de décision (decision node)		Les arcs sortants sont sélectionnés en fonction de la condition de garde qui est associée à chaque arc sortant (Possibilité d'existence du problème du choix indéterministe). Si aucun arc en sortie n'est franchissable, le modèle est mal formé, et l'utilisation d'une garde **[else]** est recommandée.
Nœud fusion (merge node)		Un nœud de fusion est un nœud de contrôle rassemblant plusieurs flots alternatifs entrants en un seul flot sortant. merge node est représenté par un losange comme le nœud de décision.
Nœud de bifurcation (fork node)		C'est un nœud de contrôle qui sépare un flux d'entrée en plusieurs flots concurrents en sortie.

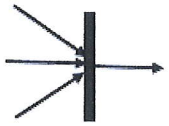
Nœud d'union (join node)		Est un nœud de contrôle qui synchronise des flots multiples. Il possède plusieurs arcs entrants et un seul arc sortant. Lorsque tous les arcs entrants sont activés, l'arc sortant l'est aussi également.

TABLE 2.1 – Les nœuds de contrôle [1].

✓ **Nœuds d'interaction :**

❖ **Interaction**"Inline" : peut être un SD dans le diagramme de séquence et un CD dans le diagramme de communication, dans notre travail on utilise diagramme de communication alors nud d'interaction inline est un CD.

☛ **Interaction**"CD" (Communication Diagram Interaction) : est une unité de comportements qui se concentre sur les échanges d'informations observables entre les éléments connectables. Elle est représentée par un diagramme d'interaction de communication.

❖ **Interaction**"use" : Une interaction "use" fait référence à un diagramme d'interaction existant (diagramme de communication). Elle permet de copier le contenu de l'interaction référencée en prenant en considération la substitution des paramètres par les arguments.

Le tableau suivant représente la notation des nœud d'interaction(use et CD).

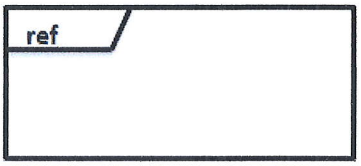
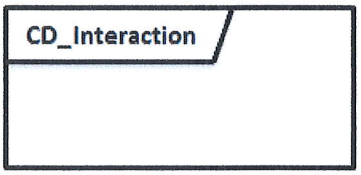
	Notation
Interaction Use	
Interaction CD	

TABLE 2.2 – Les notations des nœuds d'interaction.

➢ **Les Arcs :** (edges)

✓ **Flot de contrôle (control flow) :** Un flot de contrôle est un arc qui décrit le séquencement de deux nœuds d'interaction (un flot de contrôle permet le démarrage d'un nœud d'interaction, après la terminaison d'une interaction précédente). Les données ne peuvent pas être transmises par cet arc.

La figure suivante représente l'apparence graphique d'un arc.

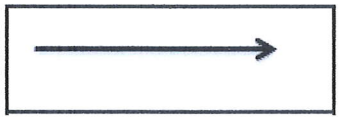


FIGURE 2.1 – Notation arc

2.3.3 Relations entre les éléments du diagramme global d'interaction

Dans un diagramme d'interaction nous pouvons observer des relations entre les nœuds de contrôle, des relations entre les nœuds d'interaction et des relations entre les nœuds de contrôle avec des nœuds d'interaction.

Nous illustrerons ces relations entre les n œd dans la figure ci-dessous.

Relation ↱			*Nœuds de contrôle*						**Nœud d'interaction**	
			Nœud initial	Nœud final	**Abstract Join Node**		**Abstract Split Node**		Interaction Use	Interaction CD
					Merge	Joint	Decision	Fork		
Nœud de contrôle	Nœud initial							√	√	√
	Nœud final									
	Abstract Join Node	Merge		√		√	√	√	√	√
		Joint		√	√		√	√	√	√
	Abstract Split Node	Decision		√	√	√		√	√	√
		Fork			√	√	√		√	√
Nœud d'interaction	Interaction Use			√	√	√	√	√	√	√
	Interaction CD			√	√	√	√	√	√	√

√ :Relation possible
■ : Relation impossible

FIGURE 2.2 – Illustration des relations entre les éléments des IODs.

2.4 Diagramme de Communication

Les diagrammes de communication sont des diagrammes d'interaction que vous pouvez utiliser pour explorer le comportement dynamique d'un système ou d'une application logicielle.[22] Ce diagramme s'appelait diagramme de collaboration et avait une notation différente. En effet, le diagramme de communication met plus l'accent sur l'aspect spatial des échanges que l'aspect temporel. Chaque objet intervient dans ce diagramme de la même façon que dans le diagramme de séquence. Il n'est pas associé à une ligne de vie mais relié graphiquement aux objets avec lesquels il interagit.

Les envois de messages sont placés le long des liens inter-objets. Les messages sont obligatoirement numérotés, la numérotation composée étudiée dans le cadre des

diagrammes de séquence pouvant également être employée.

2.4.1 Les Composants du Diagramme de Communication

➢ **Rôle**(Objet) : Chaque participant à un échange de message correspondant à une ligne de vie dans le diagramme de séquence se représente sous forme d'un rôle dans le diagramme de communication.

➢ **Liens :** Indique un chemin de communication entre deux objets, sur lequel passent les messages.

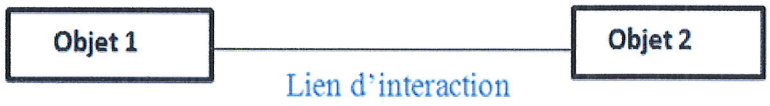


FIGURE 2.3 – liens entre les Objet

➢ **Message :** Un message est un élément de diagramme Unified Modeling Language (UML) qui définit un type particulier de communication entre les instances au cours d'une interaction. Un message fait circuler des informations d'une instance. Dans le diagramme de communication existe quatre types du message, représentent par La figure 2.4

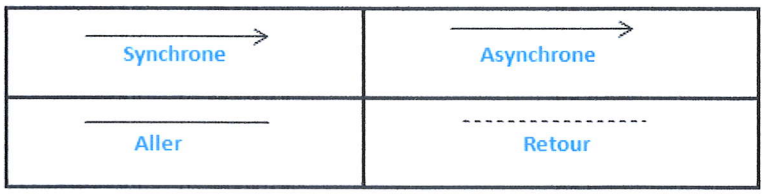


FIGURE 2.4 – Les types des messages

✓ **Synchrone :** l'émetteur reste en attends jusqu'à ce que la réponse arrive.

✓ **Asynchrone :** Un message est envoyé par à un objet à un autre, mais le premier objet n'attends pas la fin de l'action.

✓ **Aller(plat) :** Chaque flèche représente une progression d'une étape à une autre dans la séquence, La plupart asynchrone.

✓ **Retour :** Le retour explicite d'un objet à qui le message était envoyé.

Un diagramme de communication montre des acteurs, des objets (instances de classes) et leurs liens de communication (appelés liens entre objets), ainsi que les messages qu'ils échangent. Les messages sont définis sur des liens entre objets qui correspondent à un lien de communication entre deux objets qui interagissent. L'ordre dans lequel les messages sont échangés est représenté par les numéros d'ordre.[14] La figure suivante illustre les différents composants d'un diagramme de communication.

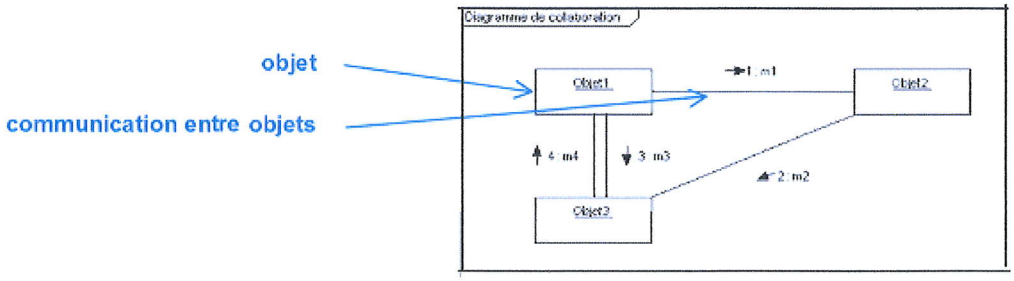


FIGURE 2.5 – Diagramme de communication[15].

2.4.2 Exemple de diagramme de communication

La figure ci-dessous représente un exemple de diagramme de communication qui est illustrer la recherche puis l'ajout, dans son panier virtuel, d'un livre lors d'une commande sur Internet.

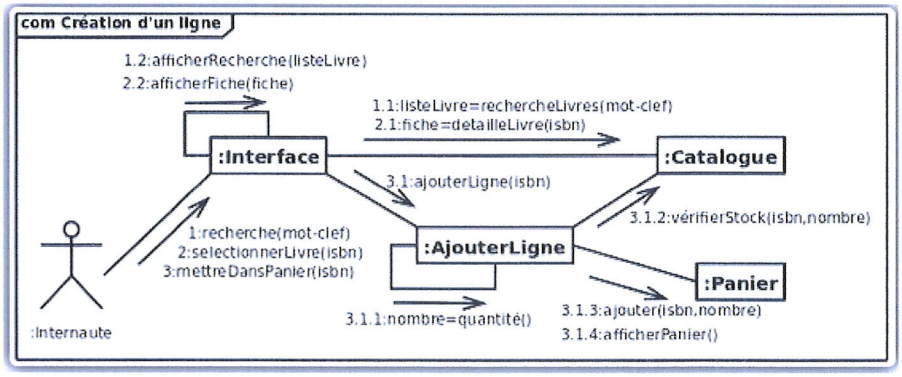


FIGURE 2.6 – Diagramme de communication illustrant la recherche puis l'ajout, dans son panier virtuel, d'un livre lors d'une commande sur Internet[19].

2.5 Exemple d'un IOD

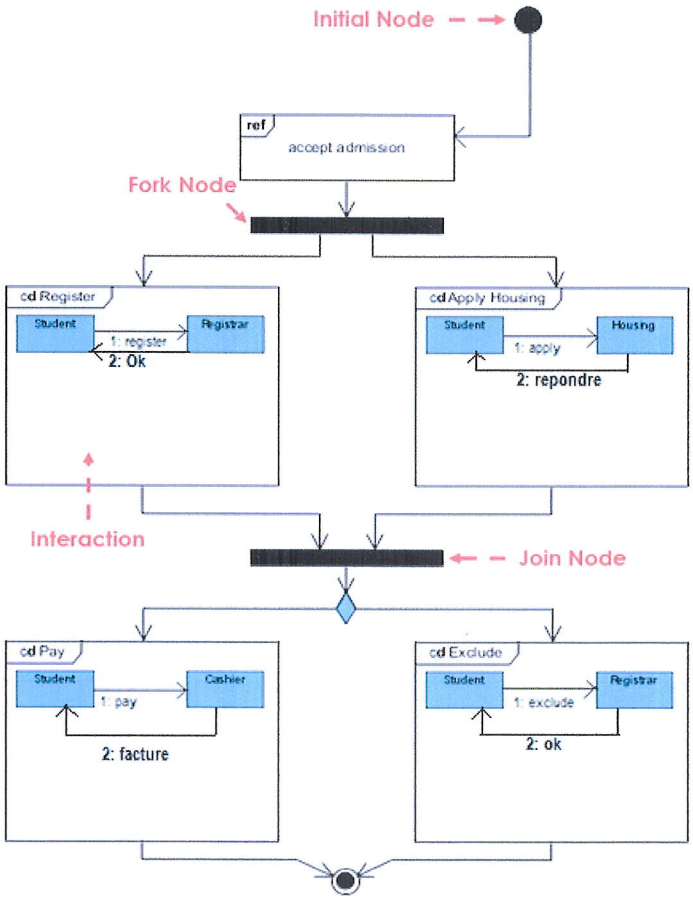


FIGURE 2.7 – Exemple de diagramme d'interaction[20].

Cette figure est représent un exemple de diagrammes d'interaction, L'exemple ci-dessus montre un étudiant qui a été accepté dans une université. Tout d'abord, l'étudiant doit être accepté l'admission. Après avoir accepté, l'étudiant doit à la fois s'inscrire aux cours et faire une demande de logement. Une fois que les deux sont terminés, l'étudiant doit payer le registraire. Si le paiement n'est pas reçu à temps, le registraire exclut l'étudiant. Il est également possible de remplacer chaque référence ref par un diagramme de communication pour accepter l'admission.

2.6 Conclusion

Nous avons présenté dans ce chapitre la classification des diagrammes d'UML. Nous nous sommes concentrés sur le diagramme global d'interaction (Interaction Overview Diagram), représentation, ses composantes ainsi que les relations entre ses dernières. IOD propose d'associer les notations du diagramme de communication et leur représentation, ses composants. Enfin, un exemple qui explique la structure d'un diagramme de communication. On peut conclure d'après ce que nous avons vu dans ce chapitre, le diagramme global d'interaction peut être aussi bien utilisé en phase d'analyse qu'en phase de conception pour la description d'une méthode complexe.

CHAPITRE 3

Logique de Réécriture et Maude

3.1 Introduction

La réécriture est un paradigme général d'expression du calcul dans des diverses logiques computationnelles. Le calcul prend la forme de règles de réécriture dans une syntaxe donnée. Dans la logique de réécriture, une réécriture d'un terme consiste à le remplacer par un terme équivalent, conformément aux lois d'algèbre de termes. Cette logique a été présentée par José Meseguer, comme une conséquence de son travail sur les logiques générales pour décrire les systèmes concurrents. La logique de réécriture présente un modèle unifié de concurrence dans laquelle plusieurs modèles bien connus des systèmes concurrents peuvent être représentés dans une structure commune.

Dans ce domaine, le langage Maude est l'un des langages les plus utilisés. C'est un langage très puissant permettant la spécification formelle des systèmes concurrents et la vérification des propriétés. Parmi ses particularités est qu'il est simple, riche et déclaratif. Elle a pour but de modéliser de manière mathématique des systèmes informatiques, et cela en présentant un formalisme unifié de concurrence dans lequel plusieurs modèles bien connus des systèmes concurrents sont intégrés (comme les réseaux de pétri, les systèmes de transitions étiquetées, la machine chimique abstraite et bien d'autres).

Nous allons tout d'abord aborder dans ce chapitre la logique de réécriture et ses principes, pour ensuite introduire une description du langage Maude, de ses éléments et de ses fondements mathématiques. Finalement, nous terminerons ce

chapitre avec une conclusion qui résumera l'utilisation de la logique de réécriture dans le langage Maude.

3.2 La logique de réécriture

La logique de réécriture est un cadre logique flexible et expressif qui unifie la sémantique algébrique dénotationnelle et structurelle La logique de réécriture est dotée d'une sémantique saine et complète, elle unifie tous les modèles formels qui expriment la concurrence. [32]

La réécriture est un modèle de calcul utilisé en informatique, en algèbre, en logique mathématique. Il s'agit de transformer des objets syntaxiques (mots, termes, programmes, logique prend la forme de règles de réécriture dans une syntaxe donnée. Dans la logique de réécriture, une réécriture d'un terme consiste à le remplacer par un terme équivalent, conformément aux lois d'algèbre de termes. [31]

L'avantage essentiel de la logique de réécriture est que les spécifications réalisées peuvent être efficacement exécutées, cela offre un moyen puissant de prototypage et d'exécution. En plus, l'interopérabilité de ces logiques et modèles devient possible grâce à leurs représentations homogènes en logique de réécriture. [33]

3.3 Définition formelle

Une théorie de réécriture $\mathfrak{R} = (\Sigma, E, L, R)$ est un 4-tuple et elle est vue comme une spécification exécutable du système concurrent quelle formalise.

Dans la logique de réécriture, les systèmes concurrents sont représentés par une théorie de réécriture $\mathfrak{R} = (\Sigma, E, L, R)$. Ou la structure statique est décrite par la signature (Σ, E), tandis que la structure dynamique est décrite par les règles de réécriture R.[34] tel que (Σ, E) est une signature, Σ un ensemble de sortes et d'opérateurs et E un ensemble de Σ -équations. La signature (Σ, E) est une théorie équationnelle qui décrit la structure algébrique des états du système.

L'ensemble $R \subseteq L \times (T\Sigma, E(X))^2$ est l'ensemble des pairs tel que le premier composant est une étiquette et le second est un pair des classes l'équivalence des termes modulo les équations E, avec X= {x1,...,xn,...}un ensemble comptable et infini des variables. [34]

La structure dynamique est décrite par les règles de réécriture $rl\ [l] : t \to t'$, ces règles représentent les transitions élémentaires et locales dans un système concurrent.

➢ **Les règles de déduction** :Etant donnée une théorie de réécriture R, nous disons qu'une règle $[t] \to [t']$ est prouvable dans R où R implique $[t] \to [t']$ si et seulement si $[t] \to [t']$ peut être obtenue par une application finie des règles de déduction suivantes :

✓ **Réflexivité** :pour chaque $[t] \in T_{\Sigma,E}(X)$,

$$\frac{}{[t] \to [t]}$$

✓ **Congruence** :pour chaque $f \in \Sigma_n, n \in N$,

$$\frac{[t_1] \to [t'_1] \ldots\ldots\ldots\ldots [t_n] \to [t'_n]}{[f(t_1,\ldots,t_n)] \to [f(t'_1,\ldots,t'_n)]}$$

✓ **Remplacement** : pour chaque règle de réécriture

$r : [t(x_1, x_2, \ldots, x_n)] \to [t'(x_1, x_2, \ldots, x_n)]$ dans R, $\dfrac{[w_1] \to [w'_1] \ldots\ldots\ldots [w_n] \to [w'_n]}{[t\,(\overline{w}/\overline{x})] \to [t'(\overline{w}'/\overline{x})]}$

tel que t(w/ x) indique la substitution simultanée des w_i pour x_i dans t.

✓ **Transitivité** :

$$\frac{[t_1] \to [t_2] \quad [t_2] \to [t_3]}{[t_1] \to [t_3]}$$

✓ **Symétrie** :

$$\frac{[t_1] \to [t_2]}{[t_2] \to [t_1]}$$

[9]

Nous pouvons visualiser graphiquement ces règles de déduction comme suit

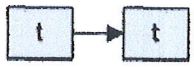
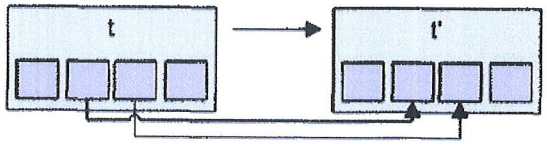
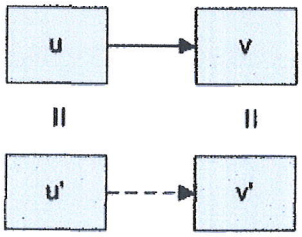
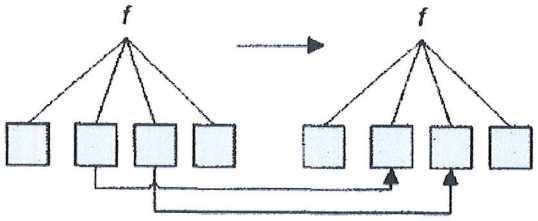
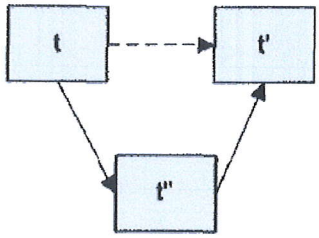
Les règles de déduction	Représentation graphique
Réflexivité	
Remplacement	
Égalité	
Congruence	
Transitivité	

TABLE 3.1 – Représentation graphique des règles de déduction.
[35]

➢ **Le modèle sémantique d'une théorie de réécriture :** Les catégories

sont le modèle mathématique associé aux théories de réécriture, celui-ci peut supporter les termes générés par la spécification algébrique (Σ, E) et les preuves générées par la déduction. Les objets de cette catégorie sont les Σ-termes et les morphismes représentent les preuves entre eux. Nous pouvons justifier l'utilisation des catégories par l'importance des morphismes car ils peuvent faire une abstraction des détails d'implémentation des objets, et donc l'accent sera mis sur les relations qui existent entre ces objets.

3.4 Le Langage Maude

3.4.1 Présentation de Maude

Maude est un langage déclaratif de haute performance, destiné à la modélisation de diverses notions relevant de plusieurs domaines. Ce langage, basé sur la logique de réécriture vue précédemment, a été développé en fonction de trois grands objectifs : la simplicité, l'expressivité et la performance. [35]

- **La simplicité :** Un programme doit être le plus simple possible et aussi facile à comprendre. Il doit également posséder une sémantique très claire.[34]
- **La performance :** Il devrait être possible d'utiliser un langage comme spécification exécutable d'un modèle, mais également comme un véritable langage de programmation compétitif. [35]
- **L'expressivité :**Il est possible d'exprimer naturellement une vaste gamme d'applications. Il est aussi possible de le faire autant pour un programme déterministe que pour un programme hautement concurrentiel. [34]

3.4.2 Les concepts de Maude

Dans cette section, nous présentons les notions syntaxiques les plus intéressantes du langage Maude (les sortes, les sous sortes, les opérations, etc.).

- **Modules :**dans Maude, les unités de base de spécification et de la programmation s'appellent les modules.[34] Maude regroupe trois types de modules :
  - **Les Modules fonctionnels :** Les modules fonctionnels définissent des types de données et des opérations sur ces données par le biais des théories

équationnelles.[31] Un module fonctionnel est déclaré avec la syntaxe suivante :

fmod (*Nom du Module*) **is** (*Declaration liste d'instructions*) **endfm**

- **Les Modules systèmes :** Un module système indique une théorie de réécriture .une théorie dont laquelle on trouve des sortes, des opérateurs (peut-être avec des arguments gelés), et peut avoir plusieurs types de déclarations : des équations, et des règles, dont toutes peuvent être conditionnelles.

  Un module système est déclaré dans Maude en employant les mots-clés :

  mod (*Nom du Module*) **is** (*Declaration liste d'instructions*) **endm**

- **Les Modules Objet :**Bien que les modules système de Maude soient suffisants pour la spécification des systèmes orientés objets. Utilisant une syntaxe plus pratique que celle des modules systèmes, tous les modules orientés objet incluent implicitement le module CONFIGURATION +. Les modules orientés objets sont transformés en des modules systèmes pour des buts d'exécution. [31]

  Un module objet est déclaré par la syntaxe suivante :

  omod (*Nom du Module*) **is** (*Declaration liste d'instructions*) ***endom***

- **la réutilisation des modules (Importation) :** La plupart des langages de programmation offrent au développeur la possibilité d'importer d'autres modules, cette façon de spécification permet d'inclure toutes les déclarations et définitions des modules importés, par conséquent cela permet de minimiser la redondance dans la spécification mais aussi permet d'offrir une meilleure modularité.[36] MAUDE utilise les mots clé suivants **"Protecting"**, **"Extending "**, ou **"Including"**, pour importer un module.

La syntaxe d'utilisation de ces trois clauses est comme suit :

"Protecting" Nom_ Du_ Module .

"Including" Nom_ Du_ Module .

"Extending " Nom_ Du_ Module .

➦ **Protecting :** dans ce mode d'importation, les éléments déclarés dans le module importés n'accepte pas la modification.

➦ **Including :** le développeur peut changer le sens des éléments déclarés dans le module importé.

➦ **Extending :**dans ce mode autorisé l'ajout de nouveaux termes à une sorte, mais il ne peut pas modifier la signification des opérations.

➢ **La Déclaration :**

◆ **Les Sortes et Sous-Sortes :** La première chose à déclarer dans une spécification est l'ensemble de nouveaux types de données utilisés, dans la communauté de spécification algébrique ces types sont appelés sorte.[34] Une sorte est déclarée en utilisant le mot clé sort suivi par le nom de la sorte comme suit :

sort (le nom de la Sorte) .

La déclaration de plusieurs sortes peut être déclarée en suivant la forme :

sorts < Sort-1>... <Sort-K>.

Par exemple nous pouvons déclarer les sortes Nat, Zero et NzNat comme illustre dans la figure 3.1 :

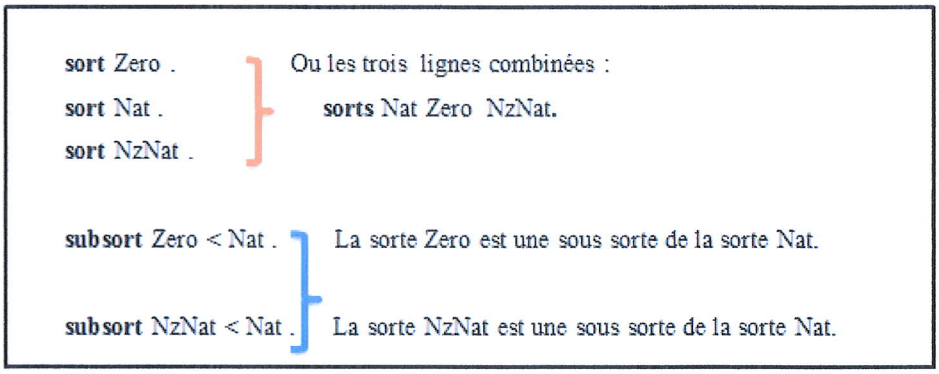


FIGURE 3.1 – Déclaration de sorte et de sous sorte.

☛ **Remarque :** Le point (.) à la fin de la déclaration des sortes, comme pour les autres déclarations, est important.

◆ **Les Opérations :** Dans le langage Maude une opération est déclarée à l'aide du mot clé op suivi par le nom de l'opération, suivi par deux points, suivi par la liste des sortes arguments de l'opération (appelé le domaine), suivi par le symbole à suivi par la sorte du résultat de l'opération et optionnellement suivi par la déclaration des attributs de l'opération suivi par un espace et un point, donc le schéma général est de la forme :34

```
op (Op Name) : (Sorte-1) ... (Sorte-k) -> (Sorte) .
```

Dans la figure suivante nous présentons un exemple de déclaration de quelques opérations en Maude.

```
op zero : -> Zero .
op s_ : Nat -> Nat .
op _+_ : Nat Nat -> Nat .
op _+_ : Int Int -> Int .
ops _+_ _*_ : Nat Nat -> Nat .   } Plusieurs opérations ayant la même déclaration
```

FIGURE 3.2 – Déclaration des opérations.

☛ **Remarque :** Si la liste des arguments est vide, l'opération est une constante.

◆ **Les Attributs des opérations :** Le langage Maude permet la déclaration d'un ensemble d'attributs pour les opérations, ces attributs sont introduits entre crochet après la sorte résultat de l'opération et avant le point final. Le rôle essentiel de ces attributs est d'offrir des informations : syntaxiques, sémantiques, pragmatiques additionnelles sur les opérations.[36]

Le tableau suivant représente quelque attribut avec leur description :

Attributs	Descriptions
ctor	pour exprimer que l'opération est constructeur de la sorte résultat.
assoc	pour exprimer l'associativité d'une opération.
comm	pour représenter la commutativité.
id : <term>	pour énoncer que le terme <term> est l'élément neutre de l'opération.

TABLE 3.2 – Les attributs des opérations dans Maude.
[35]

```
sorts List Elt .
subsort Elt < List .
 op nil : -> List [ctor] .
/*signifie que nil est une opération constructeur du type algébrique List */
```

FIGURE 3.3 – Attributs des opérations.

◆ **Les variables :** Une variable doit obligatoirement avoir une sorte particulière. Elle est déclarée dans Maude avec le mot clé var (ou vars pour plusieurs variables).[31] comme suit :

var X : Nat .

var Y : Nat .

Ou : **vars** X Y : Nat .

◆ **Les termes :** Un terme est une constante, une variable, ou bien l'application d'une opération à une liste de termes d'argument. Le type d'une constante ou variable est sa sorte déclarée. Dans l'application d'une opération, la liste d'arguments doit être en accord avec l'arité déclarée de l'opération.[31]

par exemple :

s_(zero)

_+_(X :Nat, Y :Nat)

3.4.3 Les niveaux de Maude

Le système Maude inclus deux niveaux principaux bf Core Maude et bf Full Maude.

➢ **Core Maude :** C'est une interprétation implémenté en C++. Core Maude représente le niveau de base de Maude. Les unités de base dans Core Maude de spécification sont appelées modules : les modules fonctionnels, les modules systèmes et autre modules intègre par Core Maude : les modules prédéfinis des types de données et le module Model-Checker.

➢ **Full Maude :** C'est une extension du premier niveau Core Maude. les concepts exprimer en Core Maude sont exprimer en Full Maude mais respecter quelques restrictions syntaxique. modules fonctionnels et systèmes, les modules orientés objet sont présents dans Full Maude.

3.4.4 La programmation paramétrée sous Maude

Dans Full Maude, nous pouvons utiliser en plus des mots-clés protecting, extending et including, pour définir des spécifications structurées, le concept puissant de la programmation paramétrée.[31]
Dans ce contexte la matière de base de la programmation paramétrée est constituée des théories, des modules paramétrés et des vues.

➤ **Les théories :** Les théories sont utilisées pour déclarer l'interface d'un module paramétré, quant aux types de modules, Core Maude supporte deux différents types de théories : les théories fonctionnelles et les théories systèmes avec la même structure du module correspondant.[36]
La déclaration **fth... endfth, th... endth** et **oth... endoth**, respectivement.

➤ **Les modules paramétrés :** La forme syntaxique d'un module paramétré est : $\mathbf{M}$ $\mathbf{(X1 :: T1 \mid \ldots \mid Xn :: Tn)}$

où **M** est le nom du module paramètre, **X1 . . . Xn** sont les étiquettes et **T1 . . . Tn** sont respectivement les théories paramètre, et $\mathbf{(X1 :: T1 \mid \ldots \mid Xn :: Tn)}$ est son interface. Dans Full Maude toutes les

sortes parvenant des théories de l'interface doivent être qualifiés par leurs étiquettes, même s'il n'y a pas d'ambiguïté. Si Z est l'étiquette d'une théorie de paramètre T, donc chaque sorte S dans T doit être qualifié comme Z@S. il ne peut pas y avoir la surcharge des sous sortes entre un opérateur déclaré dans une théorie étant utilisée comme le paramètre d'un module paramétré et un opérateur déclaré dans le corps du module paramétré, ou entre des opérateurs déclarés dans deux théories paramètre du même module.[31]

➤ **Les vues :** Une vue établit une correspondance entre les sortes et les opérations définies dans la théorie à celles du module paramétré réel donné. une vue dans Core Maude est déclarée en suivant la syntaxe : NomdeVue from **THEORIE to MODULE** is... endv.

3.4.5 L'outil Maude :

Maude possède une librairie standard des modules prédéfinis qui, par défaut, sont chargés par le système automatiquement au début de chaque session. Chacun de ses modules prédéfinis peut être importé par un autre module défini par l'utilisateur. Ces modules prédéfinis se trouvent dans des fichiers au même répertoire que le fichier exécutable de Maude. Parmi ces fichiers, on trouve prelude.maude et model-checker.maude. En fait, le premier fichier contient plusieurs modules prédéfinis. Nous pouvons trouver comme exemples de modules prédéfinis dans ce fichier : BOOL, STRING et NAT. Ces modules déclarent les sortes et les opérations pour manipuler respectivement les valeurs booléennes, les chaînes de caractères et les nombres naturels.[37]
La figure 3.4 représente l'interface d'exécution de Maude.

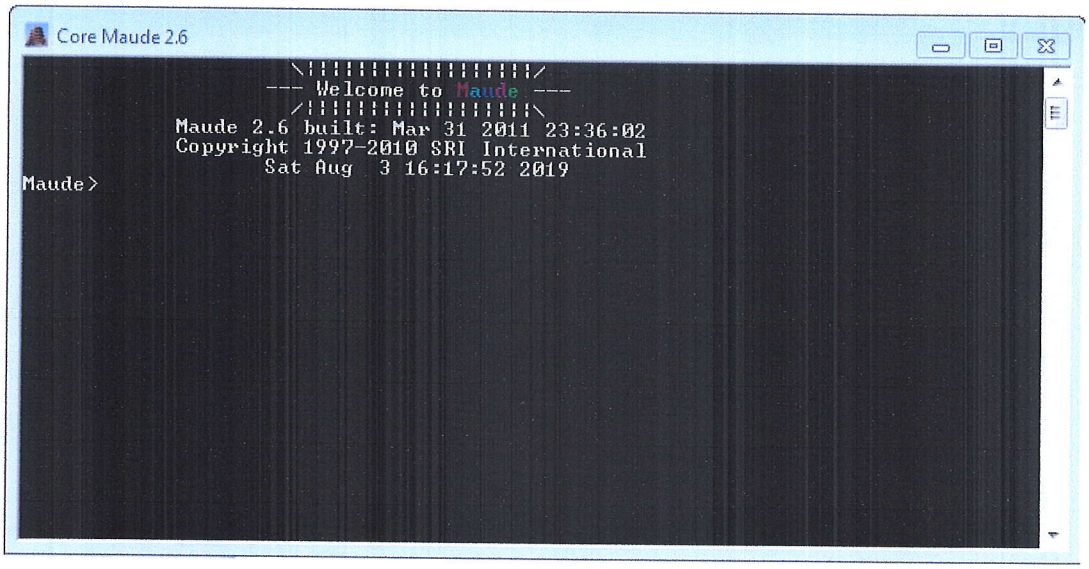


FIGURE 3.4 – Exécution de Maude

3.4.6 Les commandes les plus utilisées dans Maude

Le tableau suivant représente quelques commandes du langage Maude, les plus fréquemment utilisées :

Commande	Description
set trace on	Demande à Maude d'afficher comment il applique les équations.
reduce	Cette commande est utilisée pour évaluer des expressions.
show module	Cette commande, permet à Maude d'afficher le module courant.
show vars	Demande à Maude de Visualiser la liste des variables.
show ops	Cette commande, permet à Maude de visualiser la liste des opérations.

Set trace on	Demande à Maude d'afficher comment il applique les équations.
show rls	Demande à Maude de Visualiser la liste des règles.

TABLE 3.3 – Quelques commandes dans Maude.

3.4.7 Les caractéristiques de Maude

- **Basé sur la logique de réécriture :** exprime le changement d'états du système. Les programmes sont des théories de la logique de réécriture (une signature et un ensemble de règles de réécriture).
- **Multi-paradigme :**combine les programmations fonctionnelle et orientée objet. il inclut en plus un certain nombre de modules prédéfinis.
- **Stratégies internes :** les stratégies servent à guider le processus de réécriture.
- **Maude est aussi utilisé pour :** la spécification, la vérification.
- **Maude est un langage** déclaratif de haut niveau et haute performance.

3.5 Conclusion

Dans ce chapitre nous avons présenté deux concepts importants : la logique de réécriture et le langage Maude que nous avons utilisé pour l'implémentation de notre application.

Nous nous sommes concentrées d'une part sur la logique de réécriture qui est une logique présentée par Jose Meseguer comme le résultat de son travail sur les logiques générales.

Dans une deuxième partie, nous avons donnés les différents concepts de base du langage Maude. Le langage Maude et ses caractéristiques ont été présentés ainsi que les déférents modules du langage, les concepts de base et les niveaux de programmation dans Maude. Enfin on a terminé par présenter quelques commandes les plus utilisées dans la programmation Maude et quelque caractéristique de langage Maude.

Le chapitre qui suit, constituera en une description détaillée de notre application, et l'utilisation de langage Maude dans notre travail.

CHAPITRE 4

Une Approche de transformation de IOD vers Maude

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.

Nous allons voir en détail dans ce chapitre, l'approche que nous avons proposée pour effectuer cette transformation de graphes en utilisant un méta-modèle qui va générer un modèle, puis on appliqueras les règles de transformation de notre grammaire de graphe.

Nous terminerons ce chapitre par un exemple d'étude de cas qui résume toutes les étapes de la transformation de graphe.

4.2 Scénario de la transformation

Notre approche comporte trois phases principales :

1- Méta-Modélisation et génération d'outils :

dans cette phase, nous allons définir le méta-modèle du digramme global d'interaction (IOD), en utilisant l'outil AToM^3.

2- Proposition d'une grammaire :

nous proposons une grammaire de graphe pour automatiser la transformation de notre diagramme vers Maude.

3- Grammaire de graphes :

dans cette étape, la description formelle en langage Maude est générée par l'exécution des règles de transformation de graphes.

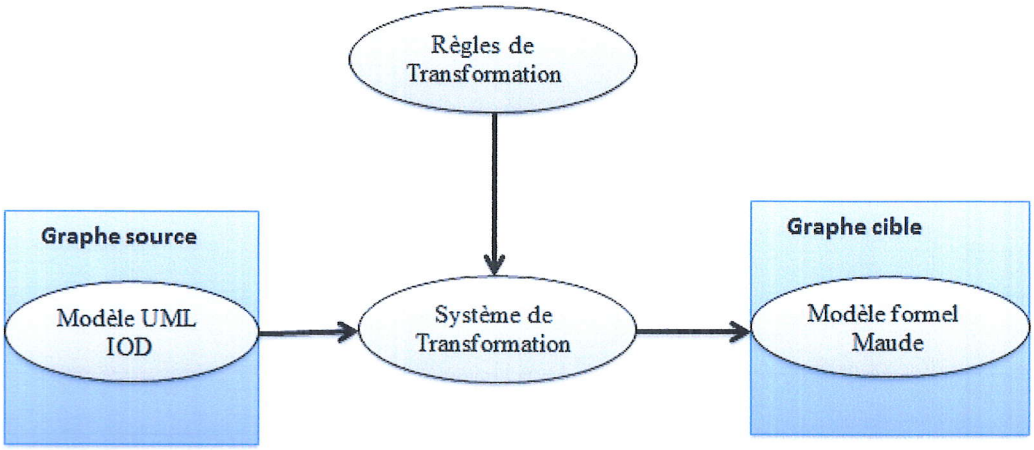


FIGURE 4.1 – Schéma de l'Approche proposée.

4.3 Transformation du diagramme global d'interaction vers le langage Maude

4.3.1 méta-modèle de diagramme global d'interaction

Ce méta-modèle spécifie les attributs, les contraintes, les relations ainsi que l'apparence graphique des nœuds et arcs de l'IOD.

Notre méta-modèle est composé de 11 classes et 20 associations développé par le méta-formalisme CD_ ClassDiagramsV3, dans le but d'avoir un outil intégrant AToM^3 qui offre les outils nécessaires pour modéliser les diagrammes d'interaction IOD (voir la figure 4.2).

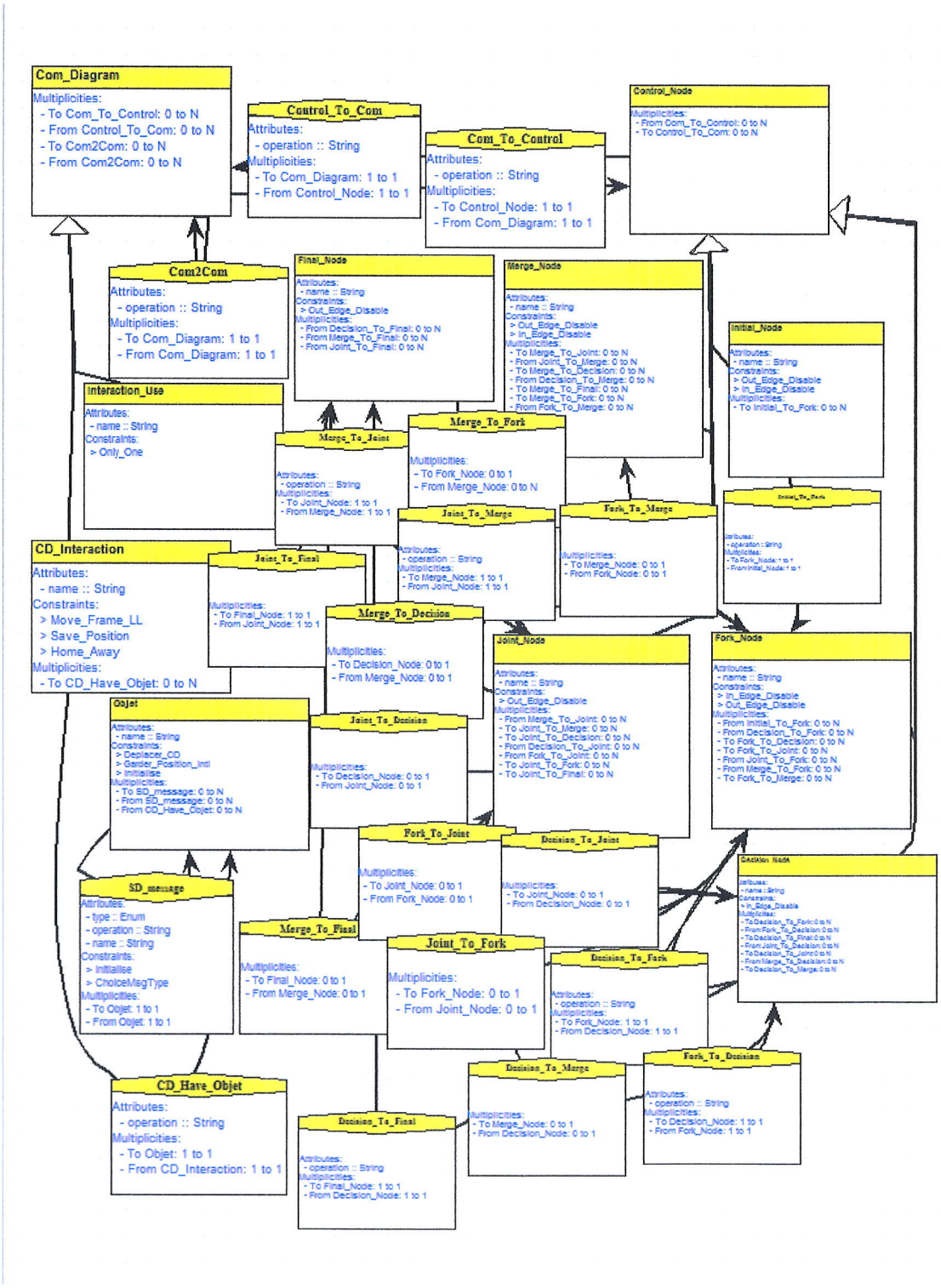


FIGURE 4.2 – Le méta-modèle du modèle global d'interaction proposé.

4.3.2 Les classes et leurs relations

➢ **Controle_ Node :**

C'est une classe mère abstraite qui regroupe tous les nœuds de contrôle, elle n'a pas d'attributs et elle est reliée avec la classe *Com_ Diagram* avec deux associations *(Com_ To_ Control et Control_ To_ Com).*

➢ **Com_ Diagram :**

C'est une classe mère abstraite qui regroupe les différents types des classes de diagramme de communication, elle n'a pas d'attributs et possède une association Com_ To_ Com qui boucle vers elle, et est reliée avec la classe mère *Controle_ Node* avec 02 autres associations *(Com_ To_ Control et Control_ To_ Com).*

➢ **Intial_ Node :**

C'est une classe qui représente le nœud initial du diagramme, elle est reliée avec la classe *Fork_ Node* avec une association *Intial_ To_ Fork.*

➢ **Final_ Node :**

C'est une classe qui représente le nœud Final du diagramme, elle est reliée avec la classe Merge par l'association *Merge_ To_ Final*, avec la classe Decision par l'association *Decision_ To_ Final* et avec la classe Joint par l'association *Joint_ To_ Final.*

➢ **Fork_ Node :**

Cette classe possède un seul arc entrant et plusieurs arcs sortants qui doivent être déclenchés simultanément. Elle n'a pas d'attributs mais possèdes deux contraintes. Elle est reliée avec la classe *Intial_ Node* par l'association *Initial_ To_ Fork*, et avec la classe Decision par les 02 associations *Fork_ To_ Decision et Decision_ To_ Fork.*

➢ **Joint_ Node :**

Cette classe possède deux contraintes. Elle est reliée avec les classes *Merge_ Node, Decision Node, Final_ Node, Fork_ Node*

➢ **Merge_ Node :**

La classe Merge pour accepter un seul flux en sortie parmi plusieurs flux

en entrée, elle possèdes deux contraintes. Elle est reliée avec les classes *Joint_ Node, Decision_ Node, Final_ Node, Fork_ Node.*

➢ **Decision_ Node :**

Cette classe représente le nœud de décision du diagramme. Elle possède une contrainte. Elle est reliée avec la classe Final_ Node par l'association *Decision_ To_ Final*, avec la classe *Fork_ Node* par l' associations *Decision_ To_ Fork*, avec la classe *Merge_ Node* par l'association *Decision_ To_ Merge* et par la classe*Joint_ Node* par l'association *Decision_ To_ Joint.*

➢ **Interaction_ Use :**

C'est une classe fille de la classe mère Com_ Diagram, elle représente le premier type d'interaction du diagramme (le type interaction use). Elle possède un attribut **Name** de type String ainsi que une contrainte.

➢ **CD_ Interaction :**

C'est une classe fille de la classe mère *Com_ Diagram*, Elle représente le cadre du diagramme de communication (de l'interaction) de l'IOD. Elle possède un attribut **Name** de type String représentant le nom de l'activité. Elle possède aussi trois contraintes. Elle est reliée avec la classe Objet par l'association *CD_ Have_ Objet.*

➢ **Objet :**

C'est une classe qui représente les acteurs du diagramme de communication (de l'interaction) de l'IOD. Elle possède un attribut **Name** de type String. Elle possède trois contraintes. Elle est reliée avec la classe *CD_ Interation* par l'association *CD_ Have_ Objet.* Elle possède aussi une association *SD_ message* qui boucle sur elle.

4.3.3 Génération de l'environnement

Après la modélisation du méta-modèle, nous procédons à la génération de l'environnement de méta-modèle. Le méta-modèle généré comporte l'ensemble des classes modélisées sous forme de boutons qui sont prêts à être utilisé pour une modélisation d'un diagramme global d'interaction.

La figure suivante montre l'environnement généré pour les diagrammes global

d'interaction.

FIGURE 4.3 – Environnement de diagramme d'interaction sous AToM^3.

4.3.4 Les Contraintes

Les contraintes de ce méta-modèle empêchent la création les liens interdits dans le diagramme global d'interaction. Nous pouvons modéliser n'importe quel diagramme global d'interaction, par l'utilisation de l'outil présenté dans la figure 4.3. Nous donnons à titre d'exemple le modèle de la figure 4.4 :

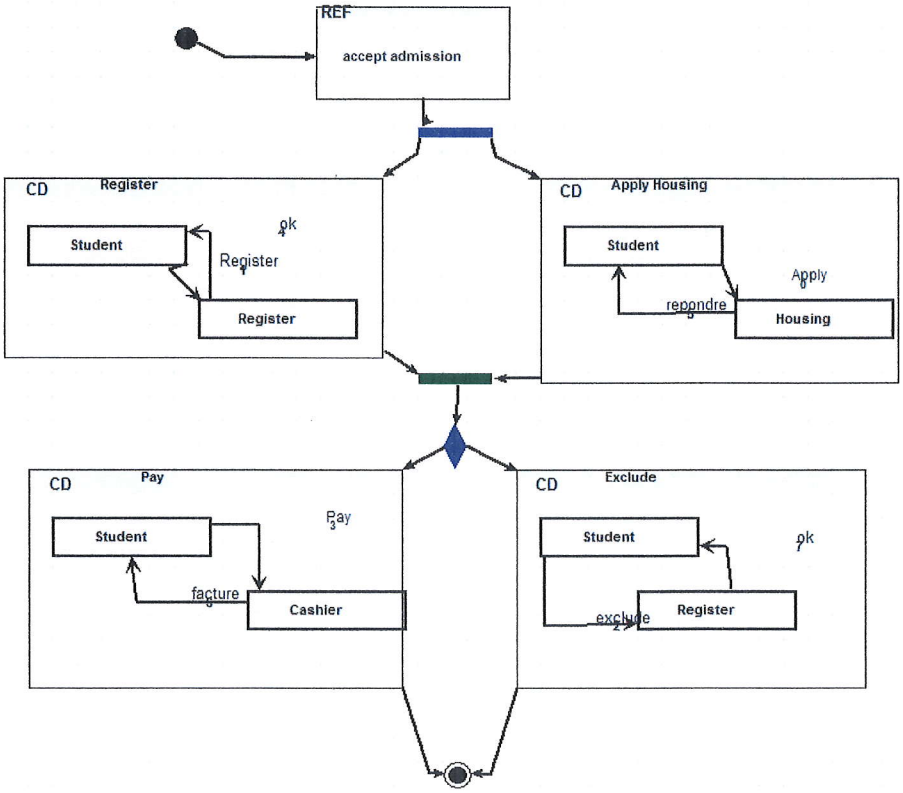


FIGURE 4.4 – Modèle IOD crée avec L'outil de modélisation.

Nous donnons des exemples de violation des contraintes avec l'outil de modélisation généré :

- **In_ Edge_ Disable :** son rôle est d'interdire la création d'un lien entrant vers le nœud initial.

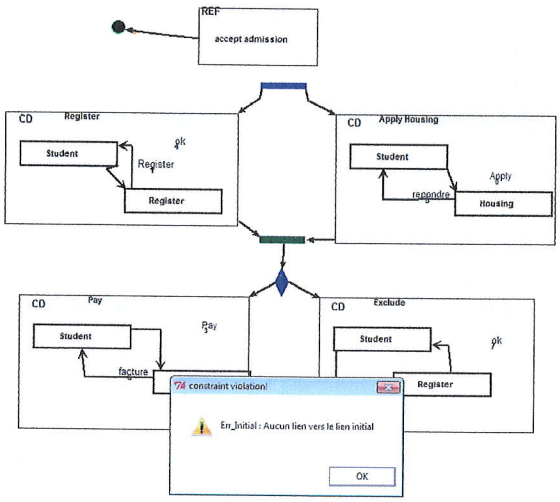


FIGURE 4.5 – Violation de contrainte de classe Inital_ Node avec l'outil de modélisation.

◆ **Out_ Edge_ Disable :** Son but est d'interdire tout lien sortant du nœud final.

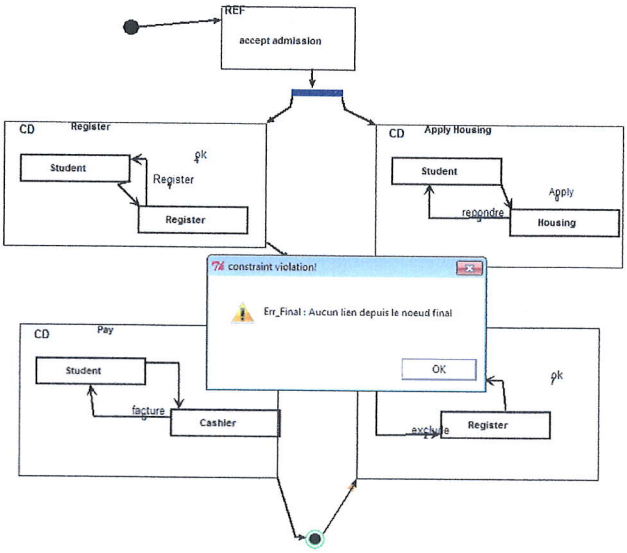


FIGURE 4.6 – Violation de contrainte de classe Final_ Node avec l'outil de modélisation.

◆ **Only_ One :** Cette contrainte se déclenche lorsqu'il y'a plus d'un nœud entrant et/ou plus d'un nœud sortant.

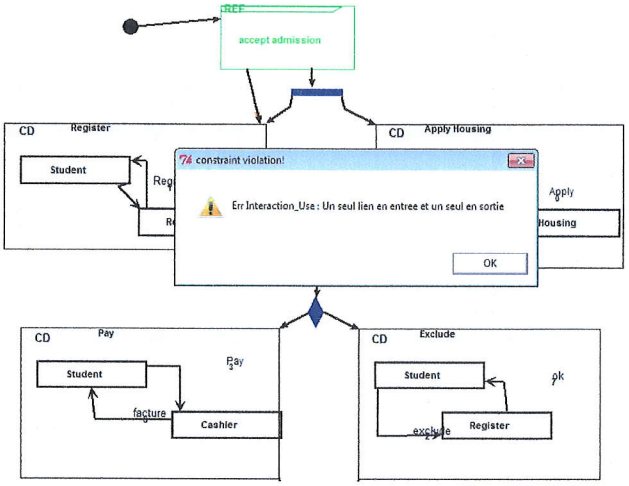


FIGURE 4.7 – Violation de contrainte de classe Interaction_ Use avec l'outil de modélisation

◆ **Autre contraintes**

➤ **ChoiceMsgType** :son rôle est de changer l'apparence de type d'un message choisi fonction de son attribut Type (Synchrone, Asynchrone, aller). Est une contrainte d'association SD_ message.

➤ **Home_ Away** : Son but est d'empêcher qu'un lien parte du nœud CD_ Interaction vers un nud et puis à partir de ce dernier il revient vers CD_ Interaction (l'effet Aller-Retour)

➤ **initialise** :cette contrainte d'association SD_ message pour l'initialisation des numéros des messages qui sont initialisés par zéro (0).

4.3.5 La grammaire de graphe proposée

Les règles de la grammaire permettent d'établir une correspondance entre le modèle de graphe d'un I.O.D et la syntaxe textuelle de Maude.
Nous avons proposé une grammaire de graphes avec une action initiale, 16 règles de transformation qui sont exécutées dans un ordre précis selon leurs priorités pour générer un IOD, et une action finale. L'application de cette grammaire à un modèle IOD conduit à la génération d'un fichier ayant l'extension **.Maude** qui contient sa description sous langage Maude, pour montrer comment passer du graphique IOD vers sa représentation Maude.

Et une autre grammaire pour la génération d'un diagramme de communication avec 4 règles, cette grammaire aussi conduit à la génération d'un fichier ayant l'extension **.Maude**.

❖ **La grammaire :**

✦ **L'action Initial :** Dans l'action initiale de la grammaire de graphe nous avons créé un fichier (.Maude) à accès séquentiel (c.-à-d. les données sont enregistrées dans le fichier les unes à la suite des autres) pour stocker le code du fichier généré. L'attribut temporaire **Visited** est utilisé pour indiquer si la classe a été déjà traitée ou non et **Current** pour indiquer si elle est en cours de traitement et l'attribut **Generer** pour indiquer si tous ces nœuds adjacents ont été importés. Tous ces attributs temporaires sont initialisés à 0.

```
############## open the file ################
# 1st, find the path of files
stb = self.rewritingSystem.parent.statusbar
state = stb.getState(stb.MODEL)
fichiers = state[1][0]
FilePath = os.path.split(fichiers)

#2nd open files
name = graph.name.toString()
if name !="":
    self.rewritingSystem.ADFile = open(FilePath[0]+"/"+name+".Maude","w+t")
else:
    self.rewritingSystem.ADFile = open(FilePath[0]+"/name_fich.Maude","w+t")
```

FIGURE 4.8 – La création du fichier sous AToM^3.

L'action initiale écrit l'entête du fichier Maude, qui représente le module fonctionnel du diagramme IOD. La figure suivante représente la création de Module fonctionnel sous fichier **.Maude** créé.

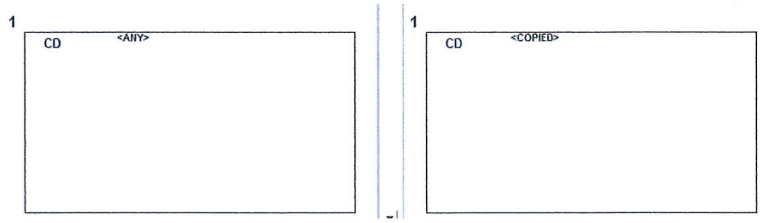
```
## 3rd  write in files
self.rewritingSystem.ADFile.write('mod IOD_Com_diag is''\n'
'including NAT .''\n' 'including STRING .                    ''\n''including
LISTEMSG .''\n''\n'
'\n''sort CONFIGURATION . ''\n'
'sorts InitialNoeud FinalNoeud CADRE NomCD FinCADRE Objet Message Type ARC
.''\n'
'subsorts InitialNoeud FinalNoeud CADRE NomCD FinCADRE LifeLine Message ARC <
CONFIGURATION .''\n'
'subsort TypeMsg < arc .''\n'
'\n'
'op _ _ : CONFIGURATION CONFIGURATION -> CONFIGURATION [assoc comm id: null] .'
'\n'
'op null : -> CONFIGURATION [ctor] .''\n'
'op Initial : -> InitialNoeud .''\n'

'op Final : -> FinalNoeud . ''\n'
'op Obj : -> Objet . ''\n'
'ops Asynchrone Synchrone aller : -> TypeMsg .''\n'
'op Envoi : Objet ARC Objet -> Message [ctor] .''\n'
'op START : NomCD -> CADRE . ''\n'
'op END : NomCD -> FinCADRE .' +string.whitespace[1])
```

FIGURE 4.9 – Création de Module fonctionnel de fichier sous AToM^3.

✦ **Ensemble des règles :**

☛ **Règle1 : Gene_ CD**

LHS **RHS**



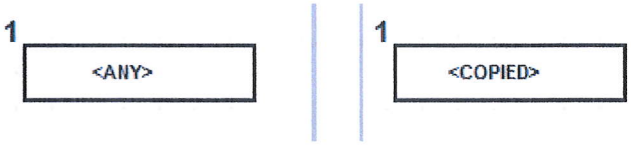
Condition : Generer == 0

Action : Generer == 1

☛ **Règle2 :Gen_ Objet**

Objectif : Cette règle permet de convertir l'attribut Name de la classe conformément à la syntaxe de Maude et de le stocker.

LHS **RHS**



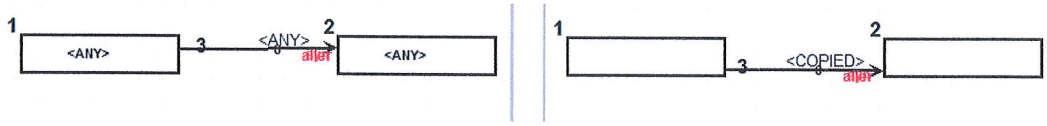
Condition : Generer == 0

Action : Generer == 1

☛ **Règle3 :Gen_ Msg**

Objectif : Permet de stocker le nom du message et de préciser l'ordre du message.

LHS **RHS**



Condition : Generer == 0

Action : Generer == 1

☛ **Règle4 :Gen_ Fork**

Objectif : Permet de stocker tous les nœuds sortants du nœud Fork.

LHS **RHS**

Condition : Generer == 0

Action : Generer == 1

☛ **Règle5 :GenDecision**

Objectif : stocke tous les nœuds sortants avec leur type (Interaction_ Use, CD_ Interaction ou Final_ Node).

LHS **RHS**

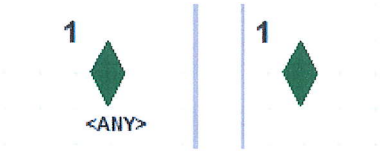
Condition : Generer == 0

Action : Generer == 1

☛ **Règle6 :Gen_ Merge**

Objectif : Stocke tous les nœuds entrants avec leur type.

LHS **RHS**



Condition : Generer == 0

Action : Generer == 1

☛ **Règle7 :Gen_ Joint**

Objectif : Stocke pour chaque nœud Joint, tous les nœuds entrants avec leur type.

LHS **RHS**

Condition : Generer == 0

Action : Generer == 1

☛ **Règle8 :Gen_ Initiale**

Objectif : Permet de définir les opérations utilisées dans le module fonctionnel de Maude à partir des listes collectées par les règles précédentes. Elle introduit aussi dans le fichier Maude, la toute première règle du

LHS **RHS**

module système de Maude.

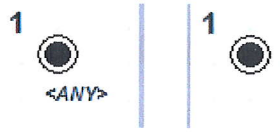
Condition : Generer == 0

Action : Generer == 1

☛ **Règle9 :Gen_ Finale**

Objectif : Stocke tous les nœuds entrants vers le nœud final avec leurs types.

LHS **RHS**



Condition : Generer == 0

Action : Generer == 1

☛ **Règle10 :rule_ Finale**

Objectif : Permet de produire dans le fichier Maude toutes les règles du module système de MAUDE relative au nœud Final (c.-à-d., entrants dans le nœud Final). Et cela à partir de la liste générée par la règle **Gen_ Finale** .

LHS **RHS**

Condition : Generer == 1 ; Current == 0 ; Visited == 0

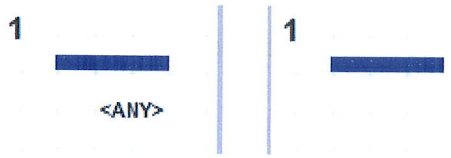
Action : Generer == 2 ; Current == 1 ; Visited == 1

☛ **Règle11 :rule_ Fork**

Objectif : Permet de produire toutes les règles du module système relatives au nœud **Fork** courant sauf s'il a pour entrer un nœud initial

(car cette règle sera produite par **Gen_ Initiale**) de Maude dans le fichier.

LHS **RHS**



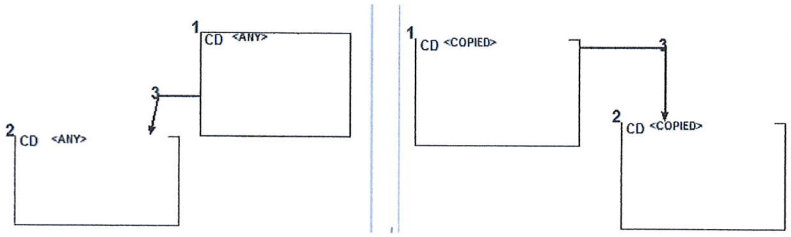
Condition : Generer == 1 ; Current == 0 ; Visited == 0

Action : Generer == 2 ; Current == 1 ; Visited == 1

☛ **Règle12 :CD2CD**

Objectif : Produit une règle dans le fichier Maude permettant de faire le passage d'une CD_ Interaction vers une autre CD_ Interaction.

LHS **RHS**



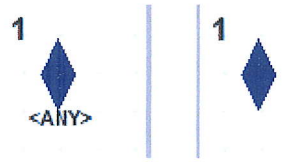
Condition : CD.Generer == 1

Action : CD.Generer == 2

☛ **Règle13 :rule_ Decision**

Objectif : En se basant sur la liste contenant tous les nœuds sortants, elle produit toutes les règles Maude équivalentes. S'il y'a dans la liste des nœuds sortants vers un nœud final, il ne sera pas traité par cette règle mais par **Final** .

LHS **RHS**



Condition : Generer == 1 ; Current == 0

Action : Generer == 2 ; Current == 1

☛ **Règle14 : rule_ Merge**

Objectif : En se basant sur la liste contenant tous les nœuds entrants vers le nœud Merge, elle produit toutes les règles Maude équivalentes. Si le **Merge** a un arc sortant vers le nœud Final. Il ne sera pas traité par cette règle mais par la règle **Final** .

LHS **RHS**

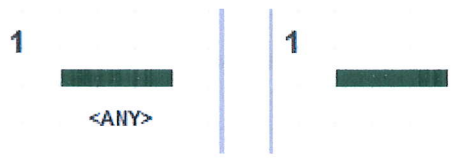
Condition : Generer == 1 ; Current == 0

Action : Generer == 2 ; Current == 1

☛ **Règle15 : rule_ Joint**

Objectif : En se basant sur la liste contenant tous les nœuds entrants vers le nœud Joint, elle produit toutes les règles Maude équivalentes. Si le Joint à un arc sortant vers le nœud Final. Il ne sera pas traité par cette règle mais par la règle **Final** .

LHS **RHS**

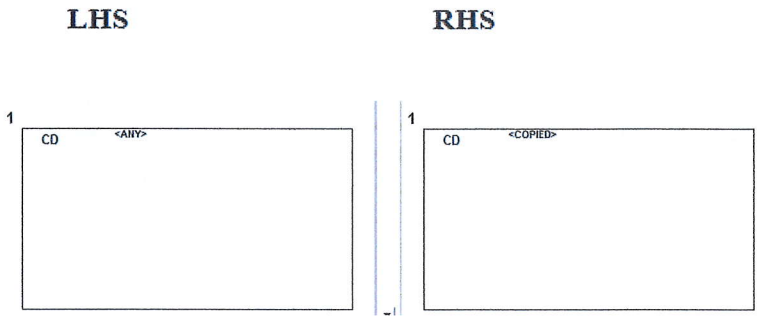


Condition : Generer == 1 ; Current == 0

Action : Generer == 2 ; Current == 1

☛ **Règle16 : rule_ CD_**

Objectif : A partir des listes collectées par Gene_ CD, Gen_ Objet Gen_ Msg, et en se basant sur l'ordre définit par cette dernière. InCD permet de produire les règles Maude équivalentes qui représente le diagramme de communication à l'intérieur de l'interaction(CD_ Interaction).



Condition : CD.Generer == 2 ; CD.Visited == 0

Action : CD.Generer == 3 ; CD.Visited == 1

✦ **L'action Finale :** À la fin de l'exécution des règles, l'action finale de cette grammaire sert à détruire toutes les variables temporaires créées au cours de cette grammaire, ainsi qu'à enregistrer et fermer le fichier Maude généré.

```
#------------------ Finir le fichier --------------------
self.rewritingSystem.DAFile.write('\n')
self.rewritingSystem.DAFile.write("endm")
self.rewritingSystem.DAFile.write('\n')
self.rewritingSystem.DAFile.write("set trace on .")
self.rewritingSystem.DAFile.write('\n')
self.rewritingSystem.DAFile.write("rew Initial .")
###-------------------- Fermeture du fichier -----------
self.rewritingSystem.DAFile.close()
```

FIGURE 4.10 – Fermeture de fichier dans l'action finale

4.4 Exemple

✦ **Description textuelle générée :** Afin de mieux comprendre cette description, nous proposons un exemple d'étude de cas qui est représenté par son diagramme IOD dans la figure ci-dessous. Nous commencerons par le modéliser grâce à notre outil visuel de modélisation, puis nous exécuterons la grammaire de graphe ainsi que le fichier (.Maude) généré dans l'environnement Maude pour une vérification formelle.

Cet exemple contient 4 nœuds de contrôle, le nœud Initial, le nœud Final, le Fork et le Decision, ainsi que 3 nœuds d'interaction de type CD_interaction.

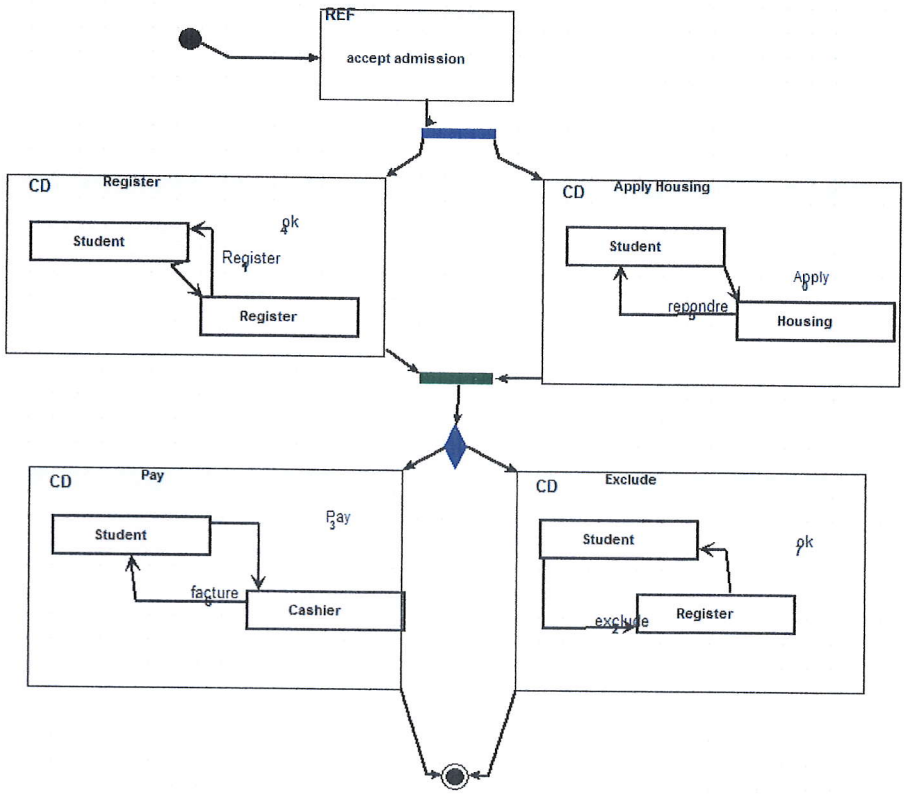


FIGURE 4.11 – Exemple montre un étudiant qui a été accepté dans une université modélisé avec l'outil.

Après la modélisation du diagramme IOD (Exemple), nous exécutons les règles de transformation avec ATOM^3 comme montré dans la figure 4.12 suivante :

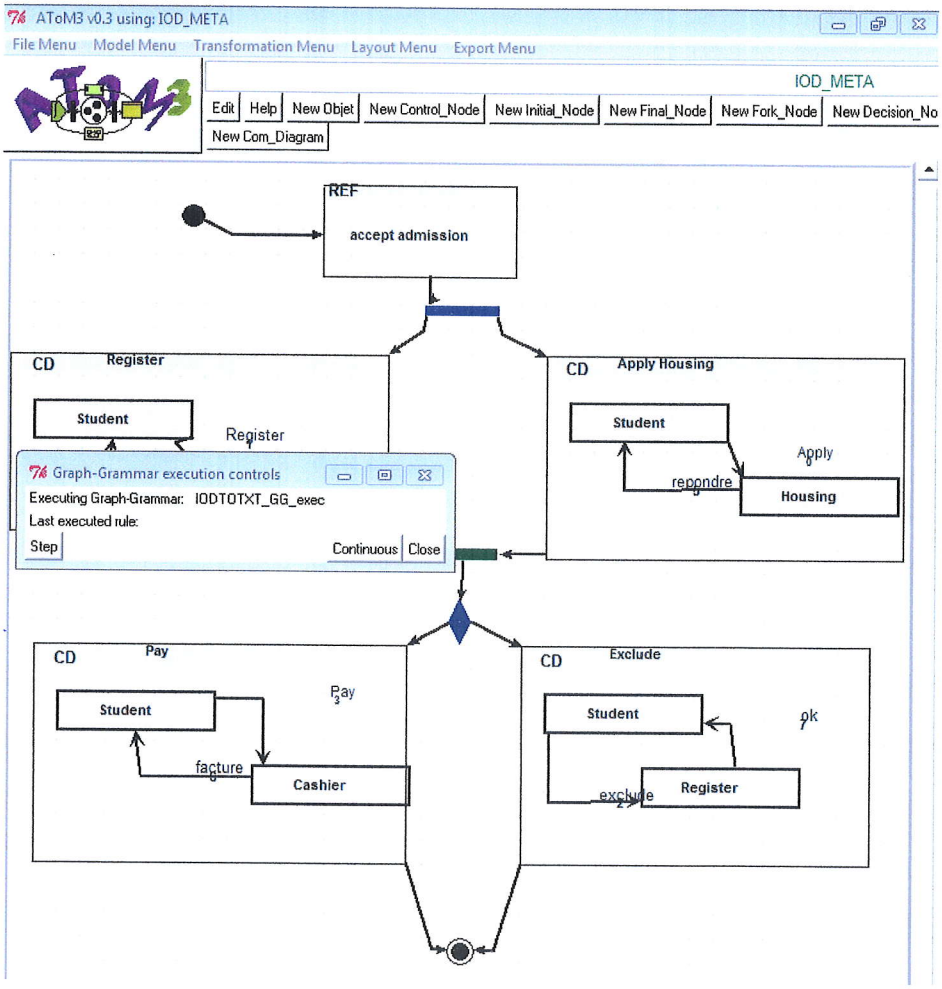


FIGURE 4.12 – Exécution des règles de transformation.

Après la fin de l'exécution, l'outil génère la description textuelle du diagramme dans un fichier .Maude qui est représenté dans la figure suivante :

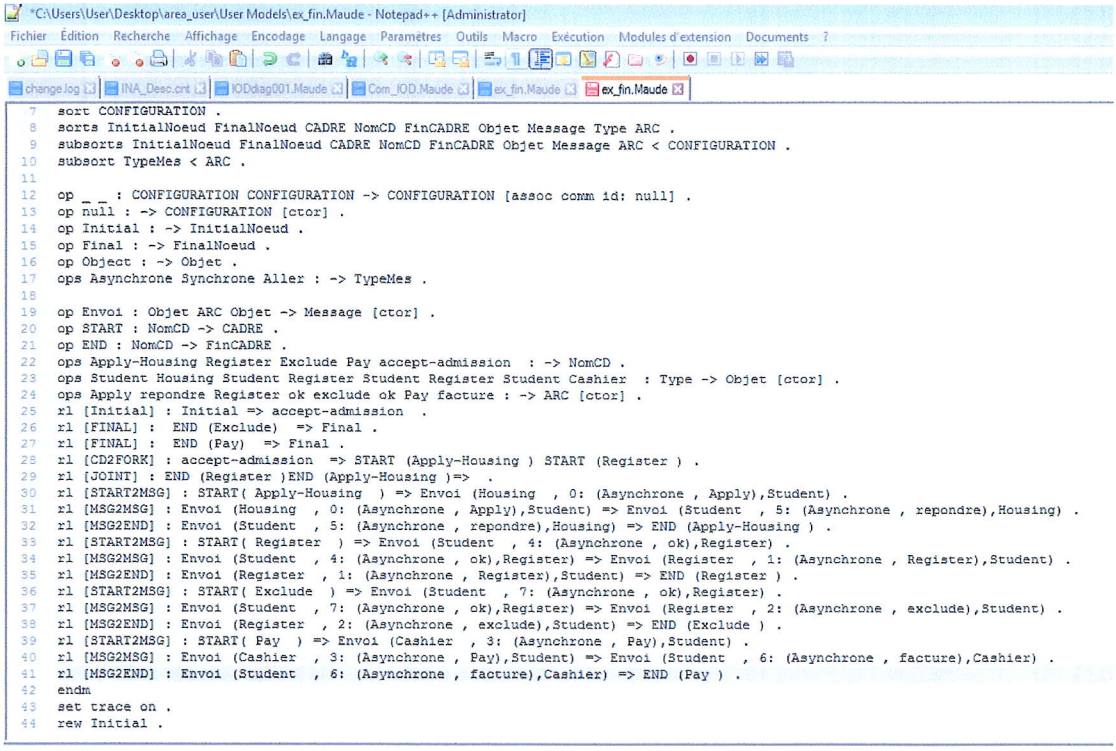


FIGURE 4.13 – La description textuelle sous MAUDE.

4.5 Conclusion

Dans ce chapitre nous avons proposé une approche de transformation du diagramme de communication composante du inclue dans le diagramme global d'interaction (IOD) vers Maude.

Notre approche est basée sur la méta-modélisation et l'automatisation de la transformation en se basant sur des grammaires de graphes et en utilisant l'outil ATOM^3. L'exécution de la grammaire de graphe transforme ces derniers vers le code MAUDE équivalent.

Conclusion générale

Le travail présenté dans ce mémoire s'inscrit dans le domaine de l'I.D.M (l'ingénierie dirigée par les modèles) et des méthodes formelles. Nous avons proposé des règles théoriques de transformation de modèle vers Maude, ensuite nous avons présenté notre approche pour implémenter ces règles en se basant sur la transformation de graphe à l'aide de l'outil ATOM^3. Cet outil supporte la méta-modélisation. Une fois la méta-spécification établie, ATOM^3 génère un outil graphique pour la manipulation des différents modèles décrits dans le formalisme spécifié. L'écriture des règles est facile relativement à d'autres outils du fait de son caractère graphique. Il est muni d'un éditeur de règles permettant une édition facile de la partie gauche (LHS) et de la partie droite (RHS).

Cette approche permet aussi de palier à une des faiblesses que présente UML devant les systèmes à grande complexité, et cela en se basant sur les méthodes formelles qui offrent beaucoup plus de rigueur dans la vérification de ces systèmes. Tout au long de notre travail, nous avons abordé les deux points étapes essentielles suivants :

- Nous avons tout d'abord proposé une méta-modélisation du diagramme source (I.O.D), puis nous avant généré à partir de cette méta-modélisation notre modèle et cela grâce à un outil visuel permettant la création des modèles nommé ATOM^3.
- La deuxième étape, contient une grammaire de graphe que nous avons proposée et qui exécute des règles de transformation, nous avons abouties à un fichier qui représente une description textuelle du modèle source, cette description est formulée en respectant la syntaxe d'un langage de

programmation formel nommé MAUDE.

Maude est un langage de spécification et de programmation basé sur la logique de réécriture. Cette dernière a été tout particulièrement développée en vue de spécifier plus aisément les systèmes concurrentiels.

Dans un futur travail, nous voulons continuer la transformation de graphe, pour les nœuds non traités tels que l'interaction use ainsi que la phase de vérification en utilisant le noyau Model Cheking de l'environnement Maude.

Finalement, et dans le but d'arriver à développer une approche totalement automatique, incluant tous les diagrammes d'UML, nous suggérerons de continuer la transformation des autres diagrammes UML (diagramme de temps, diagramme d'état/transition, etc) vers le langage Maude en utilisant toujours la transformation de graphes et l'outil ATOM^3.

Bibliographie

[1] Modélisation uml. https ://fr.wikiversity.org/wiki/Mod

[2] *UML 2 et les design patterns.Analyse et conception orientées objet et développment itératif.* 2005.

[3] Modélisation avec uml. https ://www.commentcamarche.net/contents/1142-modelisation-avec-uml.

[4] Mecheri Nacera. Une approche hybride pour transformer les modèles. Master's thesis, Université Ahmed Ben Balla Oran, 27.10.2015.

[5] Uml 2 tutoriel. https ://www.sparxsystems.fr/resources/uml2_ tutorial/index.html.

[6] Mouna AOUAG. Des diagrammes uml 2.0 vers les diagrammes orientés aspect à l'aide de transformation de graphes, 2014.

[7] La modélisation uml. http ://dspace.univ-tlemcen.dz/bitstream/112/6325/3/chapitre2.pdf.

[8] El hillali Kerkouche. Modélisation multi-paradigme : Une approche basée sur la transformation de graphes, 04 / 07 /2011.

[9] Bouchemal Bedouhene. VÉrification formelle des ecatnets : Une approche basÉe sur la logique de rÉÉcriture. Master's thesis, Université de Jijel, 2010.

[10] Amira Hakim. Ingéierie des modèle atlas transformation language, 2015.

[11] Sylvain ANDRE. Mda (model driven architecture) principes et états de l ?art. Master's thesis, CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERSCENTRE D ?ENSEIGNEMENT DE LYON, 2004.

[12] Mouna Bouarioua. Une approche basée transformation de graphes pour la génération de modèles de réseaux de petri analysables à partir de diagrammes uml, 24/ 04 /2013.

[13] *UML 2 par la pratiqueétude de cas et exercice corrigés.* septembre2006.

[14] Diagrammes de communication. http ://infocenter.sybase.com/help/index.jsp ?topic=/com.sybase.infocenter.dc31018.1653/doc/html/

[15] Kholladi Mohamed Naoufel. Une approche de transformation de la notation bpmn vers bpel basée sur la transformation de graphe. Master's thesis, Université Mentouri Constantine.

[16] L'approche orientée objet et uml. http ://lim.univ-reunion.fr/staff/courdier/old/cours/GLAndCOO/2_ coursOMT-UML/2_ coursOMT-UML.pdf.

[17] Model driven architecture. https ://fr.wikipedia.org/wiki/Model_ driven_ architecture.

[18] Zahoui Anissa Amel. Developpement d'une chaine d'outil en fonction du nouveaux standard fondationnel uml(fuml). Master's thesis, Université Badji Mokhtar, 2014.

[19] Uml 2. https ://laurent-audibert.developpez.com/Cours-UML/ ?page=diagrammes-interaction.

[20] 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.

[21] Bernard Coulette Samba Diaw, Redouane Lbath. Etat de l'art sur le développement logicielbasé sur les transformations de modèles. https ://pdfs.semanticscholar.org7617/1c05180e17dc24519f51db506367ee35f437.pdf.

[22] *UML 2 ANALYSE ET CONCEPTION.* 2009.

[23] Bendiaf Messaoud. Spécification et vérification des systèmes embriqués stemps réel en utilisant la logique de réécriture, 15/05/2018.

[24] Mohamed Rédha BAHRI. Une approche intégrée mobile-uml/réseaux de petri pour l'analyse des systèmes distribués à base d'agents mobiles, 2015.

[25] Khaled Khalfaoui. Une approche de spécification des changements des besoins basée transformationde graphes, 24Juin 2014.

[26] https ://blog.octo.com/introduction-aux-graphes-avec-neo4j-et-gephi/.

[27] Généralités sur l'approche mda. https ://laine.developpez.com/tutoriels/alm/mda/gene approche-mda/.

[28] Ingénierie dirigée par les modèles (idm). https ://www.urbanisation-si.com/articles/ingenierie-dirigee-par-les-modeles-idm.

[29] 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 Aout-1955-SKIKDA.

[30] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical computer science*, 96(1) :73–155, 1992.

[31] Douibi Halima. Intégration de xml dans le cadre de la logique de réécriture. Master's thesis, UNIVERSITÉ MENTOURI DE CONSTANTINE, 2006.

[32] José Meseguer and Grigore Roşu. The rewriting logic semantics project. *Electronic Notes in Theoretical Computer Science*, 156(1) :27–56, 2006.

[33] 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.

[34] Boudia Malika. Transformation des diagrammes d'états-transitions vers maude. Master's thesis, UNIVERSITE DE M'SILA, 2007.

[35] PATRICE GAGNON. VÉrification formelle de diagrammes uml : Une approche basÉe sur la logique de rÉÉcriture. Master's thesis, UNIVERSITÉ DU QUÉBEC.

[36] LATRECHE FATEH. Expression et vérification des contraintesnon fonctionnelles d'une architecture sadl. Master's thesis, UNIVERSITÉ MENTOURI DE CONSTANTINE, 10/12/2007.

[37] Noura Boudiaf. Développement des outils basés maude pour les ecatnets. domaine d'application : Analyse des programmes ada, 2007.

[38] Firas Bacha. Une approche mda pour l'intégration de la personnalisation du contenu dans la conception et la génération des applications interactives, 2013.

[39] 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.

[40] MERMOUNE Hamza et CHELGHAM Mohammed-Anis. Une approche de transformation automatique sur l'analyse du diagramme global d'interaction. Master's thesis, Université de Jijel Faculté des Sciences Exactes et Informatique Département d'Informatique, 2015.

Résumé

Le travail présenté dans ce manuscrit s'inscrit dans le domaine de l'Ingénierie Dirigée par les Modèles (IDM) en général et dans la transformation et la vérification formelle de modèles a l'aide des grammaires de graphes en particulier. Nous proposons une approche de transformation de modèles qui se base sur la méta-modélisation. Cette approche propose une transformation des diagrammes globaux d'interaction (IOD) vers Maude qui est un langage de programmation formel et déclaratif basé sur la théorie mathématique de la logique de réécriture.

L'approche proposée permet de générer du code automatiquement à partir d'un graphe, via une grammaire de Transformation proposée par un outil de multi-modélisation ATOM^3 avec lequel il est possible de spécifier un méta-modèle pour l'IOD et une grammaire de graphes pour effectuer la transformation.

Mots clés : Ingénierie Dirigée par les Modèles, Méta-modélisation, Transformation de Modèles, Transformation de Graphes, Grammaires de Graphes, AToM^3, Logique de réécriture, Langage Maude, UML, Diagrammes de communication, Diagramme Global d'Interaction, IOD.

Abstract

The work done by this manuscript considers generally as a field of a Model Driven Engineering (MDE) and particularly as a formal transformation and verification models using the graph grammar. We propose a model transformation approach based on the meta-modeling. This approach proposes a global interaction diagrams (IOD) transformation to Maude which is a formal and declarative programming language based on the mathematical theory of rewriting logic.

This approach makes possible to generate the code automatically from the graph, via a transformation grammar proposed by ATOM^3. This multi-modeling language allows to specify IOD meta-model and graphs grammar for the conversion process.

Keywords: Model Driven Engineering, Meta-modeling, Model Transformation, Graph Transformation, Graph Grammars, AToM3, Rewriting logics, Maude language, UML, Communication diagrams, Interaction Overview Diagram.

ملخص

العمل المنجز في هذه المذكرة ينتمي عموما إلى مجال الهندسة الموجهة بنموذج (IDM) وفي التحول الرسمي والتحقق من النماذج باستخدام قواعد الرسم البياني على وجه الخصوص. لهذا نقترح اتباع منهجية التحويل النموذجي القائم على نموذج الفوقية الذي يسمح بتحويل مخططات التفاعل العامة (IOD) إلى لغة Maude . هذه الأخيرة تمثل لغة برمجة تشكيلية و بيانية أساسية تعتمد على النظرية الرياضية المنطيقية لإعادة الصياغة.

تتيح الطريقة المتبعة إنشاء الرموز تلقائيًا من خلال الرسم البياني، بواسطة قواعد التحويل المتحصل عليها بلغة ATOM^3 والتي من خلالها يمكن تحديد نموذج meta لنموذج IOD و الرسوم البيانية لأداء التحول.

الكلمات المفتاحية: الهندسة النموذجية ، النمذجة الفوقية ، تحويل النموذج ، تحويل الرسم البياني ، قواعد الرسم البياني ، AToM3 ، إعادة كتابة المنطق ، لغة Maude، UML ، مخططات الاتصال ، مخطط التفاعل العالمي ، IOD.