الجمهوريـة الجزائـريـة الديمقراطيـة الشعبيـة
République Algérienne Démocratique et Populaire
وزارة التعليـم العالـي والبحـث العلمـي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Nº Réf :……

Centre Universitaire
Abd elhafid boussouf Mila

Institut des sciences et de la technologie **Département de Mathématiques et Informatique**

Mémoire préparé En vue de l'obtention du diplôme de Master

En: Informatique

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

Spécification formelle des modèles orientés aspect, une approche basée sur la transformation de graphes et le langage Maude

Préparé par : Dib Wiam
Saadaoui Amani

Soutenue devant le jury

Djaaboub Salim	**MCB**	**C. U. Abdelhafid Boussouf, Mila**	**Président**
Aouag Mouna	**MCB**	**C. U. Abdelhafid Boussouf, Mila**	**Rapporteur**
Hettab Abdelkamel	**MCB**	**C. U. Abdelhafid Boussouf, Mila**	**Examinateur**

Année universitaire :2022/2023

REMERCIEMENT

*Nous remercions d'abord et avant tout **Allah** qui nous adonné le courage et la patience pour réaliser ce travail.*

*Un remerciement particulier à notre encadreur **Aouag Mouna** pour sa présence, son aide et surtout pour ses précieux conseils qui nous ont assistés pour l'accomplissement de notre projet,*
Nous adressons également nos vifs remerciements aux membres du jury qui ont bien voulu et accepté d'examiner ce modeste travail,
Sans oublier tous les enseignants qui ont contribué à notre formation durant notre vie Scolaire particulièrement les enseignants de notre institut.

Finalement, nous remercions très sincèrement tous nos famille pour leur encouragement sans limite.

Wiam et Amani

اهداء

هو الله أول من يُشكر آناء الليل و أطراف النهار إذ أرسل فينا عبده و رسوله عليه الصلاة و السلام بقرآنه المبين فعلمنا مالم نعلم و حثنا على طلب العلم أينما وجد .. له الثناء العظيم حيث وفقنا لتثمين هذه الخطوة في طريق الجد .

إلى الذي باع راحة شبابه ليشق لي الطريق ، من كان في حياتي شمعة ساطعة البريق ، من غرس فيَّ مكارم الأخلاق و تحمل لأجلي المشاق ، الحنون الغالي .. إليك أبتي.

إلى رونق حياتي و فرحة أيامي ، من سهرت لأجلي الليالي لكي أسمو و أصل للمعالي إلى منبع الحب و الحنان .. إليك أمي

لى توأم روحي و ريحاني و سندي في الحياة و من بها أتكئ و أحتمي أختي نهلة .. و لابنة عمي "ريمة " التي سعت دائما لتحقيق دور الاخت و بجدارة .

إلى إخوتي و عزتي و مفخرتي عماد الدين و مولود ، و أختاي لميس عائشة و مسك الختام سلسبيل .

إلى الخال " نذير " و الأعمام " مراد و عبد المجيد " من كانوا دائما الكتف الذي لا يميل و لكل فرد تمنى لنا طريق النجاح و الخير .

إلى زميلاتي و زملائي حيث أصر كل أحد منهم أن يكون صديق صدوق صادق الوعد منصفا .

و يحق للعقل أن يسترخي بذكر طيبة الأثر رفيقة الدرب خليلة العمر صديقتي في الدراسة و صحبتي في حفظ كتاب الله أختي من الأيام " وئام ذيب ".

و قال ربي حدثوا بنعمي وليشهد من في السماء و من في الارض أنكم نعمُ الله إلي .

طبعا إلى الأستاذة المشرفة " عواق منى " التي كان لها دور كبير و مهم و بارز في انجاح هذا العمل . و للاستاذ " مغزيلي السعيد " له الاحترام و التقدير على ما أبذله من جهد لمساعدتنا .

و دون شك كل الإمتنان و الفخر لي .. لأني سعيت للنجاة من أمور لا يعلمها إلا رب الخلائق ، فأيقنت و آمنت حينها أن ليس للإنسان إلا ما سعى .

قال تعالى :" و لَسوْف يعْطِيك ربُك فترضَى "

أماني...

اهداء

بسم الله وكفى والصلاة والسلام على الحبيب المصطفى الحمدلله ما انتهى دربٌ ولا ختمَ جهدٌ ولا تم سعيٌ إلا بفضلهِ وما تخطّى العبدُ من عقبات وصعوبات إلا بتوفيقه فالحمدلله حمد الشاكرين وعدد ما ذكره الذاكرين، له عظيم الثناء على تكريمه لنا بطلب العلم وعلى منزلة وقوفنا في هذا اليوم لنعلن تخرجنا بكل فخر {وَآخِرُ دَعْوَاهُمْ أَنِ الْحَمْدُ لِلَّهِ رَبِّ الْعَالَمِينَ}

اما بعد أهدي كلماتي وما تسعكم الكلمات ..

إلى أمي وأبي ما تصف الكلمات وما تبلغ عظيم ما اكرمتموني به، نوري المضاء دائما وسندي في الدنيا، من جاهدا من أجلي من شقا الطريق لرؤيتي وأنا اتوج بتخرجي، ما بوسعي رد جميلٍ لكما الا رؤيتي وأنا في مرتبة العلى رزقني الله بنيل عظيم الشرف في الدنيا بطلب العلم الذي اوصانا به وحثنا عليه رسوله الكريم، وبتوفيق الله ان يجعلني خاتمة لكتابه وتتوجيكما في الآخرة بتاج الوقار ...

إلى إخوتي سلمى وإكرام ورحاب من أتوجه لهم في محنتي في سعادتي وحزني من ساندوني بكلماتهم وأفعالهم، الى ابناءك اختي سلمى يا رزق الله لي

إلى اعمامي وعماتي الى كل اخوالي وخالاتي الى زوجات اعمامي اولادهم صغارهم وكبارهم الى بنات عماتي سندي لينا وندى الى روحك عمي 'عصام' أهديك تخرجي يا من كنت تسأل عني في كل حين من كنت تواسيني اذا واجهة صعوبة في دراستي ومن كنت تسعى لاضحاكي رغم محنتك الى روحك الطاهرة في جنات النعيم

الى صديقاتي إكرام وريما وآمال الى من كانو يتحملون ثرثرتي عن دراستي وكانو هم السند ورفقاء الدرب

الى من التقيتهم صدفة فنثروا عبق لطفهم في طريقي من دعى لي جهرا او خفاء زميلاتي وزملائي من جمعتني بهم مقاعد الدراسة الى كل روح صافية كانت بجانبي شكرا

الى من اعانونا على هذا العمل استاذتنا "عواق منى" من ساهمت في انجاحه الى الاستاذ "مغزيلي السعيد" الذي ساعدنا كثيرا الى كل استاذ درسني وجد من اجلي اليكم يا رسل العلم

وهنا يطيب الخاطر بتقديم جزيل الشكر لك الى رفيقة الدرب رفيقة الدراسة 'سعداوي أماني' من واجهنا الكثير مع بعض من كانت عمادي اذا يئست من ضحكت معي وحزنت لحزني من رافقتني في حفظ كتاب الله .. اليك يا رفيقة دربي

وفي الاخير فليشهد الله على جهدنا في سنوات الدراسة السبعة عشر وعلى جدنا واجتهادنا لنيل هذه المرتبة وعلى ما تحملناه من كلمات البعض وتحطيم الاخرين لنا.. نحن هنا بفضل الله ثم سعينا

قال تعالى: " وَأَنْ لَيْسَ لِلإنسَانِ إِلَّا مَا سَعَى ۝ وَأَنَّ سَعْيَهُ سَوْفَ يُرَى"

وئام ...

Résumé

La modélisation en informatique est l'étape la plus importante dans le développement d'un logiciel. Elle facilite la compréhension du fonctionnement d'un système avant sa réalisation en produisant un modèle. Ils existent plusieurs méthodes de modélisation comme la Modélisation Orientée Aspect(MOA) et la Modélisation Orientée Objet(MOO). Avec l'utilisation de toute la puissance des langages orientés objet, l'approche orientée objet montré ces importances et efficacités depuis ces apparition dans les systèmes complexes. Mais avec l'évolution de l'informatique, les problèmes deviennent plus complexes, de nombreuses limites de cette approche ont été trouvées. Par ailleurs, la modélisation orientée aspect a montrée son utilité dans la conception et le développement des systèmes complexes, pour cela il existe plusieurs modèles qui sont orientés aspects, mais les diagrammes UML2.0 Orientés Aspects ne possède pas de sémantique. En plus, il n'existe pas des outils qui permettent de vérifier et valider ces modèles. Donc, nous avons proposé une approche de transformation des modèles orientés aspect vers des modèles formels.
Dans ce mémoire, nous avons proposé une transformation des diagrammes de classes, état-transition et communication orientés aspect vers le langage Maude en se basant sur le paradigme de la transformation de graphe. Notre approche consiste à proposer des méta modèles (les modèles orientés aspect et Maude), une grammaire de graphe et des règles pour la transformation entre deux formalismes différents. Finalement on va argumenter notre proposition avec des études de cas bien illustrées.
Mots clés: UML 2.0 Orientés Aspect, Le langage Maude, Grammaire de graphes, AToM3 , IDM, Transformation de graphes.
Les outils:
-UML.2.0
-Python
-AToM3
-Maude

Abstract

Software modeling is the most important step in the software development process. It facilitates the understanding of the functioning of a system before its realization by producing a model. There are several modeling methods such as Aspect Oriented Modeling (MOA) and Object Oriented Modeling (MOO).With the use of all the power of object-oriented languages, the object-oriented approach has shown these importance and efficiencies since these appearances in complex systems. But with the evolution of computing, the problems become more complex, many limitations of this approach have been found. In addition, aspect-oriented modelling has shown its usefulness in the design and development of complex systems, for which there are several models that are aspect-oriented, but the UML2.0 Aspect-oriented diagrams have no semantics. In addition, there are no tools to verify and validate these models. So, we proposed an approach to transforming Aspect oriented models towards formal models.

In this memory, we proposed a transformation of aspect-oriented class diagrams, statechart and communication to the Maude language based on the graph transformation paradigm. Our approach consists in proposing Meta models (aspect-oriented models and Maude), a graph grammar and rules for the transformation between two different formalisms. Finally we will argue our proposal with well-illustrated case studies.

Keywords: UML 2.0 Aspect Oriented, Maude Language, Graph Grammar, AToM3, IDM, Graph Transformation.

The tools:

-UML.2.0

-Python

- AToM3

-Maude.

ملخص

النمذجة الحاسوبية هي أهم خطوة في تطوير البرامج. وهو يسهل فهم أداء النظام قبل إعماله عن طريق وضع نموذج. هناك العديد من طرق النمذجة مثل النمذجة جانبية التوجه (MOA) والنمذجة كائنية التوجه (MOO) .مع استخدام كل قوة اللغات كائنية التوجه ، أظهر النهج كائني التوجه اهمية اعداده منذ ظهور الأنظمة المعقدة. ولكن مع تطور الحوسبة، أصبحت المشكلات أكثر تعقيدًا، وقد تم العثور على العديد من قيود هذا النهج. بالإضافة إلى ذلك، أظهرت النمذجة جانبية التوجه فائدتها في تصميم وتطوير الأنظمة المعقدة، ولهذا هناك العديد من النماذج جانبية التوجه ، لكن المخططات جانبية التوجه UML2.0 ليس لها معنى. بالإضافة إلى ذلك، لا توجد أدوات للتحقق من صحة هذه النماذج و التأكد منها. لذلك، اقترحنا نهجًا لتحويل النماذج جانبية التوجه الى نمادج رسمية. في هذه المذكرة، اقترحنا تحويل الرسوم البيانية جانبية التوجه الى لغة Maude والاعتماد على نموذج تحويل الرسم البياني. يتكون نهجنا من اقتراح نماذج Meta (النمادج جانبية التوجه و لغة Maude)، وقواعد الرسم البياني وقواعد التحول بين شكليتين مختلفتين. أخيرًا، سنناقش اقتراحنا بدراسة حالات موضحة جيدًا.

الكلمات الرئيسية: UML 2.0 Aspect Oriented، Maude Language، Graph Grammar، AToM3، IDM، تحويل الرسم البياني.

الأدوات:

UML.2.0-
python-
AToM3-
Maude -

Table des matières

Table des figures

Introduction Générale

Introduction

Avec le temps et l'évolution technologique, les systèmes informatiques ont joué un rôle important dans de nombreux domaines tels que l'architecture, la médecine et les mathématiques... Pour concevoir le système et en vérifier la validité, il faut d'abord le modéliser, ce qui est une étape difficile.
C'est pourquoi nous nous sommes concentrés dans notre mémoire sur le système dans lequel il y a beaucoup de langages de modélisation comme UML2.0 et MERISE.
Dans notre travail, nous avons appliqué la modélisation en utilisant des diagrammes UML2.0, de sorte que depuis son apparition en 1990, il a prouvé son importance et son efficacité.
UML2.0 a treize diagrammes divisés en deux catégories statiques et dynamiques nous permettant de concevoir le système. La catégorie statique représente l'aspect statique du système, tandis que la catégorie dynamique représente l'aspect du comportement ou de la communication d'un système. Les diagrammes UML sont des modèles orientés objet (OO) Mais avec l'évolution de l'informatique ce paradigme a trouve des limites tels que : (la duplication , l'enchainement et l'éparpillant du modèle). et pour lequel n'a donné aucune solution. Pour résoudre et améliorer ces limites, les développeurs et les programmeurs ont identifié un nouveau modèle appelé Oriente Aspect(OA) qui a été classé parmi les dix nouvelles technologies.[Aouag,2014]
La modélisation orientée aspect fournit des modèles composites en séparant les modèles fonctionnels et non fonctionnels. mais les diagrammes UML2.0 Orientés Aspects a des limites ,c'est pourquoi nous avons proposé une approche de transformation des modèles orientés aspect vers des modèles formels .

La problématique

la modélisation orientée aspect a montrée son utilité dans la conception et le développement des systèmes complexes, pour cela il existe plusieurs modèles qui sont orientés aspects, mais les diagrammes UML2.0 Orientés Aspects ne possède pas de sémantique formelle. En plus, il n'existe pas des outils qui permettent de vérifier et valider ces modèles. Pour résoudre certains de ces problèmes nous proposons dans ce memoire une approche de transformation des modèles orientés aspect vers des modèles formels. Cette approche s'appuie sur le langage spécification Maude afin d'améliorer et résoudre les limites citées précédemment.Pour réaliser ce processus, nous appliquons la Transformation de modèles et plus précisément la transformation de Graphes. La transformation des graphes est effectuée par l'application de la grammaire de Graphes (la réecriture des graphes). Cette grammaire est composée d'un ensemble des règles

applicables sur le modèle source et produit automatiquement un code .

Contributions

Dans ce mémoire, nous choisissons les diagrammes UML 2.0 : de classes, état-transition et communication orientés aspect. L'idée est de transformer ces diagrammes vers le langage Maude pour résoudre les limites de la modélisation Orientée Aspect en se basant sur le paradigme de la transformation de graphe. Notre approche consiste à proposer des méta-modèles (les modèles orientés aspect et Maude), une grammaire de graphe et des règles pour la transformation entre deux formalismes différents , et par l'utilisation de l'outil ($AToM^3$). notre approche est inspirée par le travail de Chama Wafa à[Chama,2011]
Finalement on va argumenter notre proposition avec des études de cas bien illustrées.

Organisation de mémoire

Nous avons organisé ce mémoire comme la suite :
nous commencons par une introduction générale qui résume le contenu de notre travail.
Chapitre 01 : Modélisation orientée Aspect
Un aperçu sur la Modélisation orientée Aspect avec ces différent concepts de base leurs approches et leur importance, ainsi que les inconvénients des approches orientées Aspect.
Chapitre 02 : Langage maude
présentation de langage spécial Maude , qui est basé sur la logique de réécriture .ensuite les principales caractéristiques et les différents modules dont dépend.
Chapitre 03 : Transformation Des Modéles
Sera dédie à la transformation de modèles à l'aide de transformation de graphes. Nous donnons un rappel sur l'architecture dirigée par les modèle ou MDA modèle driven architecture),avec une brève présentation de la théorie de transformation de graphes ,ainsi nous présentons l'environnement de travail ($AToM^3$).
Chapitre 04 : L'approche De Transformation Proposée
Application d'une approche de transformation de graphes pour transformer les Diagrammes UML2.0 Orientés Aspect(classe, états-transitions, communication) vers le langage de spécification Maude.
Chapitre 05 : Études de cas sur la transformation des Diagrammes UML 2.0 Orientés Aspect vers le langage Maude
nous présentons quelques deux études de cas pour illustrer notre approche de transformation de modèles. Finalement, une conclusion résumant ce qui a été réalisé tout au long de notre travail.

Chapitre 1

Modélisation orientée Aspect

Introduction

Pour réussir à développer des logiciels, nous nous concentrons sur l'étape la plus importante qui est la modélisation informatique.Elle facilite la compréhension du fonctionnement d'un système avant sa réalisation en produisant un modèle. Ils existent plusieurs méthodes de modélisation comme la Modélisation Orientée Aspect(MOA) et la Modélisation Orientée Objet(MOO). La Modélisation orientée objet (MOO) travaille à résoudre la complexité des problèmes mais connaît des limites avec l'évolution de l'information comme la duplication, l'enchainement et l'éparpillant du modèle.
C'est pourquoi les développeurs ont identifié le nouveau paradigme appelé "modélisation orientée aspect". Dans ce chapitre nous commençons par des définitions de base de modéle et la modélisation avec ces différents types. Par la suite, nous présentons les limites de l'approche orientée objet ,et quelques concepts de base de la modélisation orientée aspect. On termine par la modélisation graphique d'un aspect par des diagrammes UML2.0 .

1.1 Modèle

Un modèle est une abstraction d'un système construite dans un but précis. On dit alors que le modèle représente le système. Un modèle est une abstraction dans la mesure où il contient un ensemble restreint d'informations sur un système. Il est construit dans un but précis et les informations qu'il contient sont choisies pour être pertinentes vis à vis de l'utilisation qui sera faite du modèle.[Muller,2006]

1.2 Modélisation

La modélisation est le processus de spécification du modèle, elle consiste à représenter le système comme des modèles utilisant des concepts prédéfinis dans un langage de modélisation. La modélisation est l'une des tâches les plus importantes du processus de développement du système. La phase d'analyse peut être considérée comme plus stratégique que la phase de conception et de mise en œuvre. Il est nécessaire de représenter, de comprendre et d'identifier les exigences du système afin de concevoir et de mettre en œuvre une application stable et efficace. Avec l'augmentation de la complexité des systèmes à élaborer, le choix d'une méthode de développement appropriée

se révèle primordial pour le succès des travaux. Pour cela, il existe plusieurs orientations disponibles, c'est-àdire des approches différentes pour comprendre, représenter, analyser et concevoir un système. Le problème sera décomposé en plusieurs modèles différents liés entre eux.[Wauteletet al,2004]

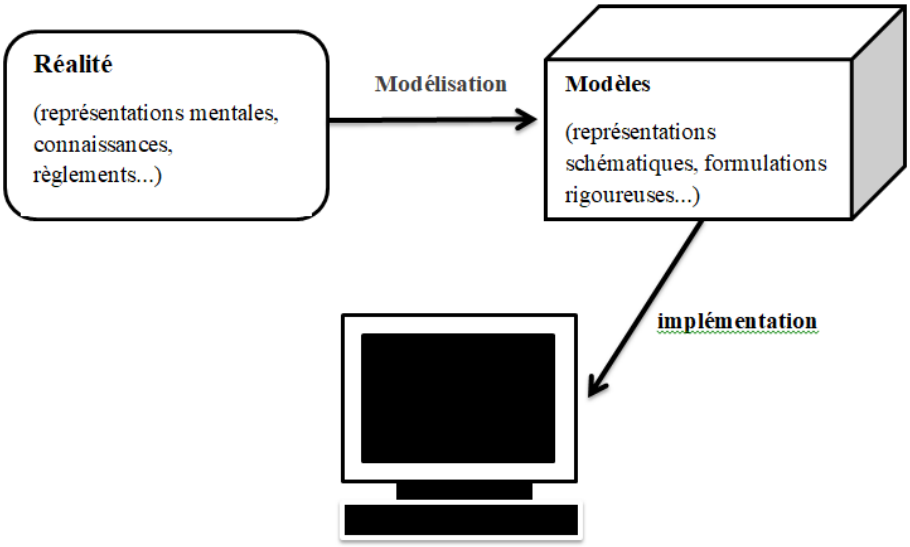


FIGURE 1.1 – Représentation de modélisation.

1.3 Objectif de la modélisation

La modélisation permet de mieux comprendre le système à développer. Il permet donc :[Terchi et Mezahi,2018]
• Visualiser le système tel qu'il est ou devrait être.
• Valider le modèle auprès des clients.
• Préciser les structures de données et le comportement du système.
• Fournir un guide pour la construction du système.
• Documenter le système et les décisions prises.

1.4 Les types de modélisation

La modélisation peut être classée selon le degré de formalisme des langues ou des méthodes utilisées. Ainsi, la modélisation peut être considérée comme formelle, semi-formelle ou informelle. La table ci-dessous présente une défnition des catégories de langages ainsi que des exemples de langages ou de méthodes qui l'utilisent : [Dehimi,2014]

1.4.1 Modélisation informelle

Le langage informel est une façon familière de communiquer entre les gens. D'autre part, l'utilisation d'un langage informel rend la modélisation imprécise et parfois ambiguë. Toute tentative de normalisation est difficile, étant donné la nature informelle de

Catégories de Langages			
Langage Informel		**Langage Semi-Formel**	**Langage Formel**
Simple	*Standardisé*		
Langage qui n'a pas un ensemble complet de règles pour restreindre une construction	Langage avec une structure, un format et des règles pour la composition d'une construction.	Langage qui a une syntaxe définie pour spécifier les conditions sur lesquelles les constructions sont permises.	Langage qui possède une syntaxe et une sémantique définies rigoureusement. Il existe un modèle théorique qui peut être utilisé pour valider une construction.
Exemples de Langages ou Méthodes			
Langage Naturel.	Texte Structuré en Langage Naturel.	Diagramme Entité-Relation, Diagramme à Objets.	Réseaux de Petri, Machines à états finis, VDM, Z.

FIGURE 1.2 – Classification et utilisation de langages ou de méthodes.

cette approche. Néanmoins, il est possible d'utiliser une modélisation informelle plus ou moins standardisée pour limiter ce problème, c'est-à-dire un modèle qui utilise un langage naturel tout en introduisant des règles pour l'utilisation de ce langage dans la construction de la modélisation.
Une telle modélisation conserve les avantages de la modélisation informelle en la rendant moins imprécise et moins ambiguë.

1.4.2 Modélisation semi-formelle

Le processus de modélisation semi-formel est basé sur un langage textuel ou graphique pour lequel une syntaxe précise est définie. Ce type de modélisation permet d'effectuer des contrôles et des automatismes pour certaines tâches, bien que la sémantique des langages semi-formels soit souvent assez faible.La majorité des méthodes de modélisation semi-formelles reposent fortement sur les langages graphiques. Ceci est justifié par l'expressivité qu'un modèle graphique bien développé peut avoir. Le langage texte est normalement utilisé en complément des graphiques.
De plus, la modélisation semi-formelle (comme : UML) utilise fortement les langages graphiques, ce qui permet la production de modèles assez faciles à interpréter. Néanmoins, les aspects sémantiques impliqués dans cette approche de modélisation souffrent d'une lacune remarquable.

1.4.3 Modélisation formelle

Les méthodes formelles sont basées sur des techniques mathématiques pour la spécification, le développement et la validation des systèmes. L'utilisation de méthodes formelles est importante pour assurer la sûreté du système. Les études pour la création d'autres outils formels qui couvrent l'ensemble de la spécification, en plus des outils de preuve existants, montrent l'intérêt pour ces méthodes. Bien qu'ils ne soient pas utilisés dans tous les types de modélisation, l'application de méthodes formelles, plus que souhaitable, commence à être imposée par les entreprises dans certains cas, qui exigent plus de rigueur.

1.5 Modélisation orientée objet

La modélisation orientée objet a un rôle importante dans la conception et la modélisation de systèmes complexes. La modélisation et la conception orientée objet constitue une façon de penser les problèmes en appliquant des modèles organisés autour de concepts du monde réel. Le concept fondamental est l'objet qui combine à la fois une structure de données et un comportement. Les modèles orientés objet permettent de comprendre des problèmes, de communiquer avec des experts du domaine d'application, de modéliser les métiers d'une entreprise, de réparer la documentation et de concevoir des programmes et de bases de données.[Aouag,2014]

1.5.1 Limites de l'approche orientée objet

Le modèle orienté objet a introduit les concepts et les outils les plus puissants pour créer des logiciels complexes, en plus d'utiliser les graphismes de conception les plus avancés.
Malgré cela, ils restent dans nos applications des éléments de modèles qu'il n'est pas possible de centraliser, de prendre en compte.. l'objet-orienté a trouvé des limites pour lesquelles ce paradigme n'a donné aucune solution telle que (la duplication, l'enchaînement du modèle, la difficulté de réutilisation des modèles et le problème de l'interaction entre les modèles fonctionnels et non fonctionnels). aussi :[Aouag,2014]
• Le modèle de résolution est difficile à lire et à comprendre et peut causer des erreurs.
• L'objet ne peut pas être supprimé.
• Une duplication des fonctionnalités interfonctionnelles du modèle applicatif.
• Il n'y a pas d'enchaînement De modèle.
• La réutilisation des modèles est complexe, de sorte que l'aspect est utilisé pour améliorer et résoudre les limites du paradigme orienté objet.

1.6 Modélisation orientée aspect (MOA)

1.6.1 Définition de la modélisation orientée aspect

La modélisation orientée aspect permet la séparation entre un modèle métier (fonctionnel) et un modèle technique (non fonctionnel), et utilise un mécanisme d'intégration (Weaver) pour obtenir le modèle intégré (métier et aspect). Pour cela, plusieurs travaux ont été proposés pour définir l'aspect (en tant qu'entité) dans les phases d'analyse et

de conception. [Aouag,2014]
La principale difficulté des techniques de modélisation orientées aspect est le tissage de modèles aspect qui permet de construire le modèle complet. Le processus de tissage qu'il contient le modèle de base correspond au modèle dans lequel l'aspect doit être tissé.
Un aspect est composé de deux parties : un point de coupure et un conseil. Le point de coupure décrit les structures ou comportements de base du modèle qui doivent être modifiés et le conseil précise la structure ou le comportement qui doit être composé dans le modèle de base. Pendant la phase de détection, le point de coupure est utilisé pour trouver tous les points du modèle de base sur lesquels le conseil doit être composé pendant la deuxième phase.[Terchi et Mezahi,2018]
En général, le cycle d'élaboration du protocole d'entente comporte trois étapes :[Aouag,2014]
1. Décomposition aspectuelle : Il consiste à décomposer les besoins et à identifier et séparer les modèles transversaux encapsulés dans des aspects (techniques, non fonctionnels) et des modules métiers (de base, fonctionnels) qui peuvent être modularisés en modules de base tels que la classe.
2. implementation des modèles : Elle consiste à implementer chaque modèle séparément. Les modèles métiers sont implementes par les techniques conventionnelles de la modélisation orientée objet alors que les modèles transverses sont implantés par les techniques de la modélisation orientée aspect.
3. Recomposition aspectuelle : Il consiste à construire le système final en intégrant ou superposant des modèles d'affaires avec des modèles transversaux. Cette phase est appelée tissage. Un tisseur (weaver) utilise les règles spécifiées par le concepteur de l'application afin de croiser correctement les modèles. Le tissage d'aspect est une opération qui accepte les modules de base et d'aspect comme entrée. Leur but est l'application et la fixation des aspects dans les modules de base selon les points de jonction correspondant à la spécification des points de coupure d'aspect.
Donc un aspect = les points de coupure + Conseil.
les points de coupure= $\sum$ les points de Jonction.
On peut distinguer deux types de tissage : le tissage statique (avant l'exécution) et le tissage dynamique (durant l'exécution).

1.6.2 Les approches orientée aspect

Nous avons mentionné plusieurs approches sont :[Boubendir,2011]

- Ingénierie des exigences orientée aspect

Les approches fournissent une représentation des préoccupations interfonctionnelles dans les artefacts d'exigences. Ils reconnaissent explicitement l'importance d'identifier et de traiter les préoccupations interfonctionnelles dès le début, les préoccupations interfonctionnelles peuvent être des exigences non fonctionnelles ainsi que des exigences fonctionnelles, leurs identifications précoces permettent une analyse précoce de leurs interactions. Ces approches mettent l'accent sur le principe de la composition de toutes les préoccupations pour que le système complet soit construit . Il est donc possible de comprendre les interactions et les compromis entre les préoccupations. La composition des exigences permet non seulement d'examiner les exigences dans leur ensemble, mais

aussi de détecter les conflits potentiels à un stade précoce, afin de prendre les mesures correctives ou les décisions appropriées à l'étape suivante.

- Les approches d'architecture orientée aspect

Un aspect architectural est un module architectural qui a une grande influence sur d'autres modules architecturaux. Les approches de conception architecturale orientées aspect décrivent donc les étapes pour identifier les aspects architecturaux et leurs composants enchevêtrés, cette information est utilisée pour refaire une conception architecturale donnée tout en explicitant les aspects architecturaux. Ceci est différent des approches traditionnelles où les aspects architecturaux sont des informations implicites dans la spécification de l'architecture. L'étude réalisée dans donne un aperçu assez exhaustif de ces approches.

- Les approches de conception orientées aspect

Ces approches de conception se concentrent sur la représentation explicite des préoccupations transversales en utilisant des langages de conception adéquats. Au début, les concepteurs utilisaient simplement des méthodes et des langages orientés objet (comme UML) pour la conception de leurs aspects. Cela s'est avéré difficile puisque l'UML n'était pas conçue pour fournir des constructeurs pour décrire les aspects, de sorte que la principale contribution de la conception orientée aspect était de fournir aux concepteurs des moyens explicites de modéliser les systèmes par aspect.
Dans notre memoire nous nous sommes basés sur cette approche.

- Les approches de vérification et de test des programmes orientes aspect

L'approche soulève de nouveaux défis dans les techniques de vérification et de validation des logiciels pour s'assurer que le système répond aux fonctionnalités souhaitées. Les aspects peuvent potentiellement endommager la fiabilité d'un système auquel ils sont tissés, et peuvent invalider les propriétés essentielles du système qui étaient correctes avant le tissage d'aspect. Pour assurer l'exactitude du logiciel par aspects, il y a beaucoup de recherche sur l'utilisation de méthodes formelles et techniques de test spécialement adaptées aux aspects.

- L'intergiciel orienté aspect

Bien que l'intergiciel n'est pas une étape dans le cycle de vie, il est un domaine important et vaste pour les idées orientées aspect. Beaucoup de développeurs de logiciels ont adopté des approches d'intergiciel pour aider à la construction de systèmes distribués à grande échelle. L'intergiciel facilite le développement de systèmes logiciels distribués en supportant l'hétérogénéité, cachant les détails de distribution et fournissant un ensemble de services spécifques pour un domaine commun.[Brichau et D'Hondt,2005]

1.6.3 Pourquoi l'approche orientée aspect ?

L'approche de l'aspect a été utilisée pour les premières années dans les langages de programmation en phase de codage, le paradigme orienté aspect n'est plus limité au niveau de programmation, et il s'étend maintenant aux phases en amont du

développement logiciel. . Des approches axées sur les aspects sont maintenant disponibles à chaque phase du développement logiciel : analyse des besoins, conception, voire mise en œuvre. Passer d'une phase à l'autre tout en conservant les aspects identifiés au préalable reste un défi majeur, mais peu étudié. Une approche itérative axée sur les préoccupations pour transformer automatiquement un modèle d'exigences axé sur les aspects en un modèle de conception axé sur les aspects. [Aouag,2014]

1.6.4 Les concepts de base

• ***point de jonction :*** Un point de jonction est un endroit dans le modèle d'aspect où un ou plusieurs aspects seront utilisés. Ce point de jonction est déterminé lors de la composition par le révélateur. L'utilisation d'un aspect pour un point de jonction se fait dynamiquement, i.e. une fois que le modèle est en cours d'exécution (weaver). Il existe plusieurs types de points de jonction, selon ce que le développeur veut intercepter. Ainsi, un point de jonction peut intervenir pour :[Otmane Rachedi,2015]
-Une classe.
- Une interface.
- L'exécution d'une méthode.
- Accès à une variable de classe.

• ***point de coupure :*** une coupe désigne un ensemble de points de jonction. Une coupe est définie à l'intérieur d'un aspect. Dans le cas simple, une seule coupe est suffisante pour définir la structure transversale d'un aspect, dans les cas plus complexes un aspect est associé à plusieurs coupes. Les notions de coupure et de point de jonction sont liées par leur définition. Cependant, leur nature est très différente. Une coupure est un choix de points de jointure définis dans un aspect, tandis qu'un point de jointure est un emplacement dans l'exécution d'un modèle. Par conséquent, un point de jonction peut appartenir à plusieurs sections dans le même aspect ou dans différents aspects.[Amroune,2014]

• ***Conseil (advice en anglais)*** Un conseil représente un comportement technique particulier d'un aspect. Concrètement, c'est un modèle qui sera greffé dans l'application lors de l'exécution d'un point de jonction défini dans une section. Un aspect peut contenir une ou plusieurs instructions. De même qu'un aspect peut être comparé à une classe de modélisation orientée objet, un modèle d'instruction peut être comparé à une méthode d'aspect. Comme nous l'avons vu précédemment, une section peut contenir plusieurs points de jonction, qui peuvent eux-mêmes faire référence à plusieurs aspects. Une instruction est donc susceptible d'être utilisée à plusieurs endroits dans l'application et un point de jonction peut éventuellement greffer plusieurs instructions au moment de l'interception.[Otmane Rachedi,2015]

• ***Tisseur (weaver en anglais) :*** Le tissage est une opération automatique absolument nécessaire pour obtenir une application opérationnelle, cette étape est utilisée pour intégrer les fonctionnalités des classes et celles des aspects. Le programme qui fait cela est appelé un tissage d'aspect. L'application obtenue après le tissage est dit être tissé.[Otmane Rachedi,2015]
la figure 1.3 qui explique le processus de modélisation axée sur les aspects tels que : E1, E2 et E3 sont l'état, la vérification et la sécurité sont des aspects.

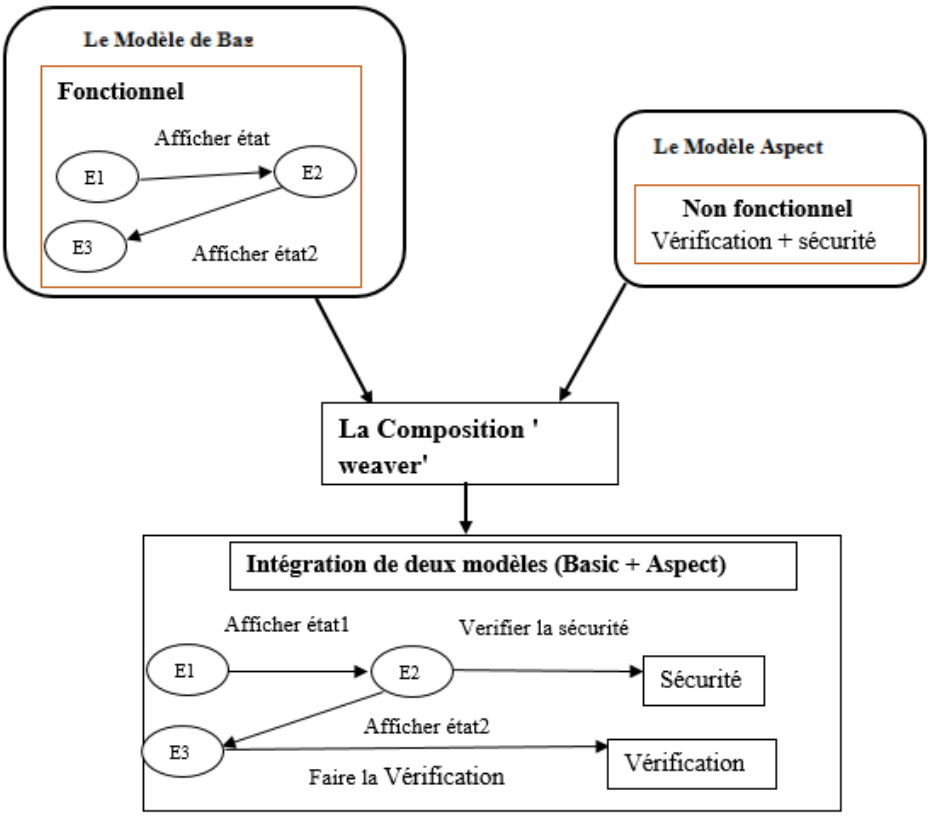


FIGURE 1.3 – Le processus de modélisation orientée aspect.

1.6.5 Les inconvénients de la MOA

La modélisation orientée aspect ne peut être élégamment appliquée à toutes les situations de problèmes possibles en plus des limites de MOA ce qui suggère que [Aouag,2014] :
- La gestion des transactions, par la coupe transversale, est difficile à factoriser dehors dans un aspect distinct.
- La MOA est surtout adaptée pour les projets de développement de logiciels à grande échelle.
- Dans les systèmes distribués la MOA apporte avec elle certaines difficultés en ce qui concerne les tests et le débogage. Cela est dû à des effets secondaires qui se dégagent de la dynamique injection du modèle et qui pourraient, dans le pire des cas, conduire à des ambiguïtés sémantiques dans le flux de contrôle d'un modèle orienté aspect .
- Les différents aspects peuvent effectivement nuire même aux points de jonction dans le tissage. Ainsi, la MOA peut violer le principe d'encapsulation, bien que d'une manière assez systématique et bien contrôlée.

Conclusion

Dans ce chapitre nous avons présenté brièvement la déffinition de la modélisation des systèmes avec ces diffiérents types, plus particulièrement les limites de la modélisation orientée objet.la notion de la modélisation orientée aspect qui permet d'améliorer et de donner des solutions au inconvénients (problèmes) de l'approche orientée objet, nous avons vu la définition de la modélisation orientée aspect .Par la suite nous avons cité quelques approches de la modélisation orientée aspect. Le chapitre suivant (chapitre 2) va introduire une spécification formelle est le langage Maude et les concepts de base de ce langage qui basée sur une logique de réécriture, Ce qui exprime une étape essentielle pour la transformation des diagrammes oriente aspect .

Chapitre 2

langage maude

Introduction

La spécification du logiciel représente une activité extrêmement importante dans le développement de logiciels de qualité. Cette activité peut se faire de façon informelle, semi-formelle ou formelle. Chacune de ces approches présente des avantages et des inconvénients, les deux dernières étant les plus utilisées dans la littérature. L'approche formelle nous permet d'automatiser les activités de vérification et de validation des systèmes au moyen de spécifications formelles.Plusieurs langues de spécifications formelles ont été proposées dans la littérature, par exemple le langage Maude . Maude est un langage basé sur une logique solide et complète appelée la logique de réécriture, qui est considérée comme un cadre unificateur de plusieurs modèles formels exprimant la concurrence. [J. Meseguer. 1992] . Maude prend en charge la spécification et la programmation de systèmes concurrents, représente un système puissant avec des caractéristiques très intéressantes, telles que :
simplicité de syntaxe, expressivité et haute performance, ce qui nous donne la possibilité de créer des cadres logiques pour la modélisation et la vérification du système.Ce chapitre est consacré à la présentation de la logique de réécriture et les notions fondamentales du système Maude.

2.1 La logique de la réécriture

La réécriture est un paradigme général d'expression numérique dans différentes logiques de calcul. Le calcul consiste à réécrire des règles dans une syntaxe donnée. Dans la logique de réécriture, Une réécriture d'un terme est de le remplacer par un terme équivalent, selon les lois des termes algébriques.Cette logique a été présentée par José Meseguer 1992 dans le laboratoire ISR aux États-Unis, à la suite de ses travaux sur les logiques générales pour décrire les systèmes concurrents.
Cette logique est proposée comme un cadre dans lequel d'autres logiques peuvent être représentées et aussi comme un cadre sémantique pour la spécification des langages et des systèmes.
La logique de réécriture est appliquée dans plusieurs domaines tels que : spécification et vérification des logiciels et du matériel, sécurité et déduction automatique. Son pouvoir est d'exprimer à la fois la concurrence et la logique de déduction. [Nouri,2020]

2.1.1 Théorie de réécriture

Une théorie équationnelle de l'appartenance ($\sum$, E) est décrite dans Maude par un module fonctionnel sachant que $\sum$ est une signature algébrique et E représente un ensemble d'équations. Les modules fonctionnels fournissent des modèles exécutables pour les théories équationnelles spécifiées. Une théorie de réécriture $\hat{A} = (\sum , E, R)$ est spécifiée dans Maude par un module système ou bien orienté objet. Un module système peut inclure à la fois une représentation fonctionnelle de la théorie équationnelle ($\sum$, E) et la spécification des règles de réécriture dans R. [Nouri,2020]
L'application des règles de réécriture est non déterministe, ce qui fait de la logique de réécriture un bon candidat pour la modélisation de la concurrence.

2.2 Définition de Maude

Maude est un langage déclaratif haute performance autour duquel est construit un système haute performance. Les programmes Maude sont des théories de réécriture et les calculs concurrents à Maude représentent des déductions dans la logique de réécriture.
système Maude est également considéré comme un méta-outil formel, grâce à sa capacité à construire d'autres outils, y compris ceux dans son propre environnement. Le langage Maude peut également être utilisé comme système de vérification. Les objectifs du projet Maude sont de soutenir les spécifications formelles exécutables et d'élargir le spectre d'utilisation de la programmation déclarative, en spécifiant et en mettant en œuvre des systèmes de haute qualité dans des domaines tels que : le génie logiciel, les réseaux de communication, l'informatique distribuée, la bioinformatique et le développement d'outils officiels .[Douibi,2006]
Maude a des caractéristiques très importantes comme la simplicité, la puissance et bien d'autres :[Douibi,2006]
- Maude est simple :La programmation de Maude est très simple et facile à comprendre.
Il y a des équations et des règles, et il a dans les deux cas une sémantique de réécriture simple dans laquelle les instances du côté gauche sont remplacées par les instances correspondantes du côté droit. Maude a un ensemble de mots-clés et de symboles, et quelques lignes directrices et conventions à maintenir, donc les programmes Maude sont aussi simples et faciles à comprendre .
- Maude est expressif : Il est possible d'exprimer un très grand nombre d'applications naturellement à Maude, à commencer par les systèmes déterministes arrivant à des systèmes concurrents. Il offre également un cadre sémantique, dans lequel non seulement des applications, mais des formalismes entiers (autres langages, autres logiques) peuvent être naturellement exprimés.
- Maude est puissant : il est possible d'utiliser ce langage non seulement pour les spécifications exécutables et le prototypage système, mais aussi pour la programmation et le développement réels.
-Multi-paradigme : Il combine la programmation déclarative et orientée objet. Dans Maude, les spécifications non exécutables sont appelées théories, et leurs axiomes sont utilisés pour imposer des exigences formelles aux modules de programme et aux interfaces des modules paramétrés.

2.3 Modules dans Maude

dans Maude Pour assurer une bonne description modulaire, il propose trois types de modules :[Chama,2011]
Modules fonctionnels,Modules système,Modules orientés objet (uniquement en Maude).
1- Modules fonctionnels :Les modules fonctionnels sont des théories fonctionnelles dans la logique de réécriture, ces modules définissent les types de données et les opérations qui leur sont applicable en termes d'équation.
2- Modules systèmes :Les éléments des modules systèmes sont les même que celles des modules fonctionnelles à l'exception des règles de réécritures.Les règles de réécritures modélisent le comportement dynamique du système.Deux types des règles : inconditionnelles / conditionnelles.
3- Modules orientés-objets :Full Maude est un prototype Maude qui supporte la spécification orientée-objets, ces derniers offrent la syntaxe nécessaire pour déclarer les classes, les sous classes .

2.4 Les versions du Maude

Le système Maude a deux versions :

2.4.1 Core Maude

Core Maude étant l'interprète du Maude implémenté en C++, il offre toutes les fonctionnalités de base de Maude et accepte uniquement une hiérarchie de modules fonctionnels et de systèmes qui ne sont pas configurés. Core Maude intègre d'autres modules tels que : les modules prédéfinis de types de données et les techniques de vérification de modèle, la réflexivité, le métalangage et le focus sont pris en charge par Core Maude. [Douibi,2011]

a. Concepts de base

a.1 Déclarations des sorts et de sous-sorts :
Les types de données (Sort) sont les premiers à déclarer dans une spécification algébrique. Un sort est déclaré en utilisant le mot-clé sort suivi du nom du sort comme suit : sort (le nom du sort). et pour éviter d'avoir des erreurs, ne peut contenir un espace, un point, une virgule, ou l'un des caractères suivants :'–', '(', '[' et '', sauf si elle est précédée d'une apostrophe (' ' '). Le point (.) à la fin de la déclaration de sort , est important.on peut déclarer un ou plusieurs sort s et une hiérarchie de sort avec les mots clés : sort, sorts, subsort et subsorts . Par exemple, nous pouvons déclarer Nat et Zero comme suit :
sort Nat .
sort Zero.
Ou :
Sorts Nat Zero.
a.2 Déclarations des variables :
Une variable dans Maude est déclarée avec le mot clé var et avec vars dans le cas de plusieurs variables, par exemple :

var A : Nat .
vars A W : Nat .

a.3 Déclaration des operateurs :

Dans un module Maude, les opérations sont déclarées par le mot-clé op et ops suivi du nom de l'opération, suivi de la liste des grades formant ses arguments (l'arité de l'opérateur) selon la syntaxe générale suivante : op (operator name) : (sort-1) ... (Sort-k) $- >$(Sort).
Une constante dans Maude est un opérateur avec une liste vide d'arguments. Par exemple :
op 0 :$-$ $>$ Zero .
Les operateurs dans Maude peuvent être utilisées selon les deux préfixes et mixfixe. Le premier formulaire Prefix correspond mathématiquement à $f(x)$.Dans la deuxième forme mixfixed, nous remarquons que les variables doivent être déclarées dans des emplacements spécifiques ,le caractère $(-)$ joue un rôle particulier dans le nom de l'opérateur. Par exemple, l'opérateur d'addition en notation préfixée $(op+)$ en mixfixe serait $(op-+-)$, donc la syntaxe de la déclaration sera la suivante :
$op+ : Nat- > Nat.$
$op-+- : NatNat- > Nat$. [Nouri,2020]

a.4 Déclaration Les Termes :

Un terme est une constante, une variable, ou l'application d'une opération à une liste de termes d'argument. Le type d'une constante ou d'une variable est son type déclaré.avec une liste d'arguments selon l'un des deux préfixes ou notations mixfixed.Par exemple :
Selon le préfixe : $-+-(N : Nat, M : Nat)$.
Selon la notation de mix fixe :$N : Nat + M : Nat.$

a.5 Équations inconditionnelles :

Les équations sont déclarées en utilisant le mot-clé eq, suivi par un terme (la partie gauche), le signe d'égalité, puis un terme (la partie droite), et optionnellement suivi par des attributs encadrés par des crochets, terminés par un espace et un point. Ainsi le schéma général est le suivant :
$eq < Term1 >=< Term2 > [< StatementAttributes >]$.
Les termes t et t' dans une équation t = t' doivent être de même sorte. Pour que l'équation s'exécute, chaque variable qui apparaît dans t' doit apparaître dans t. Nous pouvons par exemple ajouter des équations relatives à l'addition dans le module NUMBERS :
vars N M : Nat .
eq N + zero = N .
eq N + s M = s (N + M) .
Avec s défini précédemment comme opérateur unaire.

a.6 Équations conditionnelles :

Les conditions dans les équations conditionnelles se composent de différentes équations t = t'. Une condition peut être une équation simple, ou une conjonction d'équations en utilisant le ''and ''qui est associative.
Ainsi la forme générale des équations conditionnelles (ceq) est la suivante :
$ceq < Term1 >=< Term2 >$.
if $< EqCondition - 1 >$ and ... and $< EqCondition - k > [< StatementAttributes >]$
.
En plus, la syntaxe concrète de la condition équationnelle "EqCondition" est peut être une équation booléenne dite abrégée de la forme t, avec t un terme dans la sorte Bool

abrégeant l'équation t = true.
Ainsi, ces conditions peuvent apparaître dans une équation :
(N == zero) = true
(M =/= s zero) = true
(N > zero or M =/= s zero) = true . [Mérouani,2011]

a.7 Les Règles non conditionnelles :

Les règles non conditionnelles sont déclarées avec la syntaxe :
rl [$<$ *etiquette* $>$] : $< Term1 >=>< Term2 >$.
Une règle de réécriture peut avoir une étiquette qui appelle l'action ou l'événement qui change l'état.[Mérouani,2011]

a.8 Règles conditionnelles de réécriture : [Mérouani,2011]

Les règles conditionnelles de réécriture peuvent avoir des conditions très générales impliquant des équations et autre réécritures.Dans leur représentation en Maude, les règles conditionnelles sont déclarées avec la syntaxe :
crl [$<$ *etiquette* $>$] : $< Term1 >=>< Term2 >$ if $< Condition - 1 >$ and .. and $< Condition - k >$.[Mérouani,2011]

2.4.2 Full Maude

La fonctionnalité Core Maude était disponible lors de la première sortie de Full Maude. En fait, Maude n'est pas seulement un complément à Core Maude, mais un moyen d'expérimenter de nouvelles fonctionnalités puissantes, telles que le paradigme de développement orienté objet et la possibilité de définir des vues en tant que modules. Full Maude fournit également un module générique pour les n-uplets.
En tant que tel, Full Maude est un peu difficile et nécessite une certaine expertise et de la patience pour l'utiliser efficacement .
En fait, pour utiliser Full Maude, toutes les commandes et tous les modules doivent être entrés entre parenthèses. Le module de bibliothèque standard LOOP-MODE (mode boucle), est le module de base dans Full Maude, car il offre au programmeur les outils pour créer un (sous-environnement) dans Maude, avec une interface utilisateur et la possibilité de définir ses propres commandes en plus de celles fournies par l'environnement Maude[Nouri,2020].
En fait, l'utilisation du mode boucle nous permet de créer des systèmes de réécriture complexes et puissants et même des langages de programmation entiers. Full Maude peut également être utilisé comme plate-forme pour construire d'autres outils, tels que : l'outil Real-Time Maude pour la spécification et l'analyse des systèmes en temps réel et l'outil Maude MSOS pour la sémantique opérationnelle structurelle modulaire. Full Maude utilise des modules orientés objet (qui peuvent également être configurés) basés sur les notions d'objet, de message, de classes et d'héritage. En fait, Full Maude offre deux outils principaux : ***modules orientés objet et Le paramétrage.*** [Nouri,2020]

A. Les modules orientés objet

Dans Full Maude, les systèmes orientés objet peuvent être définis de manière compétitive à l'aide de modules orientés objet. présentés par le omod keyword... Endom .En utilisant une syntaxe plus pratique, Maude est comme suit :
(omod $< ModuleName >$ ***is ... endom).***
Le module CONFIGURATION est implicitement inclus dans tous les modules orientés objet, il contient des déclarations des types suivants :
Oid (identifiants d'objet), Cid (identifiants de classe), Object pour les objets, et Msg

pour les messages.[Nouri,2020]

B. Le paramétrage

Le paramétrage est un outil très puissant mais aussi très complexe et nécessite une syntaxe spéciale.

Les modules orientés objet, comme tout autre type de module, peuvent être configurés et également instanciés par des traces appelées vues. Les éléments de base de la programmation paramétrée sont : théories, modules paramétrés et vues. Les théories elles-mêmes peuvent être paramétrées. Les paramètres sont des théories représentant les exigences sémantiques pour une instanciation correcte. Un module paramétré est un module avec un ou plusieurs paramètres, les modules peuvent être paramétrés via une ou plusieurs interfaces appelées théories qui définissent les règles des paramètres. Une vue vise à faire correspondre les sorts et les opérations définis dans la théorie avec ceux d'un paramètre donné, ce qui établit un lien entre le paramètre et ses règles.[Nouri,2020]

Conclusion

Dans ce chapitre, nous avons commencé par introduire les concepts de la logique réécriture , cette dernière représentant la base du langage Maude, tout en précisant la puissance de cette logique comme cadre sémantique pour la spécification des langages et des systèmes. Ensuite, nous avons introduit Maude comme langage déclaratif citant les caractéristiques essentielles de Maude et son utilisation comme langage de programmation et de spécification formelle et aussi comme système de vérification. Pour comprendre le système Maude, nous avons décrit ses différents modules : le module fonctionnel, le module système et le module orienté objet, en précisant le contenu de chaque module. Ensuite, nous avons indiqué que Maude offre deux versions : Core Maude et Full Maude.À partir du premier niveau qui est l'interprète autour duquel le système Maude est construit, car il fournit toutes les fonctionnalités de base. Enfin, nous avons présenté les principales caractéristiques du niveau Full Maude basées sur les modules orientés objet et le concept de paramétrage.
Les concepts présentés dans ce chapitre fournissent un contexte nécessaire pour comprendre de mécanisme pour contribuer à transformation graphique des daigramme UML2.0 orienté aspect vers Maude, qui nécessite en plus des mécanisme et des outils pour représenter la transformation des modèles , à présenter dans le chapitre suivant (chaptre 3).

Chapitre 3

Transformation De Modéles

Introduction

Au cours des dernières décennies, le développement de grands projets d'ingénierie, toujours plus complexes, a mis en évidence la nécessité de disposer d'outils, de méthodes et de processus permettant d'en assurer la maîtrise tout au long de leur cycle de vie. L'Ingénierie Dirigée par les Modèles (IDM), ou Model Driven Engineering (MDE) est une discipline récente du génie logiciel qui met l'accent sur les modèles au sein du processus de développement logiciel. L'IDM est donc le domaine de l'informatique mettant à disposition des outils, concepts et langages pour créer et transformer des modèles.
La transformation des modèles est une tâche complexe qui nécessite la disposition d'outils flexibles permettant la gestion des modèles et des langages durant la transformation et la manipulation des modèles les transformations permettent de choisir l'espace technique et le formalisme le plus adapté à chaque activité.
Dans cette mémoire, nous nous intéressons aux concepts de transformation de modèles et plus particulièrement à la transformation de graphes en utilisant l'outil ($AToM^3$) . Les diagrammes d'UML 2.0 Orientés Aspect (comme un modèle d'entrée) seront transformés vers le langage Maude(comme un modèle de sortie).
Le but de ce chapitre est de donner une idée sur la transformation de modèles. En suite nous présentons les différents types de transformations ,notamment la transformation de graphes, les grammaires de graphes et un aperçu sur l'outil utilisé ($AToM^3$).

3.1 L'architecture dirigée par les modèles

L'architecture dirigée par les modèles ou MDA ("Model Driven Architecture") est une démarche de réalisation de logiciel, proposée et soutenue par l'OMG. La figure 3.1 représente les différentes couches de spécification de la démarche MDA.

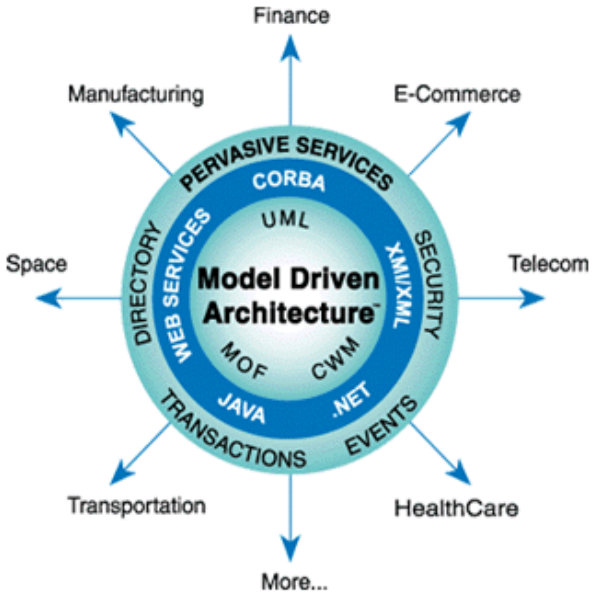


FIGURE 3.1 – Model Driven Architecture (MDA)

3.1.1 définition des Langage de modélisation , Méta-modèle et le Méta-Méta-modèle

1- Langage de modélisation : est défini par une syntaxe abstraite, une syntaxe concrète et une sémantique. La syntaxe abstraite définit les concepts de base du langage. La syntaxe concrète définit le type de notation qui sera utilisé pour chaque concept abstrait qui peut être graphique, textuelle ou mixte. Enfin, la sémantique définit comment les concepts du langage doivent être interprétés par les concepteurs mais surtout par les machines.
2- 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, c'est-à-dire une spécification de la syntaxe du langage.[Aouag,2014]
3- Méta-Méta-modèle : Le méta-méta-modèle est un méta-modèle pour les méta-modèles utilisé tout naturellement pour désigner ce méta modèle particulier. Le langage utilisé au niveau du méta-méta-modèle doit être suffisamment puissant pour spécifier sa propre syntaxe abstraite et ce niveau d'abstraction demeure largement suffisant (métacirculaire). Chaque élément du modèle est une instance d'un élément du méta-modèle.[Aouag,2014]

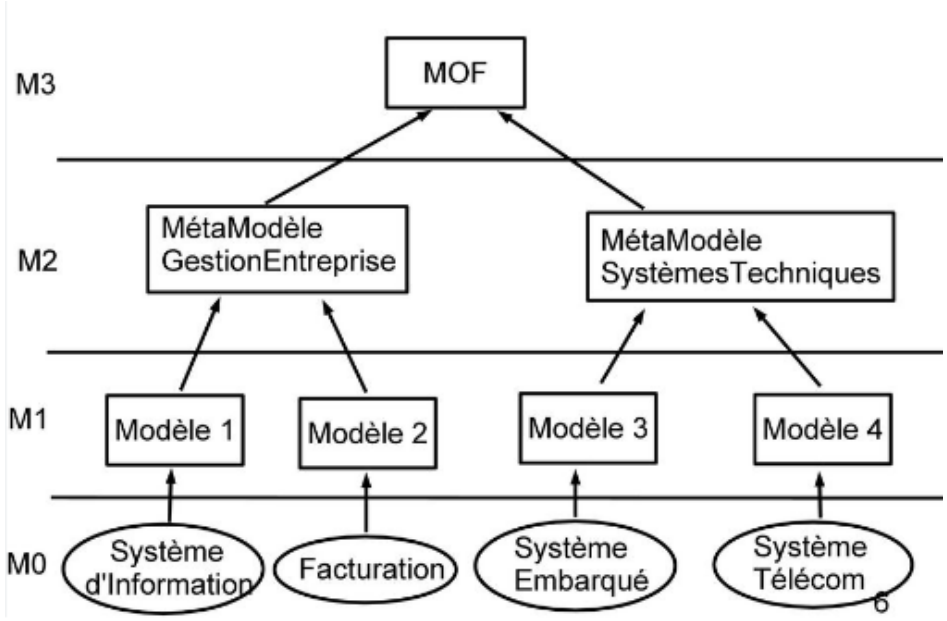


FIGURE 3.2 – Relation entre systéme,modèle,métaModéle et méta-métaModéle

3.1.2 Principe du MDA

L'approche MDA (Model Driven Architecture) Il s'agit de l'une des approches disponibles dans la littérature.Il permet de réaliser des logiciels et d'améliorer considérablement le processus de développement.des systèmes complexes. Il est proposé et pris en charge par l'OMG en 2000. basée sur les principes suivants :
- modèles d'exigences (CIM : Computation Independent Model).
- d'analyse et de conception (PIM : Platform Independent Model).
- de code (PSM : Platform Specific Model). [Aouag,2014]

A.Modèle des besoins CIM :

La première chose à faire lors de la construction d'une nouvelle application est bien entendu de spécifier les exigences du client. Bien que très en amont, cette étape doit fortement bénéficier des modèles. L'objectif est de créer un modèle d'exigences de la future application. Un tel modèle doit représenter l'application dans son environnement afin de définir quels sont les services offerts par l'application et quelles sont les autres entités avec lesquelles elle interagit. La création d'un modèle d'exigences est d'une importance capitale. Cela permet d'exprimer clairement les liens de traçabilité avec les modèles qui seront construits dans les autres phases du cycle de développement de l'application, comme les modèles d'analyse et de conception. [Blanc,2005]

B.Modèle d'analyse et de conception abstraite PIM :

Une fois le modèle d'exigences réalisé, le travail d'analyse et de conception peut commencer.Dans l'approche MDA, cette phase utilise elle aussi un modèle. L'analyse et la conception sont les étapes où la modélisation est la plus présente, d'abord avec les méthodes Merise et Coad/Yourdon puis avec les méthodes objet Schlear et Mellor, OMT, OOSE et Booch. Ces méthodes proposent toutes leurs propres modèles. Aujourd'hui, le langage UML s'est imposé comme la référence pour réaliser tous les modèles d'analyse et de conception. [Blanc,2005]

C.Modèle de code PSM :

Une fois les modèles d'analyse et de conception réalisés, le travail de génération de

code peut commencer. Cette phase, la plus délicate du MDA, MDA considère que le code d'une application peut être facilement obtenu à partir de modèles de code. La différence principale entre un modèle de code et un modèle d'analyse ou de conception réside dans le fait que le modèle de code est lié à une plate-forme d'exécution. Ces modèles de code sont appelés des PSM. Les modèles de code servent essentiellement à faciliter la génération de code à partir d'un modèle d'analyse et de conception. Ils contiennent toutes les informations nécessaires à l'exploitation d'une plate-forme d'exécution. [Blanc, 2005]
Dans la figure 3.3, nous présentons le processus de transformation de modèles de l'approche MDA.

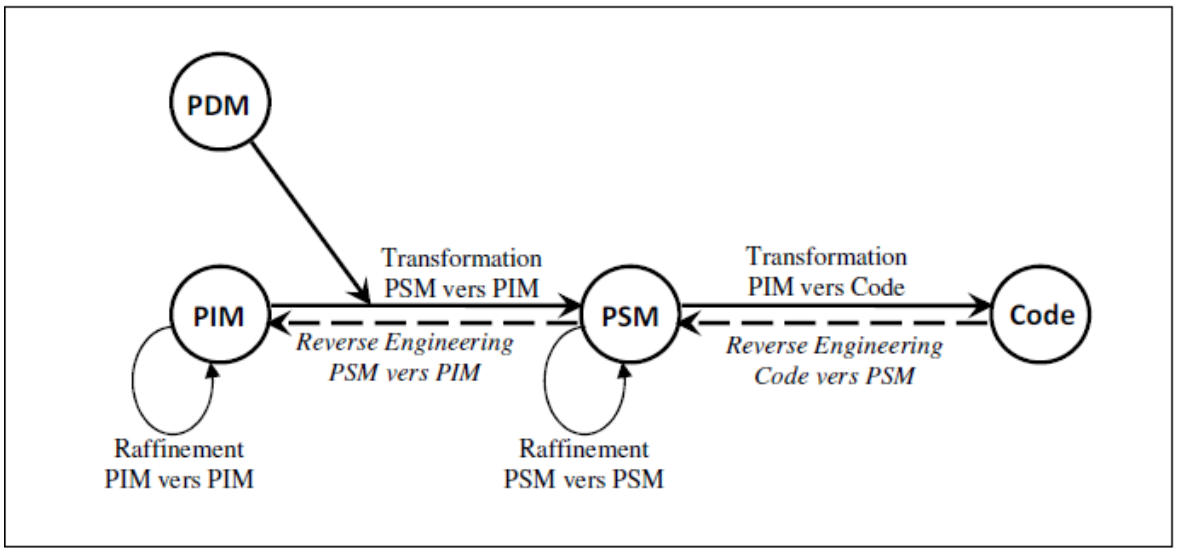


FIGURE 3.3 – Les Modéles et les transformations dans l'approche MDA

3.1.3 les quatre niveaux du Modèle d'architecture MDA

Le modèle d'architecture MDA à quatre niveaux d'abstraction. Montré dans la figure 3.4 [Bahri, 2011] :
Le niveau M0 : Niveau des instances des modèles. Il définit des informations pour la modélisation des objets du monde réel.
Le niveau M1 :Ce niveau représente toutes les instances d'un méta-modèle. Les modèles du niveau M1 doivent être exprimés dans un langage défini au niveau M2. UML est un exemple de modèles du niveau M1.
Le niveau M2 :Ce niveau représente toutes les instances d'un méta-méta-modèle.Il est composé de langages de spécifications de modèles d'information. Le métamodèle UML qui est décrit dans le standard UML et qui définit la structure interne des modèles UML, appartient au niveau M2.
Le niveau M3 : Ce niveau définit un langage unique pour la spécification des métamodèles. Le MOF élément réflexif du niveau M3, définit la structure de tous les méta-modèles du niveau M2.

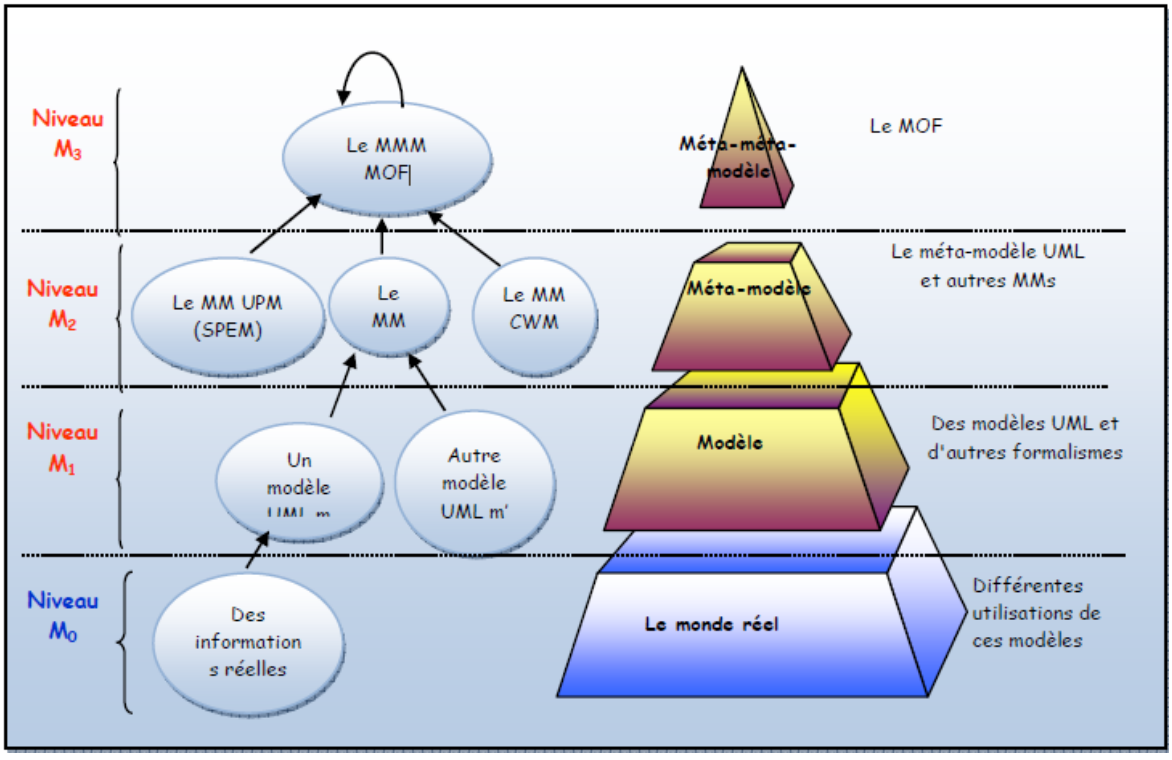


FIGURE 3.4 – Les quatre niveaux d'abstraction pour MDA

3.1.4 Classification des approches de transformation de modèles

Il existe plusieurs axes pour la classification des approches de transformation de modèles. Selon Czarnecki [Czarnecki et Helsen, 2003]qui décompose la transformation de modèle en deux catégories :
les transformations de type modèle vers code qui sont aujourd'hui relativement matures et les transformations de type modèle vers modèle.
1.Une transformation de type modèle vers code Il existe deux approches de transformations de type modèle vers code :
les approches basées sur le principe du visiteur (Visitor-based approach) et les approches basées sur le principe des patrons (Template-based approach) : [Hettab, 2009]
A. Les approches basées sur le principe du visiteur consistent à réduire la différence de sémantique entre le modèle et le langage de programmation cible en se basant sur mécanismes visiteurs dont le quel en ajoutant des éléments dans le modèle et le code est obtenu en parcourant le modèle enrichi pour créer un flux de texte.
B. Les approches basées sur le principe des patrons sont les plus utilisés actuellement et la majorité des outils MDA couramment disponibles supporte ce principe de génération de code à partir de modèle. Parmi les outils basés sur ce principe, on peut citer : OptimalJ, XDE (qui fournissent la transformation modèle vers modèle aussi), JET, ArcStyler et AndroMDA(un générateur de code qui se repose notamment sur la technologie ouverte Velocity pour l'écriture des patrons). Le principe de ces approche repose sur l'utilisation des morceaux de méta-code obtenu a partir du code cible et les utilisés pour accéder aux informations du modèle source.

2.Une transformation de type modèle vers modèle Depuis l'apparition de MDA, Les transformations de type modèle vers modèle necessent évolué jusqu'a nos jours. Il existe un grand espace d'abstraction entre un PIM et un PSM, alors l'utilisation des modèles intermédiaires au lieu d'aller directement d'un PIM vers un PSM est une chose recommandée, ces modèles utilisés pour des raisons d'optimisation ou de déboguage. Ces transformations sont utiles pour le calcul des différentes vues du système, leur synchronisation et pour la vérification et la validation. [Hettab, 2009]
- Structure d'une transformation : [Hettab, 2009]
Une transformation de modèle est principalement caractérisée par la combinaison des éléments suivants : des règles de transformation, une relation entre la source et la cible, un ordonnancement des règles, une organisation des règles, une traçabilité et une direction.
A.Spécification : les pré-conditions et les post-conditions exprimées en OCL (Object Constraint Language) ou un autre langage ce spécification est utilise dans certaines approches.
B. Règles de transformation : une règle de transformation qui possède une logique de forme déclarative ou impérative pour exprimer des contraintes, elle se compose en deux parties : la partie gauche qui s'appelle LHS (Left HandSide) et qui accède au modèle source, et une partie droite RHS (Right Hand Side) qui accède au modèle cible.
C. Organisation des règles : il existe plusieurs façons pour organiser les règles de transformation ; on peut les organiser de façon modulaire, d'un ordonnancement explicite ou d'une structure dépendante du modèle source ou du modèle cible.
D. Ordonnancement des règles : il existe deux types d'ordonnancements des règles l'ordonnancement implicite et l'ordonnancement explicite. Dans le premier type l'ordre des règles est défini par l'outil de transformation elle-même, par contre dans

le deuxième type (ordonnancement explicite) il existe des mécanismes permettent de spécifier l'ordre d'exécution des règles.
E. Relation entre les modèles source et cible : pour certains types de transformations, la création d'un nouveau modèle cible est nécessaire, par contre dans d'autres type, la source et la cible forme le même modèle, ce qui revient en fait à une modification de modèle.
F. Direction : dans le point de vue de la direction de transformation, deux types de direction sont existe : les transformations unidirectionnelles et les transformations bidirectionnelles. Dans le premier cas, l'obtention du modèle cible est basé uniquement sur modèle source, par contre dans le second cas, une synchronisation entre les modèles source et cible est possible.
G. Traçabilité : la traçabilité est le processus d'archivage des corrélations existes entre les éléments des modèles source et cible. Certaines approches supportent la traçabilité en fournissant des mécanismes dédiées pour elle. Dans les autres cas, le développeur doit implémenter la traçabilité de la même manière qu'il crée n'importe quel autre lien dans un modèle.
H. Les approches pour la définition des transformations : Plusieurs approches sont utilisées pour définir la transformation d'un modèle à l'autre. Il existe cinq approches dans la littérature [Aouag.2014] :
-les approches par manipulation directe .
-les approches relationnelles .
-les approches basées sur la transformation de graphes .
-les approches basées sur la structure .
-les approches hybrides.
Dans notre mémoire, on s'intéresse particulièrement aux approches basées sur la transformation de graphes, du fait que notre objectif est de transformer les diagrammes UML 2.0 Orienté Aspect (OA) qui sont des graphes vers le langage Maude qui eux aussi sont sous forme de code.

3.1.5 Les outils d'MDA

Pour obtenir une telle efficacité, plusieurs outils conceptuels sont mis à disposition. La technologie MDA (Model Driven Architecture) est supportée par l'OMG (Object Management Group), qui propose également UML (Unified Modeling Language) et Corba (Object Request Broker). Ces outils sont :
A. UML largement utilisé par ailleurs qui permet une mise en œuvre aisée de MDA en offrant un support connu.
B. XMI (XML Metadata Interchange) qui propose un formalisme de structuration des documents XML de telle sorte qu'ils permettent de représenter des méta-données d'application de manière compatible.
C. MOF (Meta Object Facility) spécification qui permet le stockage, l'accès,la manipulation, la modification de méta-données. Le MOF permet une unification de l'expression des méta-modèles, qu'ils soient ensuite utilisés comme profils UML ou non .
D.CWM base de données pour méta-données. [Hettab,2009]

3.2 Transformation des modèle

La notion de transformation de modèles constitue l'élément central de la démarche IDM. En effet, cette notion porte sur l'automatisation de l'opération de transformation pendant le cycle de développement qui peut avoir des sémantiques différentes en fonction des utilisation. La transformation de modèles est une opération qui consiste à générer un ou plusieurs modèles cibles conformément à leur méta-modèle à partir d'un ou de plusieurs modèles sources conformément à leur méta-modèle.
Elle est qualifiée d'endogène si les modèles sources et cibles sont conformes au même méta-modèle (source et cible sont dans le même espace technologique),sinon elle est dite exogène et elle se fait entre deux méta-modèles différents (source et cible sont dans deux espaces technologiques différents).[kerkouche,2011]
Dans la littérature, on peut distinguer trois types de transformations :[kerkouche,2011]
Les transformations verticales : La source et la cible d'une transformation verticale sont définies à différents niveaux d'abstraction. Une transformation qui baisse le niveau d'abstraction est appelée un raffinement. Une transformation qui élève le niveau d'abstraction est appelée une abstraction.
Les transformations horizontales : Une transformation horizontale modifie la représentation source tout en conservant le même niveau d'abstraction. La modification peut être l'ajout, la modification, la suppression ou la restructuration d'informations.
Les transformations obliques : Une transformation oblique combine une transformation horizontale et une verticale. Ce type de transformation est notamment utilisé par les compilateurs, qui effectuent des optimisations du code source avant de générer le code exécutable. Dans la figure 3.5 [Czarnecki et Helsen, 2006], nous présentons les concepts de base de la transformation de modèles.

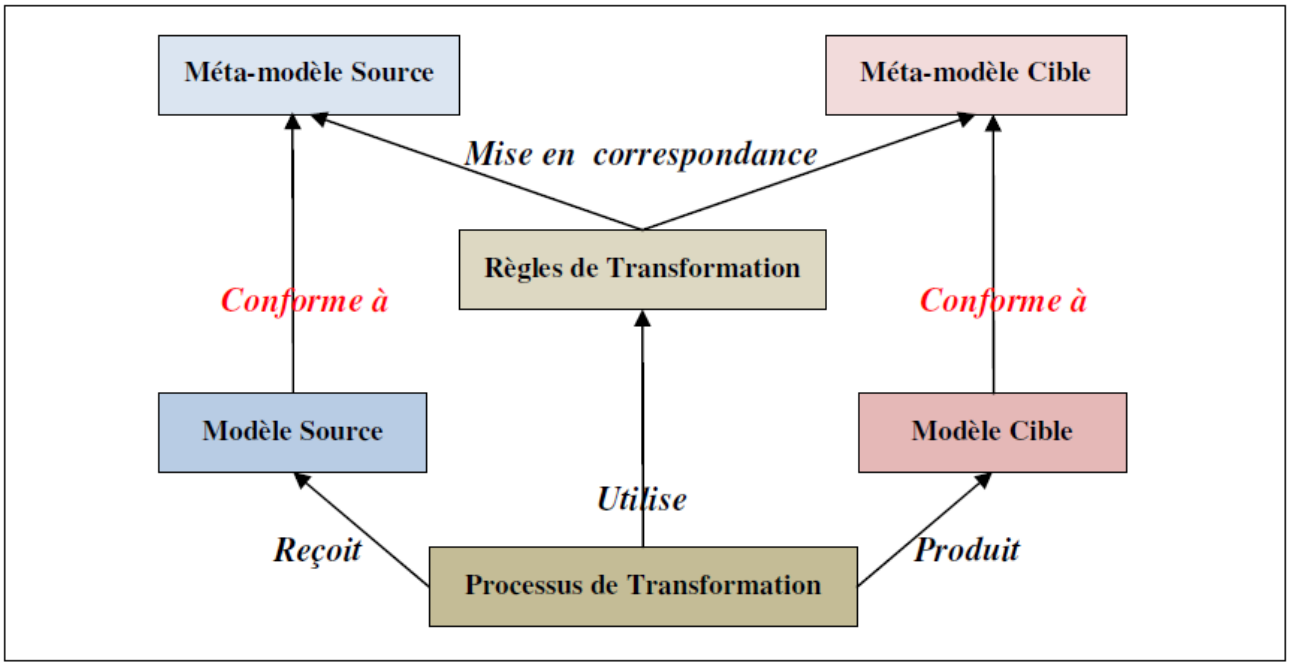


FIGURE 3.5 – Concepts de base de la transformation de modèles

3.3 La transformation de graphes

La transformation de graphes est apparue à cause du manque d'expressivité des approches classiques de réécriture comme les grammaires de Chomsky et la réécriture de termes pour gérer les structures non-linéaires.
Les graphes et les diagrammes offrent un support agréable et intuitif pour la modélisation des systèmes complexes. un graphe est un schéma constitué par un ensemble de points et par un ensemble de flèches reliant chacune deux de ceux-ci. Une transformation de graphe consiste en l'application d'une règle à un graphe, et la partie du graphe qui corresponde cette règle sera remplacée par un autre graphe. Ce processus est réitéré jusqu'à ce qu'aucune règle ne puisse être appliquée. Ces dernières sont une généralisation des grammaires de Chomsky pour les graphes. Elles sont composées de règles. Une règle est constituée de deux parties, le Left Hand Side (LHS) et le Right Hand Side (RHS).[kerkouche,2011]

3.3.1 Grammaires de graphes

Une grammaire de graphes est une structure GG , L'idée est de combiner deux concepts importants : les graphes et les règles. généralement définie par un triplet :$GG = (P; S; T)$ où :
P : ensemble de règles.
S : un graphe initial.
T : ensemble de symboles terminaux T.
Une grammaire de graphes distingue les graphes non terminaux, qui sont les résultats intermédiaires sur lesquels les règles sont appliquées, des graphes terminaux dont on ne peut plus appliquer de règles. Ces derniers sont dans le langage engendré par la grammaire de graphe. Pour vérifier si un graphe G est dans les langages engendrés par une grammaire de graphe, il doit être analysé. Le processus d'analyse va déterminer une séquence de règles dérivant G. [Kerkouche,2011]

3.3.2 Langage engendré

Soit G0 un graphe initial,Gn est le graphe final et une séquence de transformations successives de graphes :$G0 -> G1 -> ... -> Gn$ est une dérivation successive à partir de G0 vers Gn en appliquant les règles de R qui sont étiquetées par les symboles de T, est dit langage engendré par R, G0 et T et on écrit $L(R, G0, T)$ [Aouag,2014].

Le principe de règles

Une règle de transformation de graphe est définie par :[Kerkouche,2011]
$r = (L, R, K, glue, emb, cond)$. Elle consiste en :
• Deux graphes : L graphe de coté gauche et R graphe de coté droit.
• Un sous graphe K de L.
• Une occurrence glue de K dans R qui relie le sous graphe avec le graphe de coté droit.
• Une relation d'enfoncement emb qui relie les sommets du graphe de coté gauche et ceux du graphe du coté droit.
• Un ensemble cond qui spécifie les conditions d'application de la règle.

L'application des règles

L'application d'une règle $r = (L, R, K, Fcond)$ à un graphe G produit un graphe résultant H. Le graphe H peut être obtenu depuis le graphe d'origine G en passant par les cinq étapes suivantes [Aouag,2014] :
1. le choix d'une occurrence du graphe du coté gauche L dans G,selon la règle à appliquer.
2. la vérication des conditions d'application selon la condition cond.
3. la suppression de l'occurrence de L (jusqu'à K) de G ainsi que les arcs pendillés, c'est-à-dire tout les arcs qui ont perdu leurs sources et/ou leurs destinations. Ce qui fournit le graphe de contexte D de L qui a laissé une occurrence de K.
4. Coller le graphe de contexte D et le graphe du coté droit R suivant l'occurrence de K dans D et dans R. C'est la construction de l'union de disjonction de D et R et, pour chaque point dans K, identier le point correspondant dans D avec le point correspondant dans R.
5. l'enfoncement du graphe de coté droit dans le graphe de contexte de L suivant la relation d'enfoncement emb.
L'application de r à un graphe G pour produire un graphe H est appelée une dérivation directe depuis G vers H à travers r, elle est dénotée par $G- > H$. En donnant les notions de règle et de dérivation directe comme étant les concepts élémentaires de la transformation de graphe, ont peut définir les systèmes de transformation de graphe et la notion de langages engendrés. [Kerkouche, 2011] .

3.3.3 Outils de transformation de graphes

Il existe plusieurs outils pour transformer les graphes. Citons quelques exemples d'outils de transformation des graphes : AGG, TGG, ATOMPM, Eclipse Modeling, Progres, ($AToM^3$) etc.Notre choix est porté sur ($AToM^3$) à cause des avantages qu'il présente. Nous pouvons citer parmi ces avantages [Aouag,2014] :
1. sa simplicité.
2. sa disponibilité.
3. Il est multi paradigmes (la possibilité de méta-modéliser).
4. Ils'adapte bien avec les types de transformations.

3.4 Présentation de l'outil ($AToM^3$)

Aujourd'hui, plusieurs outils et technologies permettent aux modélisateurs de développer des langages de modélisation spécifiques au domaine (DSMLs) et de manipuler des modèles, tels que ($AToM^3$).
nous présentons ($AToM^3$) "A Tool for Multi-formalism and Meta-Modeling " est un outil visuel de modélisation et de méta-modélisation multi-formalismes. Il est développé au laboratoire MSDL (Modeling, Simulation and Design Lab) à l'école d'informatique de l'université de McGill.($AToM^3$) repose sur la réécriture de graphes qui utilise les règles de grammaire de graphes pour la définition des transformations entre les formalismes, ainsi que la génération du code et la spécification des simulateurs. L'utilisateur doit défini les règles (elles sont modélisées comme des entités composées de : LHS et RHS), les priorités et les conditions qui doivent être vérifiées pour que la règle soit

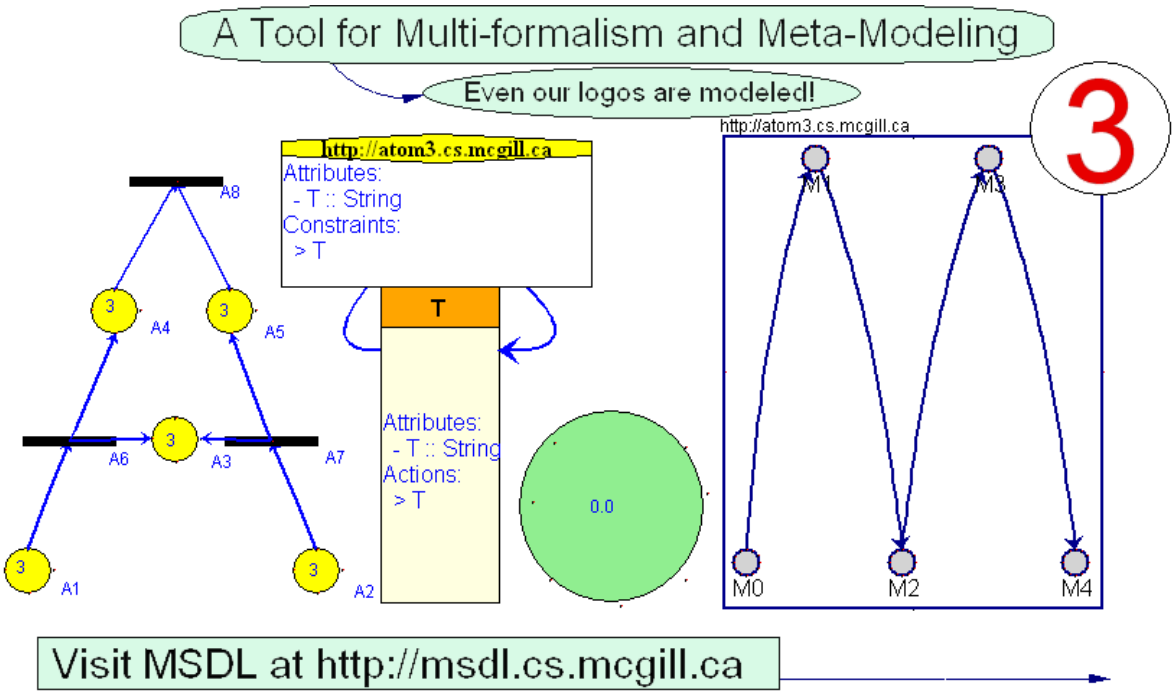


FIGURE 3.6 – Présentation de l'outil ($ATOM^3$).

applicable (pré-condition, post-condition, sur quel évènement).[Hamrouche, 2016]
La transformation de modèles est implémentée en Python et s'exécute, sans aucun changement, sur toutes les plateformes sur lesquels il existe un interpréteur de Python (Linux, Windows et MacOS).[Aouag, 2014]
La figure 3.6 présente l'outil ($ATOM^3$).

Conclusion

Dans ce chapitre, nous avons présenté les concepts de base de la transformation de modèles qui est considéré comme la clé de la démarche MDA. Dans ce contexte, nous avons présenté l'architecture MDA avec ces différents modèles, la transformation de modèles, les différentes approches existantes en se basant sur une classification proposée dans la littérature. Par la suite nous avons donné un aperçu sur la transformation de graphes qui représente une approche de transformation de modèles, la notion de graphe, la grammaire de graphes principes et les applications , quelques outils de transformation de graphes et nous avons également présenté l'outil ($AToM^3$), utilisé dans l'implémentation de notre travail. Les concepts présentés dans ce chapitre constituent un package nécessaire pour la compréhension de nos contributions dans le cadre de ce mémoire qui seront présentées dans le chapitre suivant (Chapitre 4).

Chapitre 4

L'approche De Transformation Proposée

Introduction

Dans ce chapitre, nous présentons notre contribution qui consiste à transformer des diagrammes UML2.0 Orientés Aspect vers le langage Maude en se basant sur l'approche de transformation de graphes. Le chapitre contient trois contributions :
1. La transformation des diagrammes de classe orientés aspect vers le langage Maude.
2. La transformation des diagrammes d'état-transition orientés aspect vers le langage Maude .
3. La transformation des diagrammes de communication orientés aspect vers le langage Maude .
Nous commençons chaque transformation de notre approche de transformation par la déffenition des méta-modèles associés aux modèles source et cible.Ensuite, nous proposons notre grammaire de graphes qui permet la transformation des modèles d'entrée vers des codes. Cette transformation est réalisée à l'aide de l'outil de modélisation ($AToM^3$) et les contraintes sont exprimées en python.

4.1 Transformation des diagrammes de classe orientés aspect vers le langage Maude

L'idée de base de la transformation consiste à remplacer chaque entité,classe et association d'un Diagramme de Classe Orienté Aspect par une code génère avec le langage Maude. nous avons proposé méta-modèle du Diagramme de Classe Orienté Aspect (DCOA) et une grammaire de graphe. Cette transformation est réalisée à l'aide de l'outil ($AToM^3$).La structure de cette transformation est résumée dans la figure 4.1.

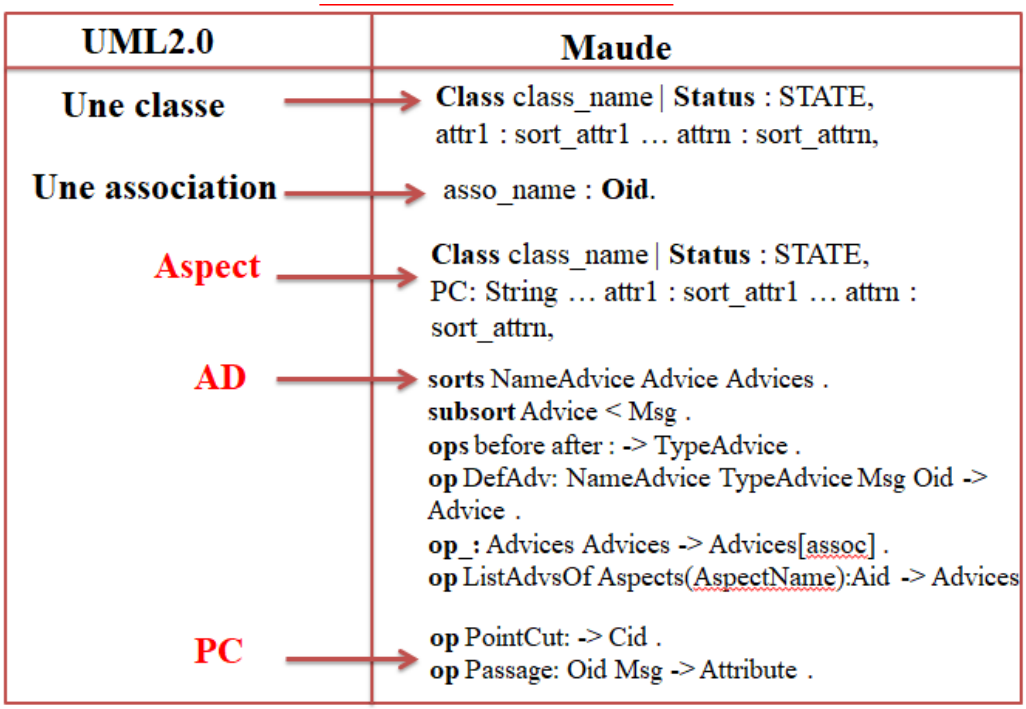


FIGURE 4.1 – Structure de transformation d'un diagramme de classe orienté aspect vers le langage Maude.

4.1.1 Méta-modélisation des diagrammes de Classe orientés Aspect

Le méta-modèle du diagramme de classe orienté aspect composé de huit classes (diagramclassaspect, Class-simple, Composition, Agrigation, Heritage (inheritance),Association, Association-naire, Aspect) et sept Associations(Association-1,Association-2, Association-3, Association-4 et Association-5, Association-6,Composer).
Dans la figure 4.2 , nous présentons méta Modéle pour les diagrammes de classe orientés aspect.

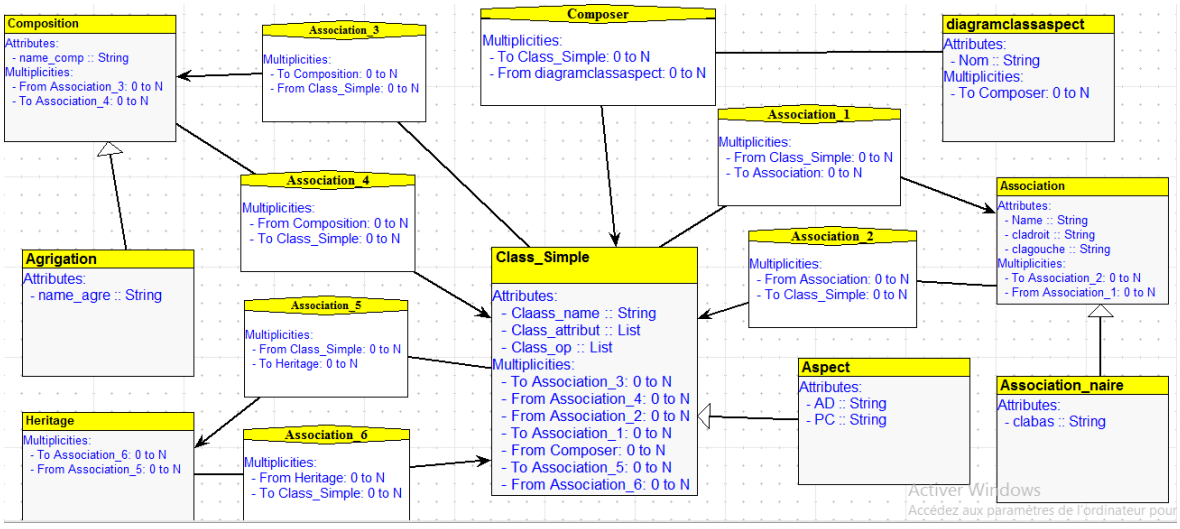


FIGURE 4.2 – Méta modèle pour les diagrammes de classe orientés aspect.

1- Les Classes :

- Diagramclassaspect : cette classe représente le diagramme de classe orienté aspect. Elle possède un attribut $< Nom >$ de type String. Graphiquement, elle est représentée par un grand rectangle rouge.
- Class-simple : cette classe représente les classe-simple. Elle possède trois attributs $< Claas - name >$ de type string,$< Class - attribut >$et $< Class - op >$ de type list. Graphiquement, elle est représentée par une classe noire.
- Aspect : est une classe qui hérite la classe $< Class - Simple >$, elle hérite tous les attributs de la $< Class - Simple >$ et plus deux attributs $< AD >$ et $< PC >$ de type string.Graphiquement, elle est représentée par une classe rouge.
- Association : cette classe représente les associations entre deux classes. Graphiquement elle est représentée par un lien, elle possède trois attributs $< Name >$ $< cladroit >$ et $< clagouche >$ de type string.
- Association-naire : est une classe qui hérite la classe $< Association >$,elle hérite tous les attributs de la classe $< Association >$ et plus l'attribut $< clabase >$ de type string. Cette classe représente les associations entre plusieurs classes. Graphiquement elle est représentée par un lien.
- Composition : cette classe représente la composition entre deux classes. Elle possède l'attribut $< name - comp >$ de type string. Graphiquement elle est représentée par une losange plaine.
- Agrigation : est une classe qui hérite la classe composition, cette classe représente l'agrégation entre deux classes. Elle possède l'attribut $< name - agre >$ de type string. Graphiquement elle est représentée par un losange vide.
- Heritage : cette classe représente l'héritage entre deux classes. Graphiquement elle est représentée par une flache.

2- Les Associations :

- Associations-1 : relie la classe $< Class - Simple >$ et la classe $< Association >$, les cardinalités de cette association sont :
-To association : 0 to N.
-From class-Simple :0 to N .
- Associations-2 : relie la classe $< Association >$ et la classe $< Class - Simple >$, les cardinalités de cette association sont :
- To Class-Simple : 0 to N.
- From association : 0 to N.
- Associations-3 : relie classe $< Class - Simple >$ et la classe $< Composition >$, les cardinalités de cette association sont :
- To composition : 0 to N .
- From Class-Simple :0 to N .
- Associations-4 : relie la classe $< Composition >$ et la classe $< Class - Simple >$, les cardinalités de cette association sont :
- To Class-Simple :0 to N .
- From Composition : 0 to N .
- Associations-5 : relie la classe $< Class - Simple >$ et le classe $< Heritage >$, les cardinalités de cette association sont :
- To Heritage : 0 to N .
- From Class-Simple 0 to N.

- Associations-6 : relie le classe $< Heritage >$ et la classe $< Class - Simple >$, les cardinalités de cette association sont :
- To Class-Simple 0 to N.
- From Heritage : 0 to N .

- Composer : relie le classe $< diagramclassaspect >$ et la classe $< Class - Simple >$, les cardinalités de cette association sont :
-To Class-Simple :0 to N.
-From diagramclassaspect : 0 to N .

La figure 4.3 représente l'outil généré pour la manipulation des diagrammes de classes orientes aspect.

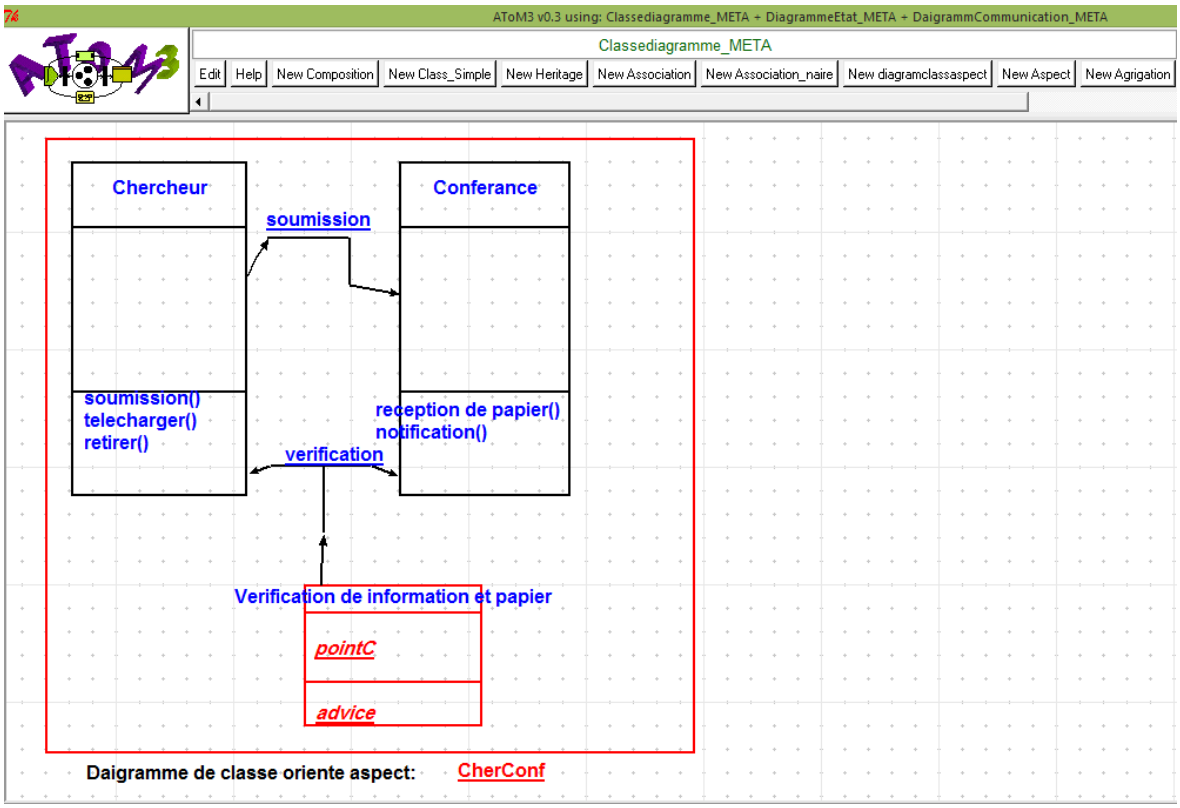


FIGURE 4.3 – l'outil généré pour le diagramme de classe orienté aspect.

4.1.2 Grammaire de graphes pour la transformation des diagrammes de classe orientés aspect vers le langage maude

Nous avons défini une grammaire de graphes composee par des règles qui seront exécutées dans un ordre ascendant selon leurs priorités,Notre technique de transformation est basée sur la transformation de graphes, alors chaque règle est composée de deux parties : une partie gauche (LHS) et une partie droite (RHS).
Définition : Une grammaire de graphes est une structure GG , généralement définie par un triplet : GG = $(P; S; T)$ où :
P : ensemble de règles.
S : un graphe initial.
T : ensemble de symboles terminaux T.[Aouag,2014]
Dans la Figure 4.4, nous présentons une grammaire de graphes contenant 20 règles chacune exprimant un cas particulier.

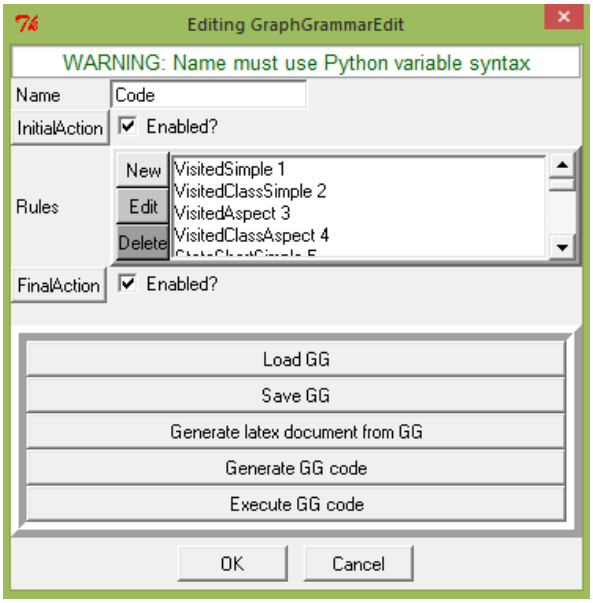


FIGURE 4.4 – Grammaire de graphes pour les diagrammes de classe orientés aspect

Catégorie 01 :

Nous décomposons les 20 règles en 6 catégories comme suit :
cette Catégorie contient 4 Régles (1,2,3,4). Ces règles sont appliquées pour localiser une classe qui n'a pas encore été traitée (Visité = = 0), et créer un fichier pour chacune. La structure de Ces règles est résumée dans les figures[4.5, 4.6 et 4.7] :

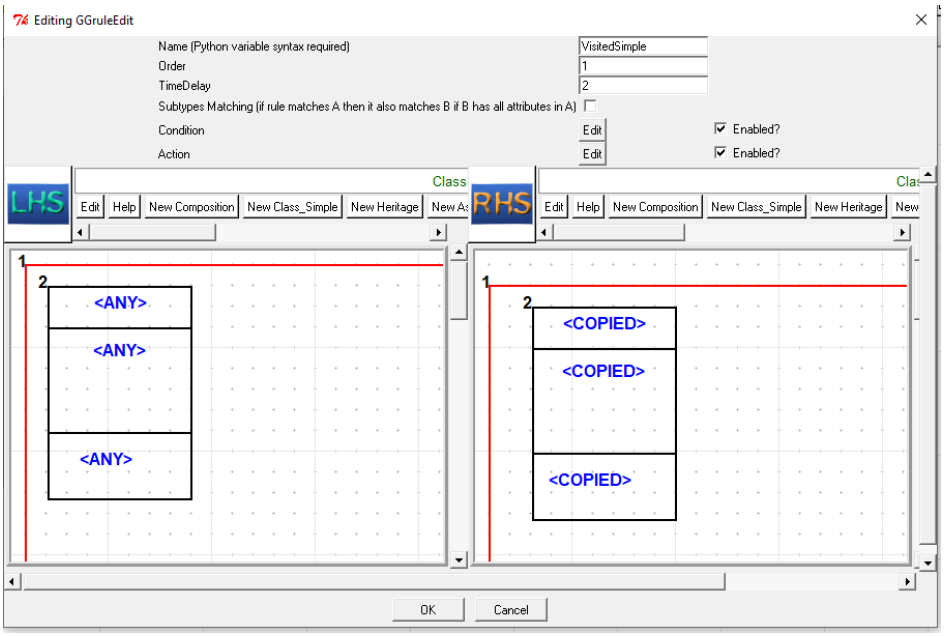


FIGURE 4.5 – La 1 ère règle

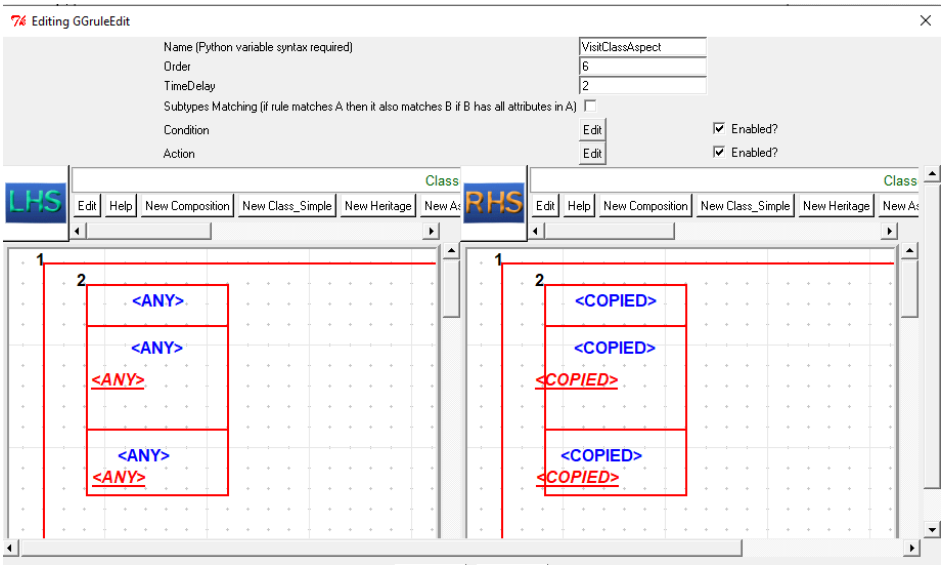


FIGURE 4.6 – La 2 ère règle

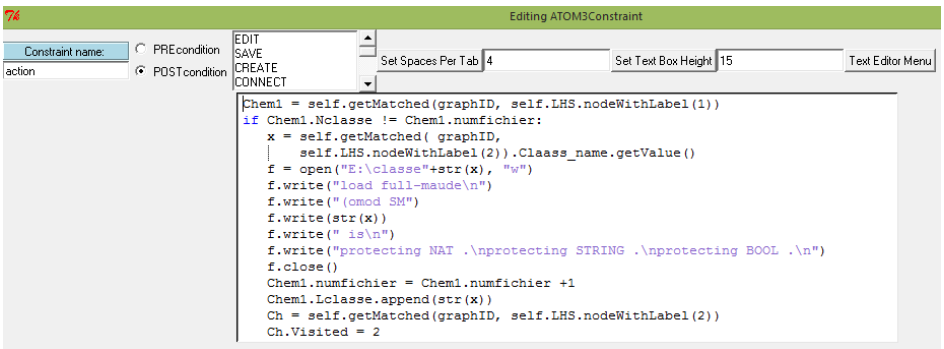


FIGURE 4.7 – Le Code Maude Proposé dans ces règles

Catégorie 02 : cette Catégorie contient 2 Régles (5,6) permettent de vérifier si une classe a sa propre State Machine pour ajouter les formats qui permettent de représenter les états simples, états composites, Aspect Simples et Aspect Composites, en fonction de la condition `(nom_class)` == `(nom_stchart)`.nous avons déclaré une structure algébrique comme :
sorts SIMSTATE COMSTATE STATE .
subsort SIMSTATE < COMSTATE .
subsort COMSTATE < STATE .
op none : − > COMSTATE [ctor] .
`op_||_` : COMSTATE COMSTATE − > COMSTATE [ctor assoc comm id : none].
La structure de Ces règles est résumée dans les figures[4.8, 4.9 et 4.10] :

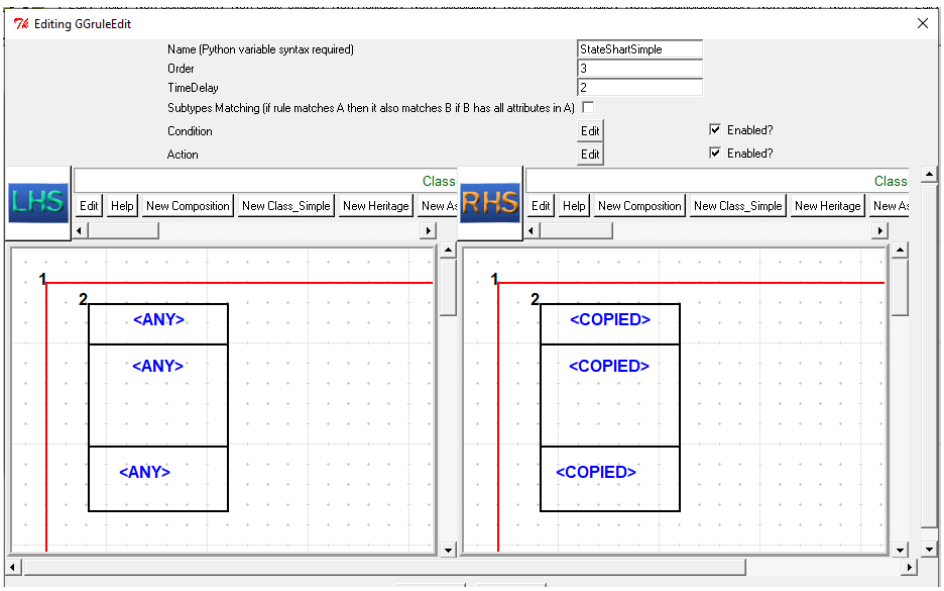


FIGURE 4.8 – La 5 ère règle

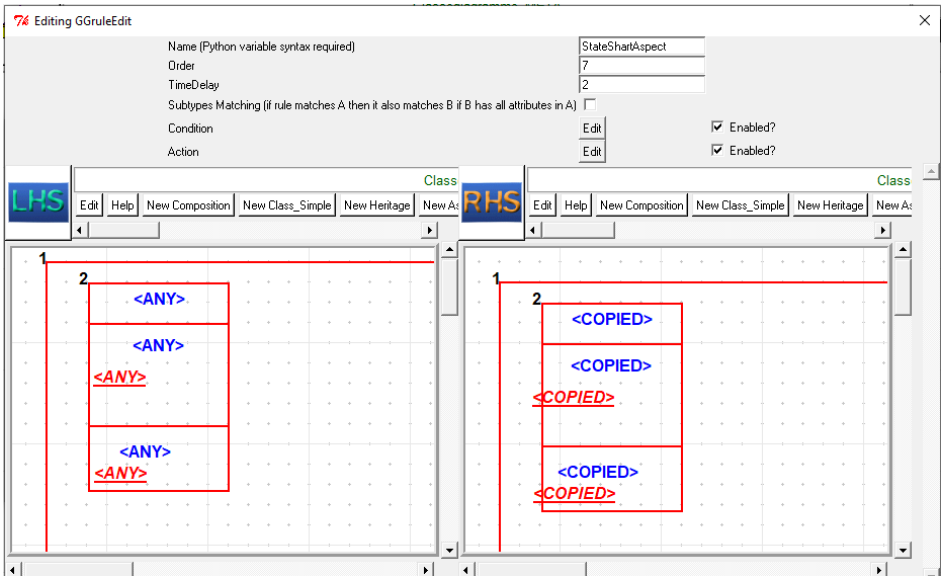


FIGURE 4.9 – La 6 ère règle

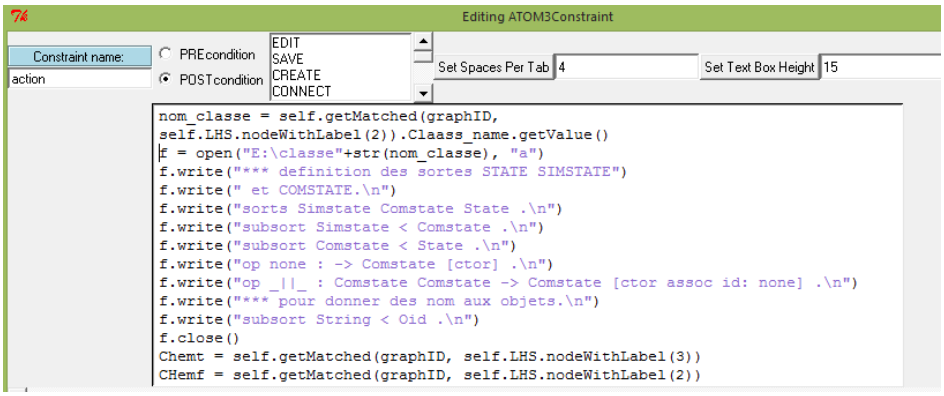


FIGURE 4.10 – Le Code Maude Proposé dans ces règles

Catégorie 03 : cette Catégorie contient 2 Régles (7,8) permettent la récupération des informations associées aux classes (Simple/Aspect), que pour définir une classe, on peut utiliser la syntaxe suivante :
Class `(nom_class)` — Status : STATE, attr1 : `(sort_attr1)`...attrn : `(sort_attrn)`, `(asso_name)` : Oid
La structure de Ces règles est résumée dans les figures[4.11, 4.12 et 4.13] :

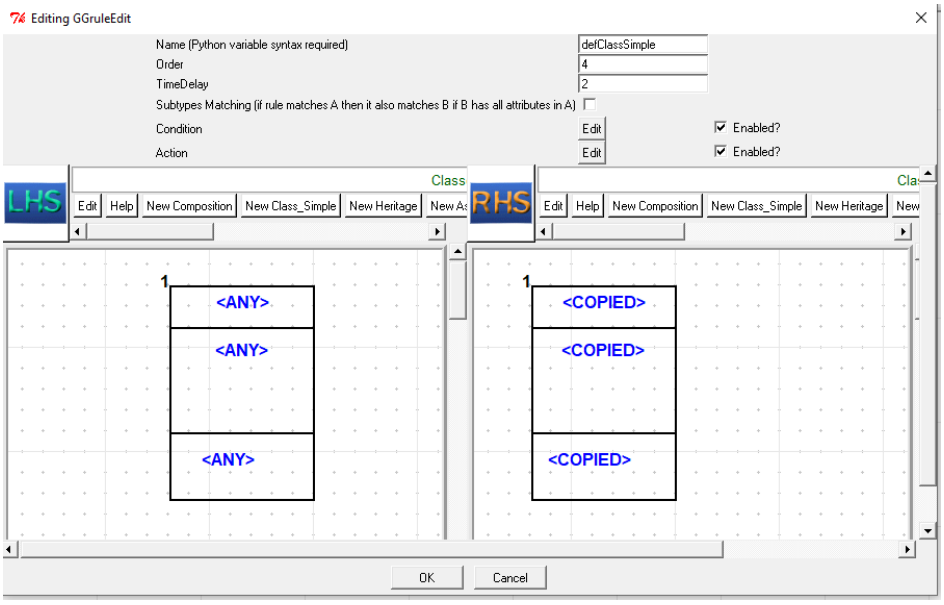


FIGURE 4.11 – La 7 ère règle

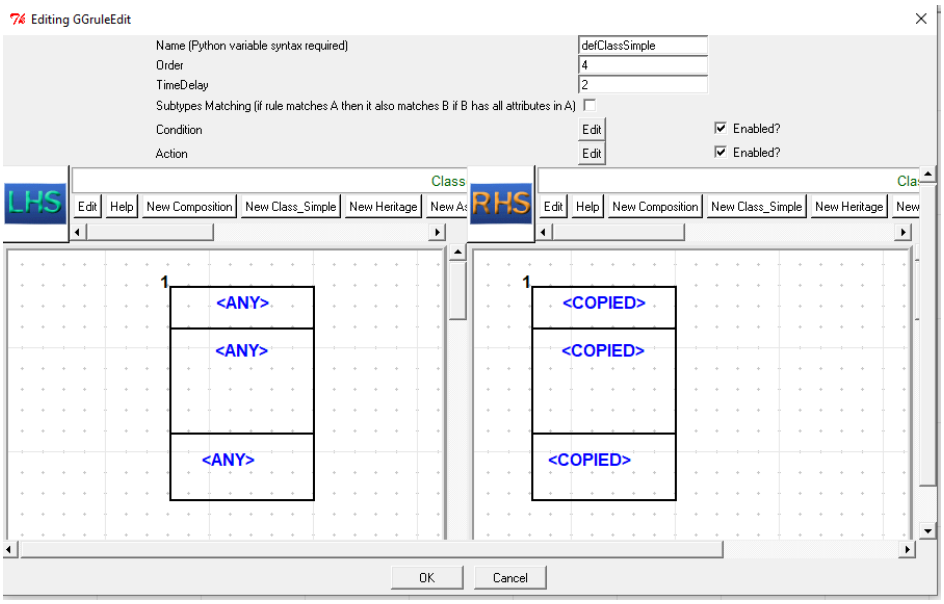


FIGURE 4.12 – La 8 ère règle

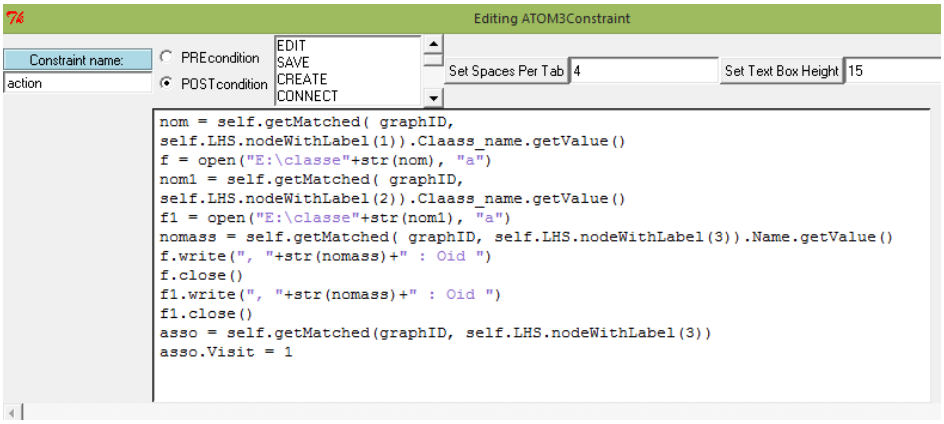


FIGURE 4.13 – Le Code Maude Proposé dans ces règles

Catégorie 04 : cette Catégorie contient 4 Régles (9,10,11,12) permettent la récupération des informations associées aux associations[Simple - Aspect, Aspect - Simple, Simple - Simple , Aspect - Aspect], et marquent l'association comme visitée (Asso.Visit = 1). La structure de Ces règles est résumée dans la figure 4.14 :

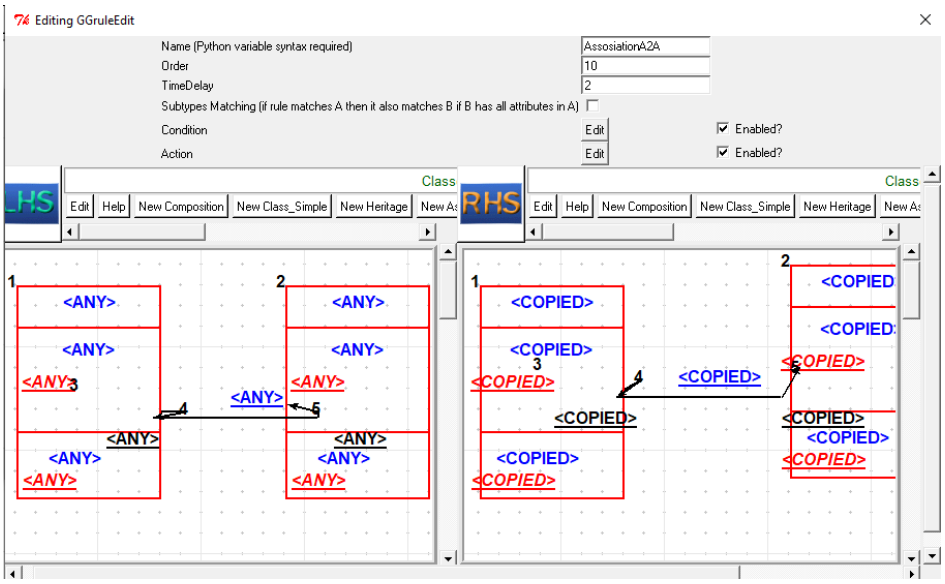


FIGURE 4.14 – La 10 ère règle

Catégorie 05 : cette Catégorie contient 6 Régles (13,14,15,16,17,18) permettent la récupération des informations associées aux associations [Simple - Simple - Simple, Aspect - Aspect - Aspect, Simple - Simple - Aspect, Aspect - Aspect - Simple, Simple - Aspect -Simple, Aspect - Simple - Aspect]. La structure de Ces règles est résumée dans la figure 4.15 :

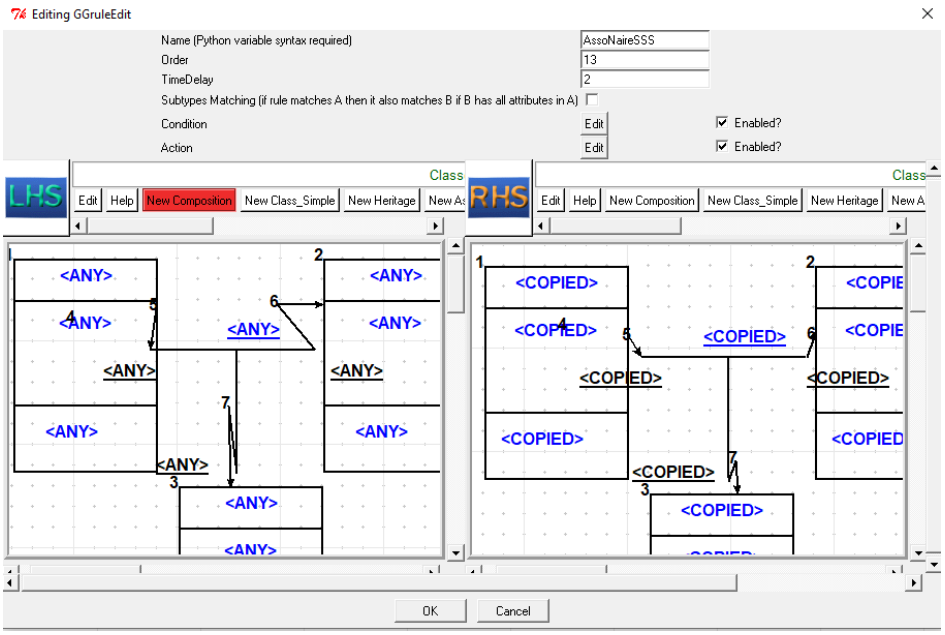


FIGURE 4.15 – La 13 ère règle

Catégorie 06 :cette Catégorie contient 2 Régles (19,20) représentent la fin des Classes (Simple/Aspect). La structure de Ces règles est résumée dans la figure 4.16 :

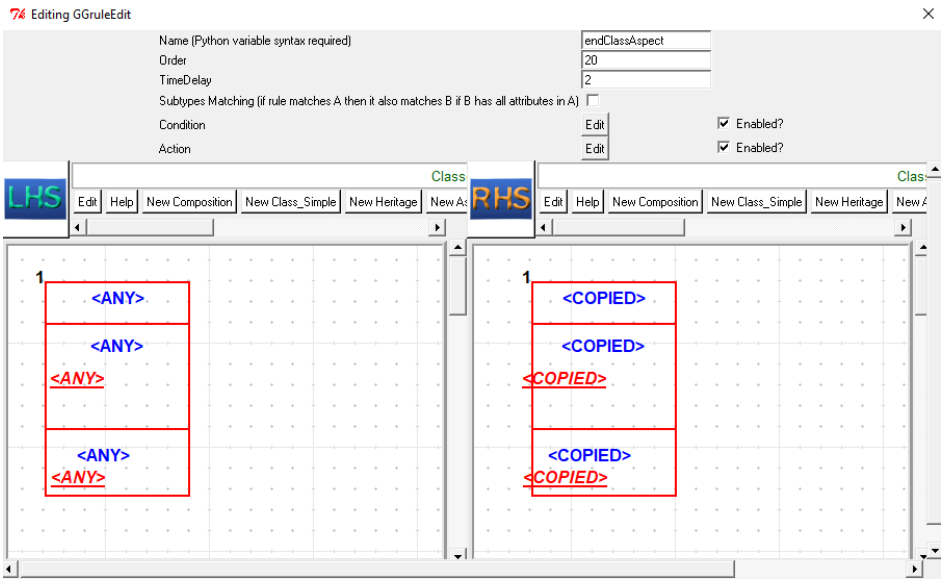


FIGURE 4.16 – La 20 ère règle

4.2 Transformation des diagrammes d'état-Transition orientés aspect vers le langage Maude

Dans cette section nous donnons l'approche de transformation des Diagrammes d'état-transition orientés aspect vers le langage Maude. Cette transformation consiste à remplacer chaque entité et Transition ou association d'un diagramme d'état-Transition orienté aspect par un code génère avec le langage Maude . Nous avons proposé méta-modèle du Diagramme d'état-transition Orienté Aspect, et une grammaire de graphe. Cette transformation est réalisée à l'aide de l'outil ($AToM^3$).
La structure de cette transformation est résumée dans la figure 4.17 .

UML2.0	Maude
Etat simple ou Composit dans le State Chart →	**sorts** SIMSTATE COMSTATE STATE . **subsort** SIMSTATE < COMSTATE . **subsort** COMSTATE < STATE . **op** none: -> COMSTATE [ctor] . **op** _‖_: COMSTATE COMSTATE -> COMSTATE [ctor assoc comm id: none].
Aspect simple ou Composit dans le State Chart →	**sorts** SIMSTATE COMSTATE STATE . **subsort** SIMSTATE < COMSTATE . **subsort** COMSTATE < STATE . **op** none: -> COMSTATE [ctor] . **op** _‖_: COMSTATE COMSTATE -> COMSTATE [ctor assoc comm id: none].

FIGURE 4.17 – Structure de transformation d'un diagramme d'état-transition orienté aspect vers le langage Maude.

4.2.1 Méta-modélisation des diagrammes d'état-transition orientés Aspect

Le méta-modèle du diagramme d'état-transition orienté aspect composé de sept classes (etat-initial, etat-final, etat-imple, etat-composit, Aspect-simple ,Aspect-composit, diag-ET-OA) et dix Associations (initial2simple , initial2composit, composit2simple , simple2composit, simple2simple , simple2finale , composit2final , contient-Esimple, contient-final, comoancer).
Dans la Figure 4.18, nous présentons méta Modéle pour les diagrammes D'état-transition orientés aspect

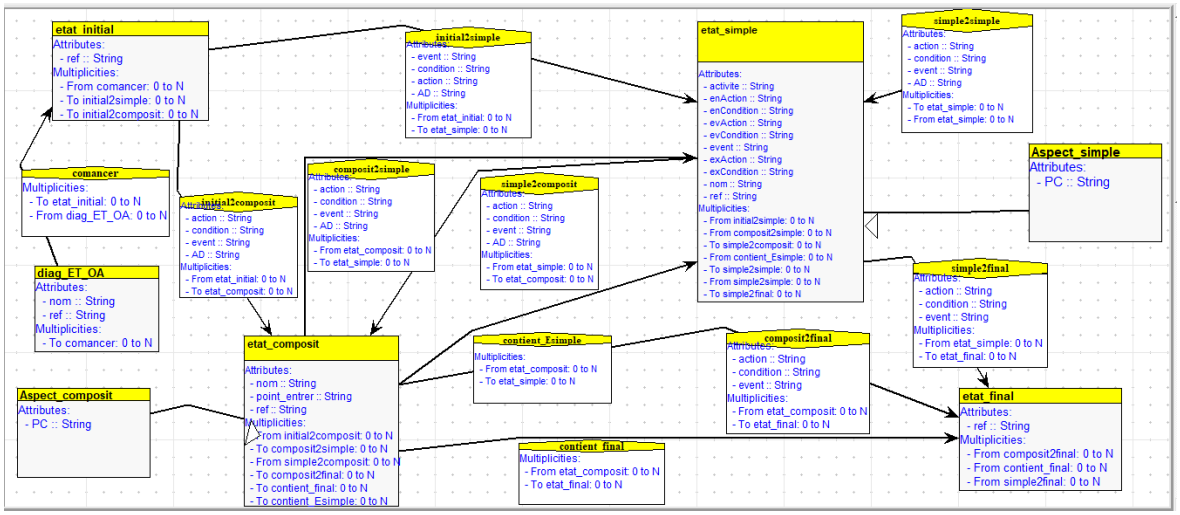


FIGURE 4.18 – Méta modèle pour les diagrammes d'état-transition orientés aspect.

Les Classes :

- diag-ET-OA : Cette classe a un attribut qui est le Nom, elle représente les états dans le diagramme. Graphiquement, elle est représentée par un grand rectangle rouge.
- etat-simple : Cette classe représenté les états simples (les étapes de la vie du système).
- etat-initial : Cette classe représenté l'état initial(l' initialisation du système).
- etat-composit : Cette classe représenté les états composites (état regroupant un ensemble d'états).
- etat-final : Cette classe représente l'état final du diagramme d'états- transitions (fin de vie du système).
- Aspect-simple : est une etat qui hérite tous les attributs de $< etat - simple >$ et plus deux attributs $< AD >$ et $< PC >$ de type string. Graphiquement, elle est représentée par une petite rectangle rouge.
- Aspect-composit : est une etat qui hérite tous les attributs de $< etat - composit >$ et plus deux attributs $< AD >$ et $< PC >$ de type string. Graphiquement, elle est représentée par une ectangle rouge.

Les Associations :

-Ces associations (initial2simple , initial2composit, composit2simple , simple2composit, simple2simple , simple2finale , composit2final) du méta-modèle possède des attributs

de type String(event : c'est l'événement c'est un fait instantané venant de l'extérieur du système ,condition : des condition vérifiées, action : c'est la réaction du système a un événement), Les cardinalités pour chaque association sont :
1. To target : 0 to N.
2. From source : 0 to N.
Et plus trois association : **- *contient-Esimple :*** relie $< etat - composit >$ et $< etat - simple >$, les cardinalités de cette association sont :
- To etat-simple 0 to N.
- From etat-comosit : 0 to N .
- contient-final : relie $< etat - composit >$ et ¡etat-final¿, les cardinalités de cette association sont :
- To etat-final 0 to N.
- From etat-composit : 0 to N .
- comoncer : relie le classe $< diag - ET - OA >$ et $< etat - initial >$, les cardinalités de cette association sont :
-To etat-initial :0 to N.
-From diag-ET-OA : 0 to N .
La figure 4.19 représente l'outil généré pour la manipulation des diagrammes d'état-transition orientes aspect.

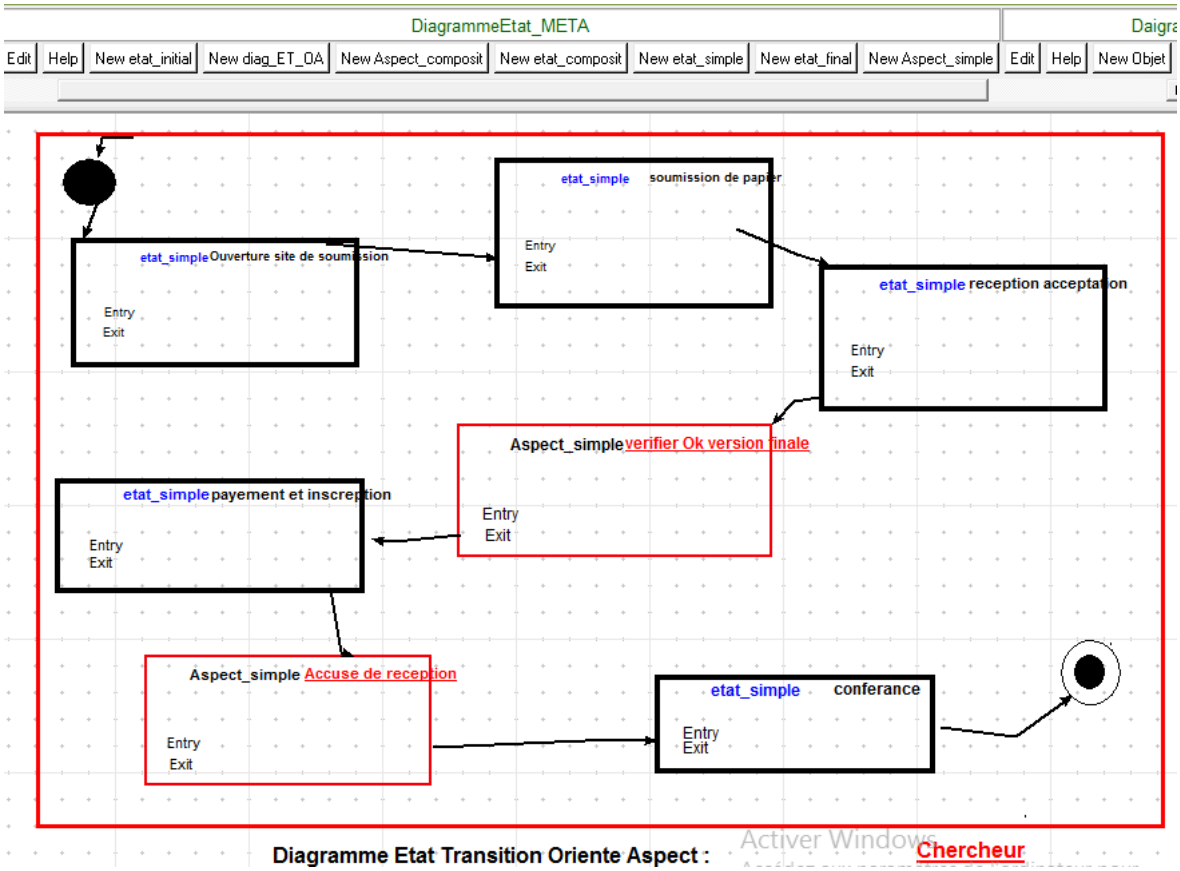


FIGURE 4.19 – l'outil généré pour les diagrammes d'état-transition orientés aspect.

4.2.2 Grammaire de graphes pour la transformation des diagrammes d'état-transition orientés aspect vers le langage maude

Nous proposons une grammaire de graphes contenant 35 règles chacune exprimant un cas particulier. Ces règles seront appliquées dans l'ordre croissant(chaque règle a une priorité). La Figure 4.20 représente notre grammaire.

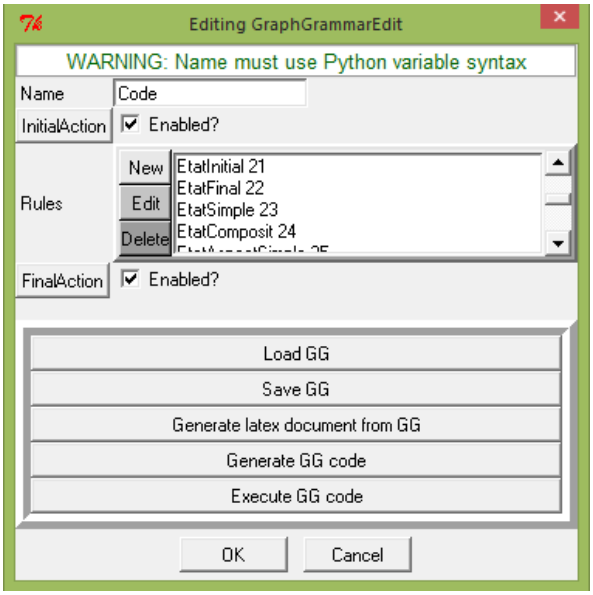


FIGURE 4.20 – Grammaire de graphes pour les diagrammes d'état-transition orientés aspect.

Nous décomposons les 35 règles en 4 catégories comme suit :
Catégorie 01 : cette Catégorie contient 6 Régles (21,22,23,24,25,26)Ces règles permettent d'ajouter le code Maude appropriés des états initial, états final, états simple, états Composit, Aspect Simple, Aspect Composit. La structure de Ces règles est résumée dans les figures[4.21, 4.22, 4.23 et 4.24] :

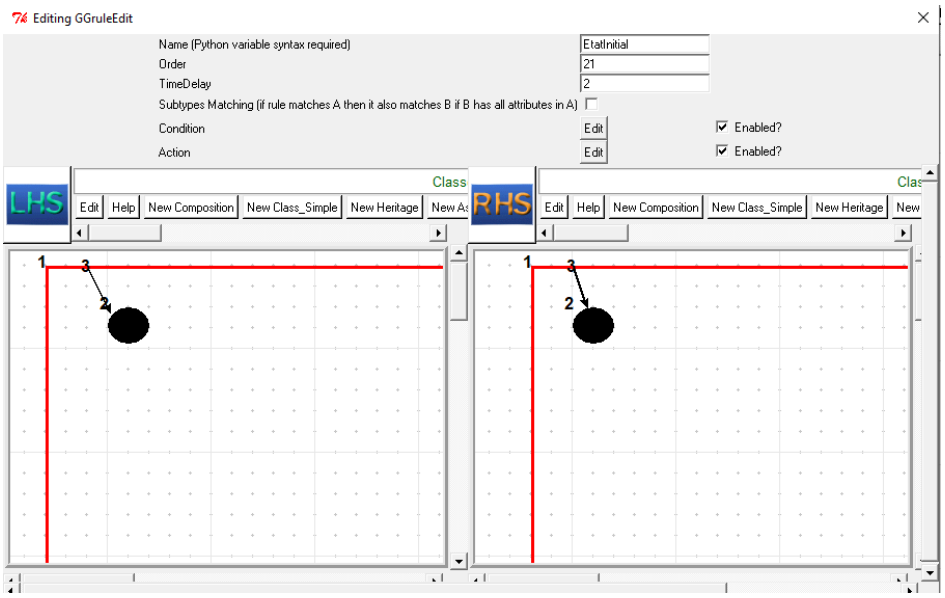


FIGURE 4.21 – La 21 ère règle

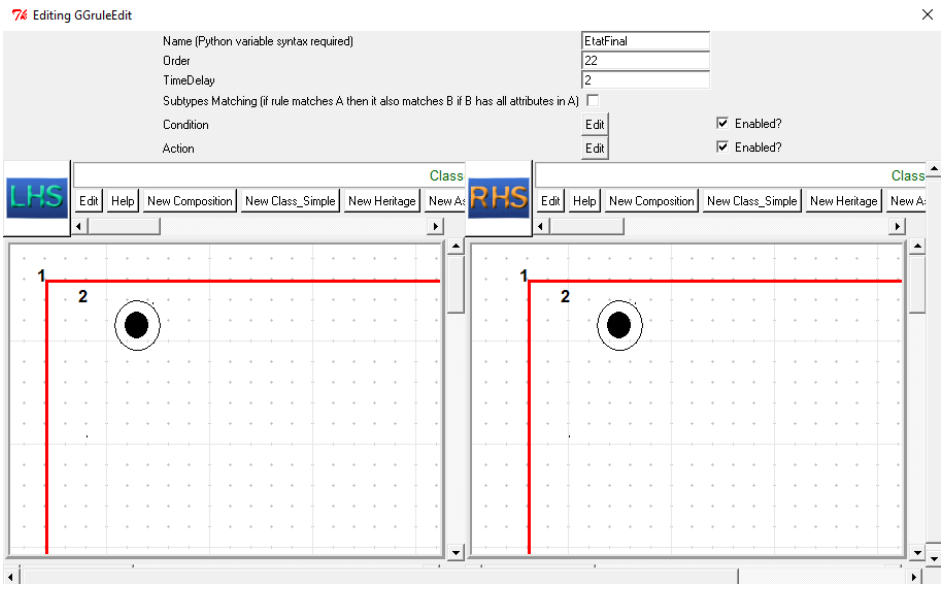


FIGURE 4.22 – La 22 ère règle

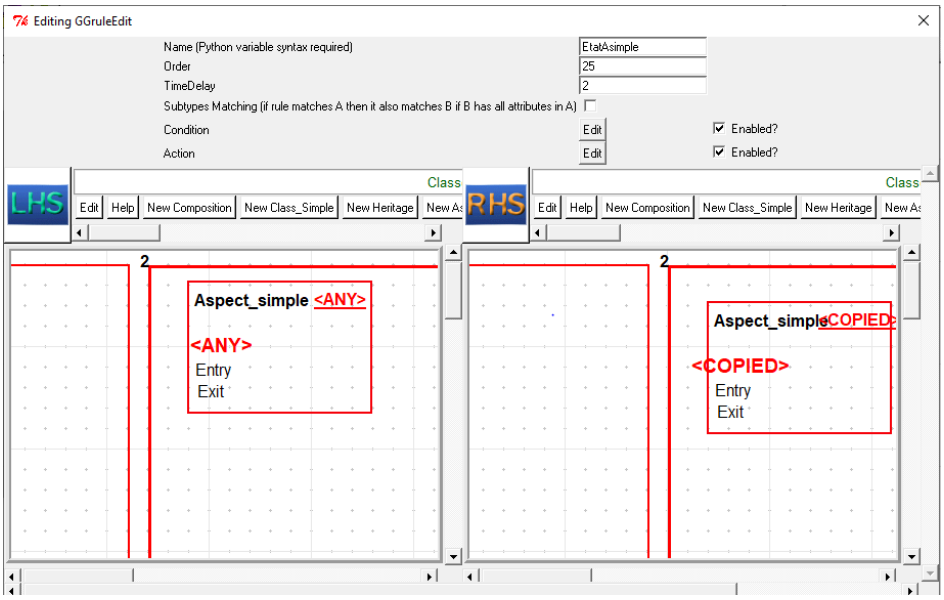


FIGURE 4.23 – La 25 ère règle

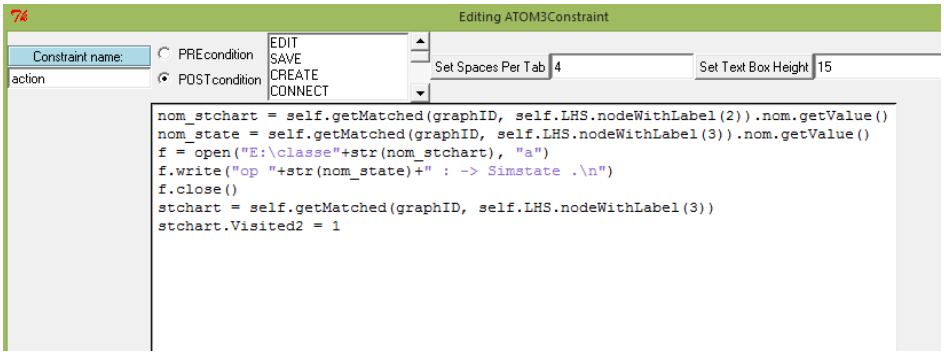


FIGURE 4.24 – Le Code Maude Proposé dans ces règles

Catégorie 02 : cette Catégorie contient 13 Régles (27...39) Ces règles permettent enregistre les événements comme des messages, et le dernière régle (39) appliqué pour déclarer toutes les variables utilisées dans les règles de réécriture. La structure de Ces règles est résumée dans les figures [4.25 et 4.26] :

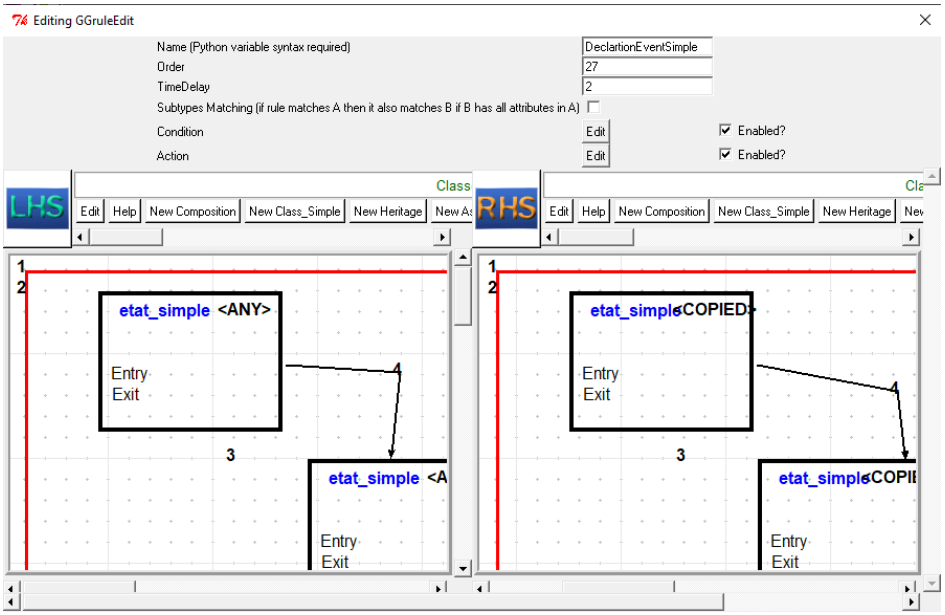


FIGURE 4.25 – La 27 ère règle

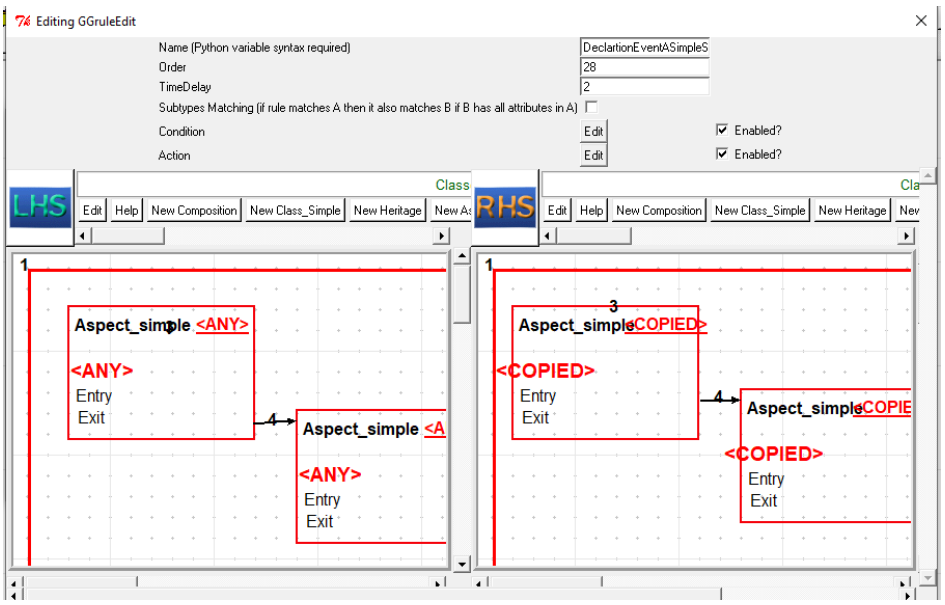


FIGURE 4.26 – La 28 ère règle

Catégorie 03 : cette Catégorie contient 5 Régles (40...53) Ces règles sont appliquées pour marquer la transition comme visitée et générer la spécification Maude correspondante. La structure de Ces règles est résumée dans les figures[4.27 et 4.28] :

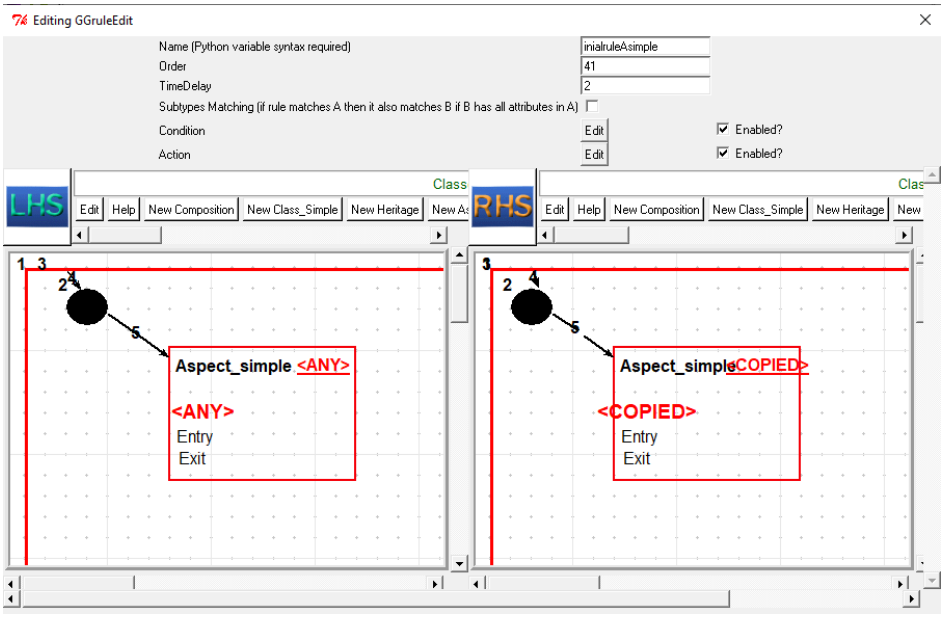


FIGURE 4.27 – La 41 ère règle

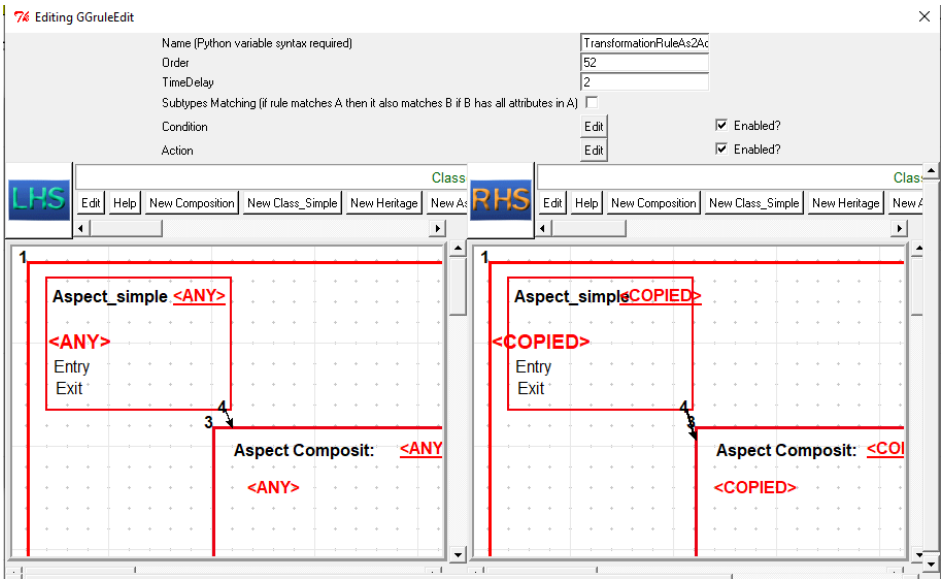


FIGURE 4.28 – La 52 ère règle

Catégorie 04 : cette Catégorie contient 2 Régles (54 , 55) Ces règles sont appliquées pour marque la fin des modules orienté-objets/orienté-Aspect.La structure de Ces règles est résumée dans la figure[4.29] :

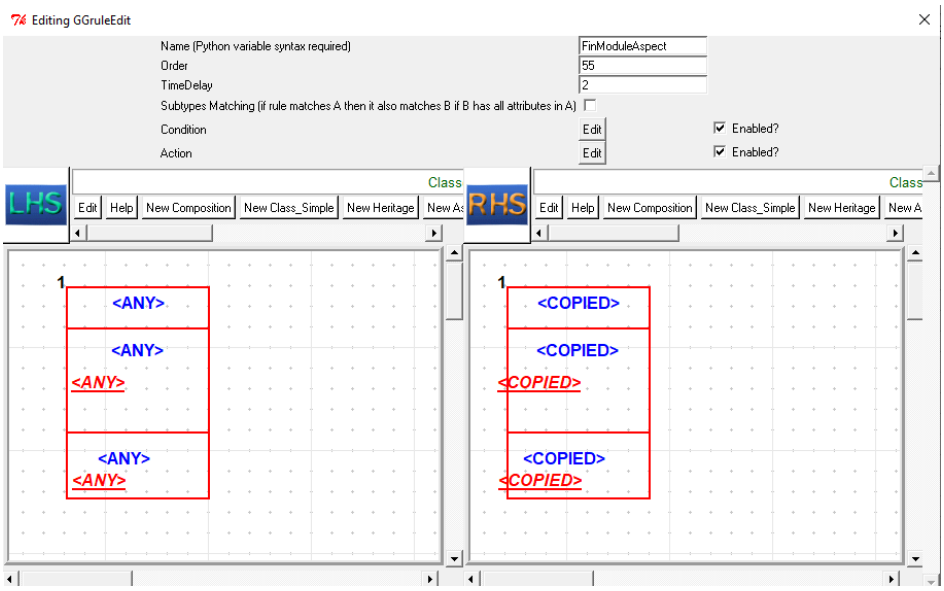


FIGURE 4.29 – La 55 ère règle

4.3 Transformation des diagrammes De Communication orientés aspect vers le langage Maude

Dans cette section nous donnons l'approche de transformation des Diagrammes de Communication orientés Aspect vers le langage Maude. Cette transformation consiste à remplacer chaque entité et association d'un diagramme de Communication orienté aspect par un code génère avec le langage Maude . Nous avons proposé méta-modèle du Diagramme de Communication orienté aspect et une grammaire de graphe. Cette transformation est réalisée à l'aide de l'outil ($AToM^3$).
La structure de cette transformation est résumée dans la figure 4.30 .

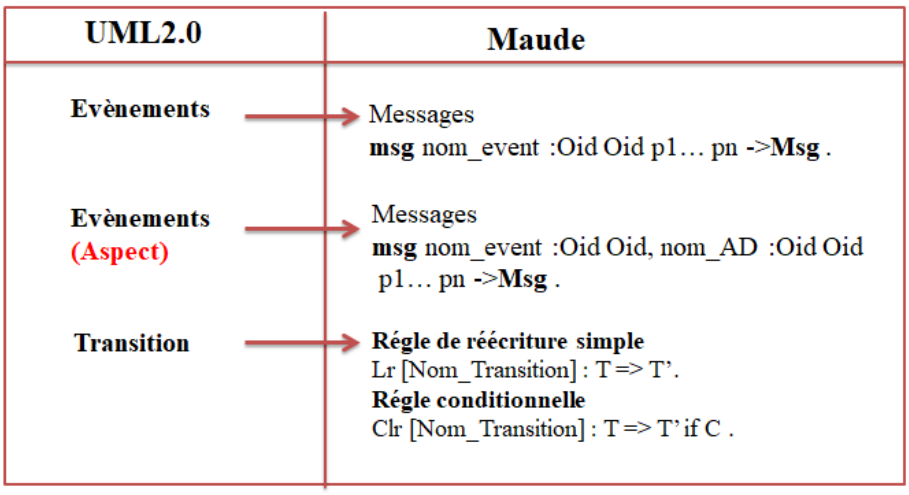


FIGURE 4.30 – Structure de transformation d'un diagramme de Communication orientés aspect vers le langage Maude.

4.3.1 Méta-modélisation des diagrammes De Communication orientés Aspect

Le méta-modèle du diagramme de Communication orienté Aspect composé de trois classes : (DaigrammCommorienteAspect, OAspect, Objet) et deux Associations (contient , lien).
Dans la Figure 4.31, nous présentons méta Modéle pour les diagrammes de Communication orientés Aspect

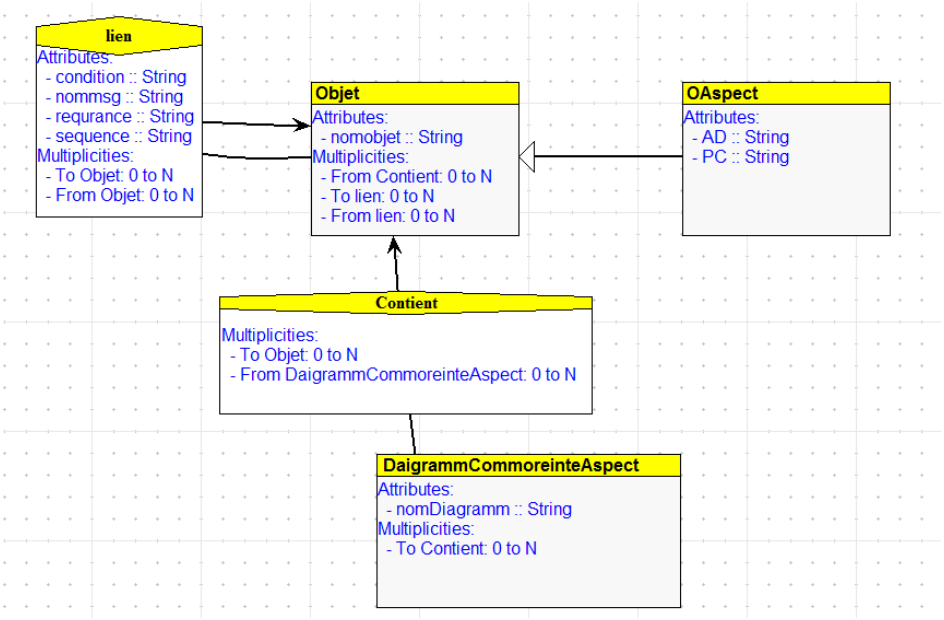


FIGURE 4.31 – Méta modèle pour le diagrammes de Communication orientés aspect.

les Classes :

- DaigrammCommOrinteAspect : Cette classe sert à représenter le diagramme de communication, Graphiquement, elle est représentée par un grand rectangle noire.
- Objet : Cette classe représente les objets ; chaque objet possède un nomobjet et peut communiquer à travers des connecteurs.
- OAspect : Cette classe représente les objets ajoutés (les aspects) au DC et hérite tous ses attributs : multiplicités, associations à partir de la classe Objet.

les Association :

- Contient : C'est une association de composition, qui permet de relier le DaigrammComm avec ses Objets.
- Lien : C'est une association simple, permettant la communication entre deux objets. possède des attributs de type String(condition ,nommsg,requrance,sequence) .Autrement dit, elle représente les messages envoyés ou reçus par un objet.
La figure 4.32 représente l'outil généré pour la manipulation des diagrammes de Communication orientes aspect.

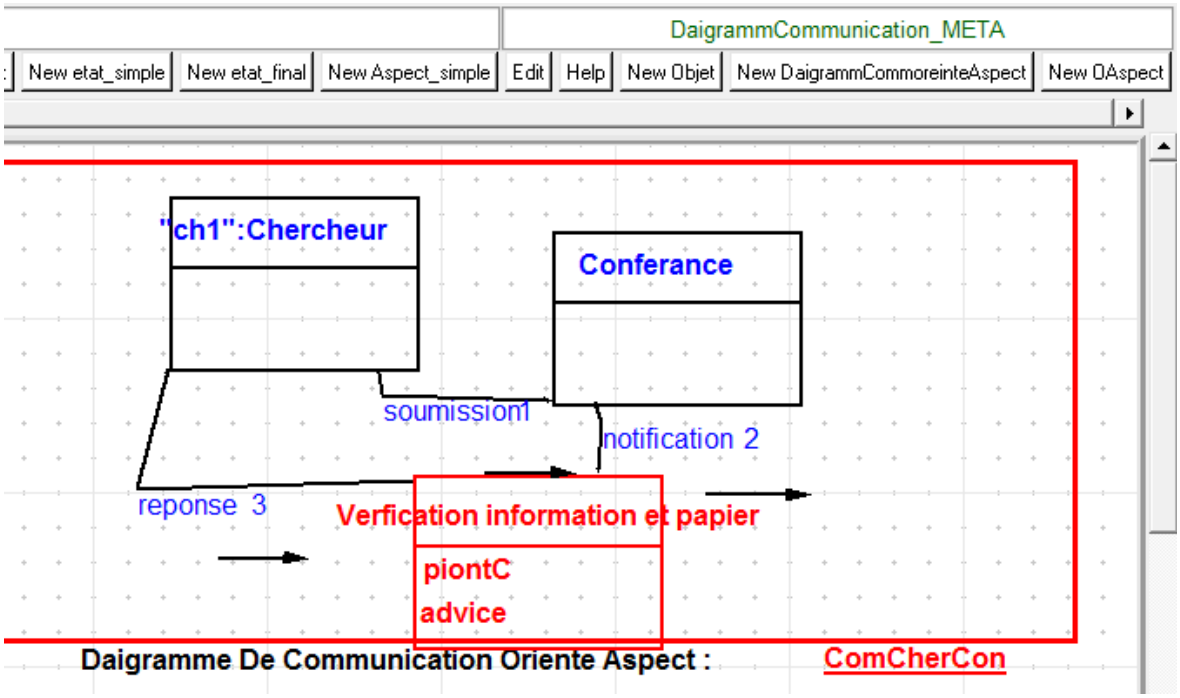


FIGURE 4.32 – l'outil généré pour les diagrammes diagrammes de Communication orientés aspect.

4.3.2 Grammaire de graphes pour la transformation des diagrammes Communication orientés aspect vers un code maude

Dans la Figure 4.33, nous présentons une grammaire de graphes contenant six (6) règles chacune exprime un cas particulier.

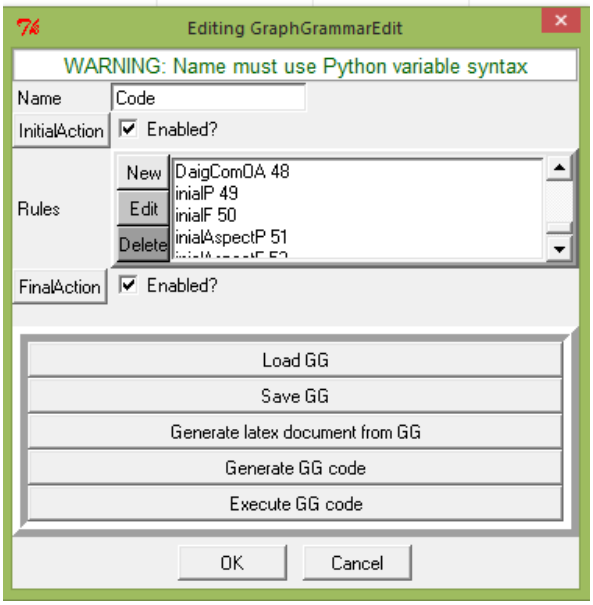


FIGURE 4.33 – Grammaire de graphes pour les diagrammes de Communication orientés aspect.

Nous décomposons les 6 règles en 3 Catégories.
Catégorie 01 : cette Catégorie contient 1 seul Régle (56) Est appliqué pour localiser un diagramme de communication non précédemment traité (Vdiag == 0) et créer un nouveau fichier inclure tous les modules orientés Aspect.
Catégorie 02 : contient 5 Régles (57...60) Ces règles sont appliquées appliquées pour sélectionner un Objet ou Objet Aspect (non traitée précédemment Comm = 0) et générer son code Maude équivalent. La structure de Ces règles est résumée dans les figure[4.34, 4.35, 4.36, 4.37] :

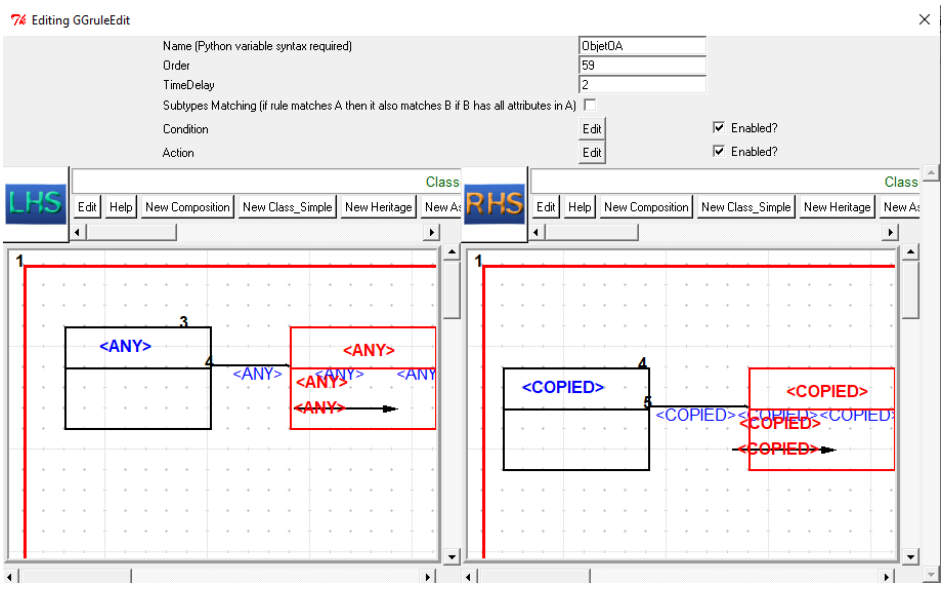


FIGURE 4.34 – La 59 ère règle

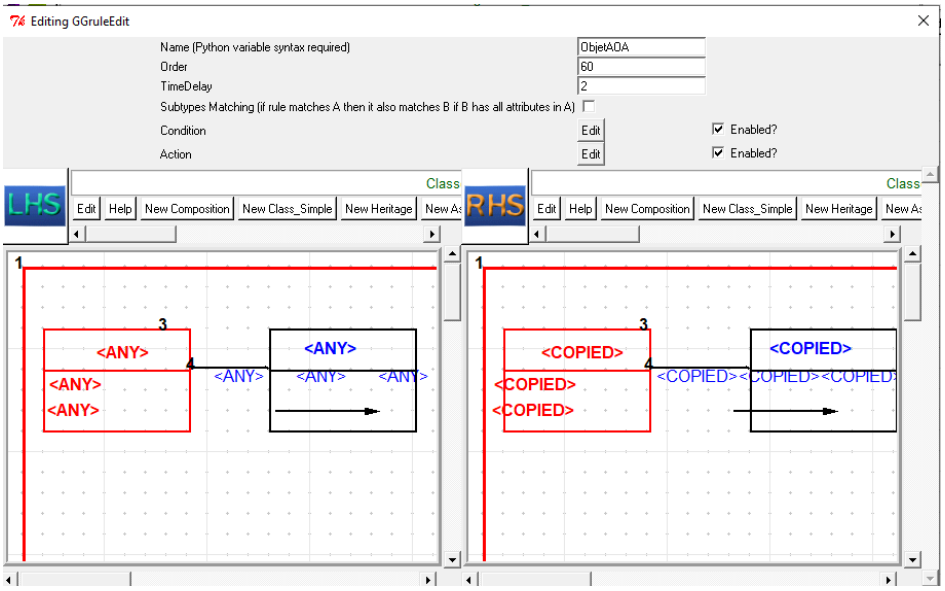


FIGURE 4.35 – La 60 ère règle

Catégorie 03 : cette Catégorie contient 1 seul Régle (61) Est appliqué pour marquer la fin du module orienté Aspect qui est lié au diagramme de communication.La structure de Ce règle est résumée dans les figures[4.38 et 4.39]

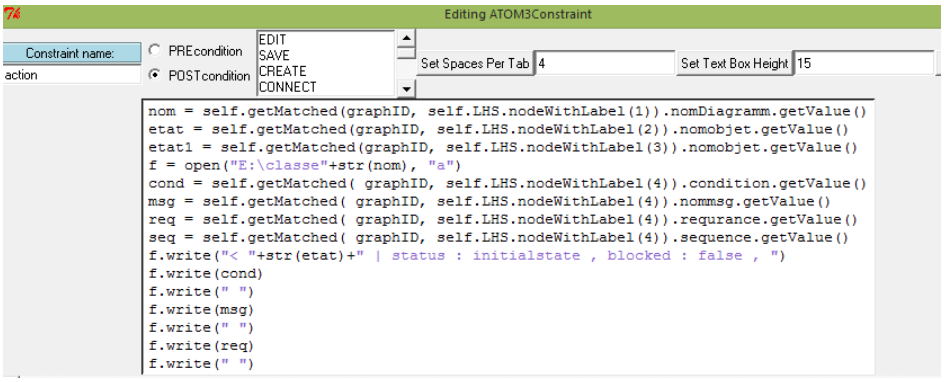


FIGURE 4.36 – Le Code Maude Proposé dans ces règles

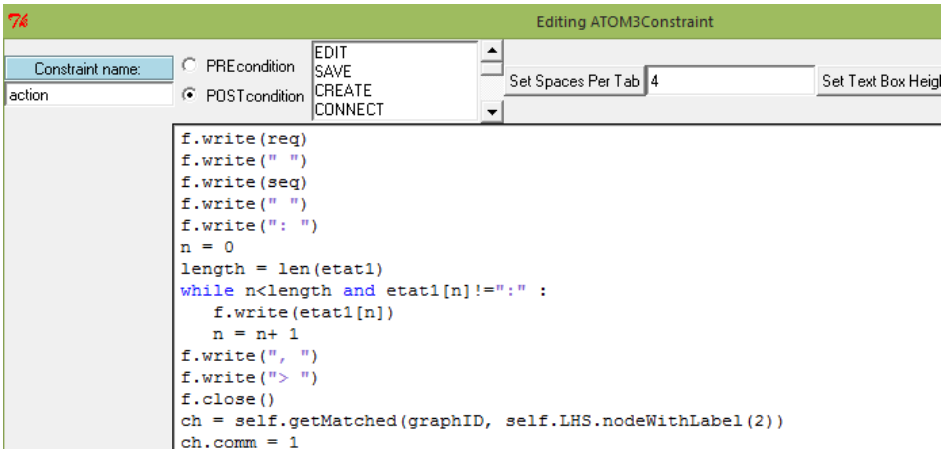


FIGURE 4.37 – Le Code Maude Proposé dans ces règles'Suit'

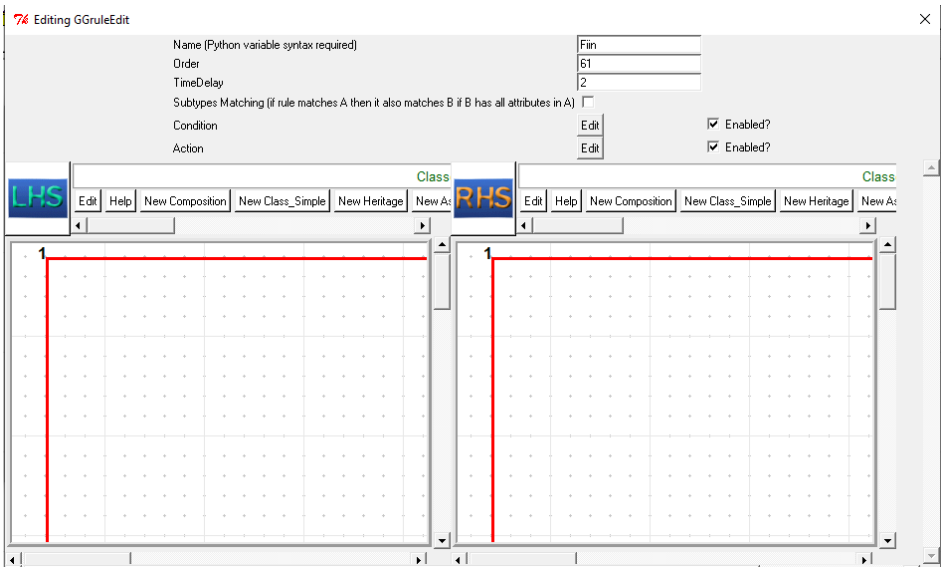


FIGURE 4.38 – La 61 ère règle

Notre grammaire de graphes a également une action finale qui efface toutes les variables globales.

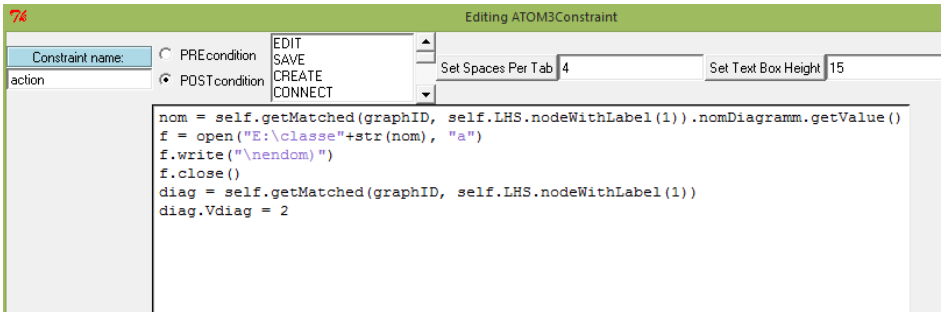


FIGURE 4.39 – Le Code Maude Proposé dans ce règle

Conclusion

Dans ce chapitre, nous avons proposé une approche automatique basée sur la transformation de graphes pour générer un code Maude à partir des diagrammes UML2.0 orientés aspect. Cette approche est basée sur la méta-modélisation (les méta-modèles). L'approche est divisée en trois sous approches :
1. La transformation des diagrammes de classe orientés aspect vers un code Maude.
2. La transformation des diagrammes d'état-transition orientés aspect vers un code Maude.
3. La transformation des diagrammes de communication orientés aspect un code Maude.
Donc, nous avons proposé dans chaque approche la définition du méta-modèle pour le modèle d'entrée ainsi que le code Maude résultat de sortie. Ensuite nous avons présenté une grammaire de graphes-ensemble de règle-permettant la réalisation de la transformation. Cette transformation a été effectuée à l'aide de l'outil de modélisation et de méta-modélisation ($AToM^3$) et les contraintes ont été exprimées en python. Dans le chapitre suivant, nous allons illustrer des exemples de transformation bien discuté dans la littérature.

Chapitre 5

Études de cas sur la transformation des Diagrammes UML 2.0 Orientés Aspect vers le langage Maude

Introduction

Nous avons appliqué notre approche de transformation sur deux études de cas. Le premier sur le problème du diner des philosophes ,le deuxième sur le processus de participation à une conférence.

5.1 Étude de cas sur le problème du diner des philosophes

Pour valider notre approche, nous l'avons appliqué sur le problème du diner des philosophes. Par conséquent, nous avons utilisé trois diagrammes orientés aspect : Classe, état-transition et communication .
Le premier est diagramme de classe Orienté Aspect. Il représente un aspect de vérification qu'il permet de Vérifier que chaque Philosophe possède la fourchette des deux côtes gauche et droite. Le deuxième est le diagramme d'état-transition Orienté Aspect. Il représente des Aspect "VérificationLeftandRight" qu'ils permet de Vérifier l'existence d'une fourchette sur les côtes gauche et droite. Le troisième est le diagramme communication Orienté Aspect. Il représente six aspects, les trois aspects "VerificationRight" ils permet de Vérifier l'existence d'une fourchette sur le côte droite, et les autres "VerificationLeft" ils permet de Vérifier l'existence d'une fourche sur l'autre côte .

5.1.1 les diagrammes de classe, état-transition et communication orientés aspect

le diagramme de classe orienté aspect

Dans la Figure 5.1, nous présentons le diagramme de classe Orienté Aspect(OA).

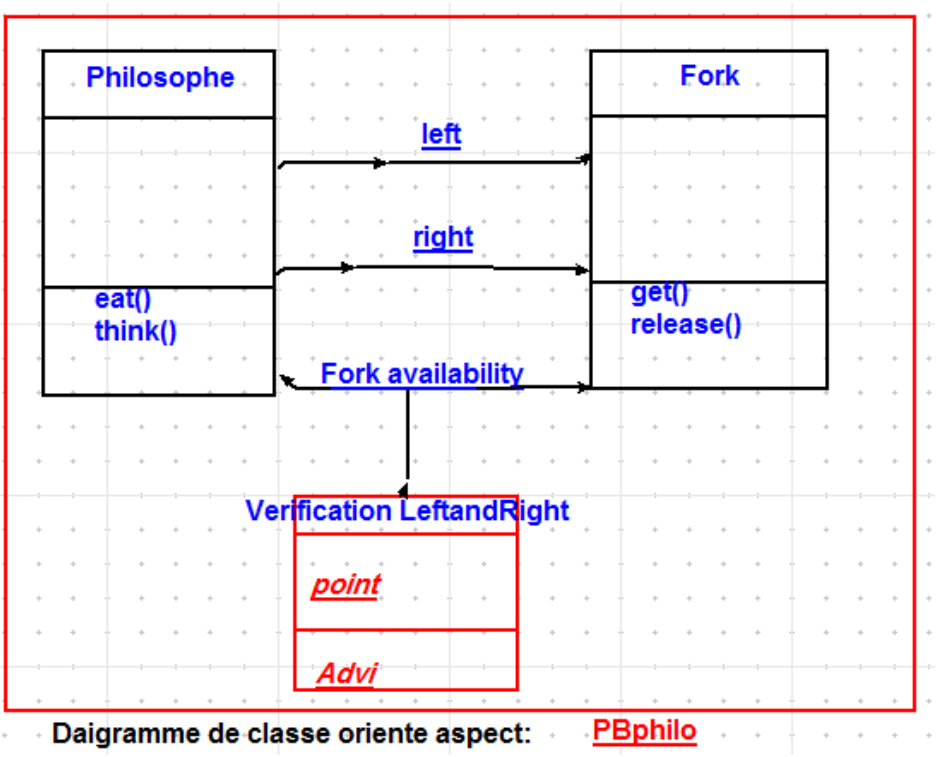


FIGURE 5.1 – le problème du diner des philosophes pour le Diagramme de classe orienté aspect

le diagramme d'état-transition orienté aspect

Dans les Figures 5.2 et 5.3, nous présentons le diagramme d'état-transition Orienté Aspect(OA).

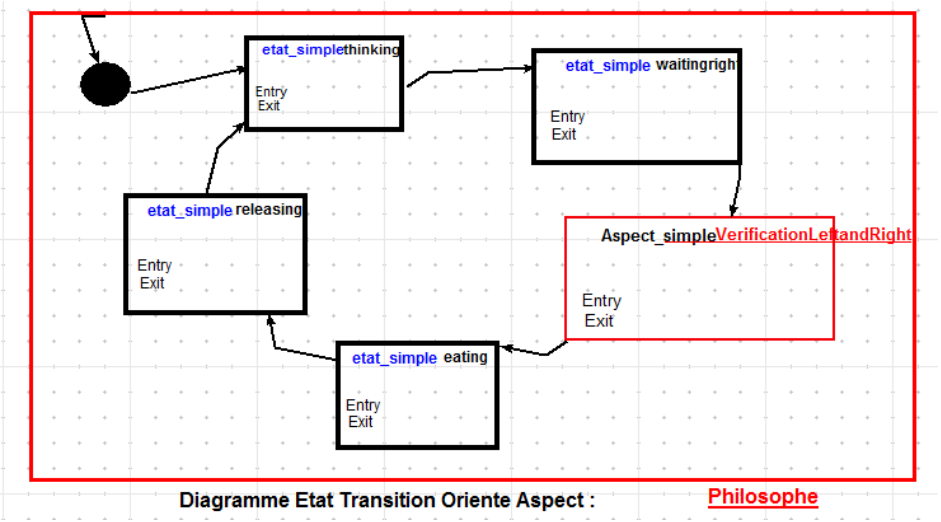


FIGURE 5.2 – le problème du diner des philosophes pour le Diagramme d'état-transition orienté aspect "Philosophe"

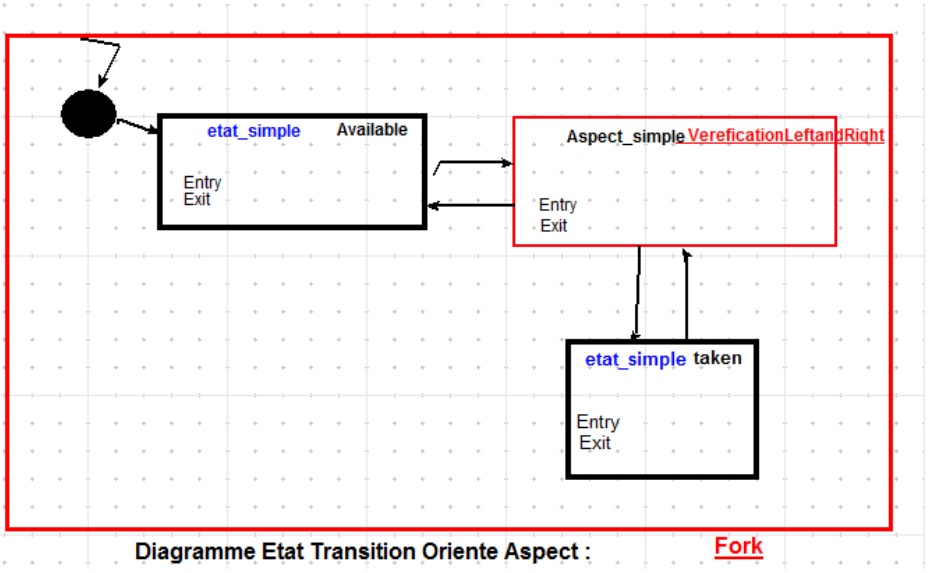


FIGURE 5.3 – le problème du diner des philosophes pour le Diagramme d'état-transition orienté aspect "Fork"

le diagramme de communication orienté aspect

Dans la Figure 5.4, nous présentons le diagramme de communication Orienté Aspect(OA).

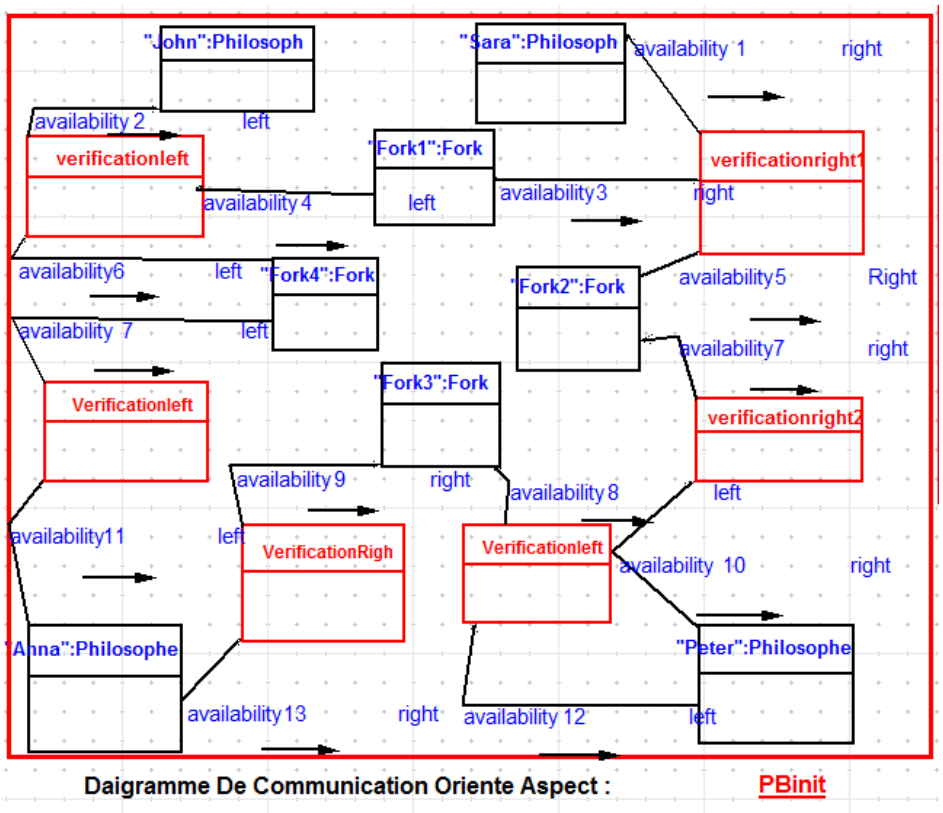


FIGURE 5.4 – le problème du diner des philosophes pour le Diagramme de communication orienté aspect

5.1.2 Résultat Finale en Maude

Après l'application des règles de grammaire on obtient le Code Maude présenté dans les figures suivante[5.5, 5.6, 5.7, 5.8, 5.9] :

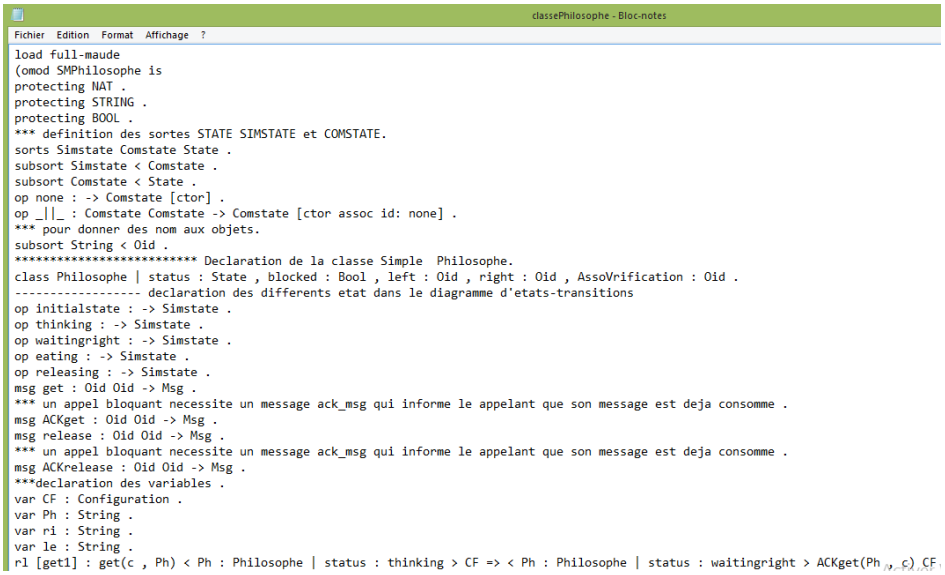


FIGURE 5.5 – Spécification Maude générée de Class Philosopher avec son propre état-transition

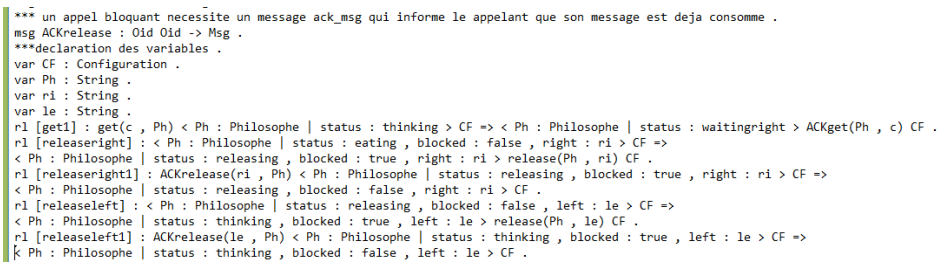


FIGURE 5.6 – Spécification Maude générée de Class Philosopher avec son propre état-transition"Suit"

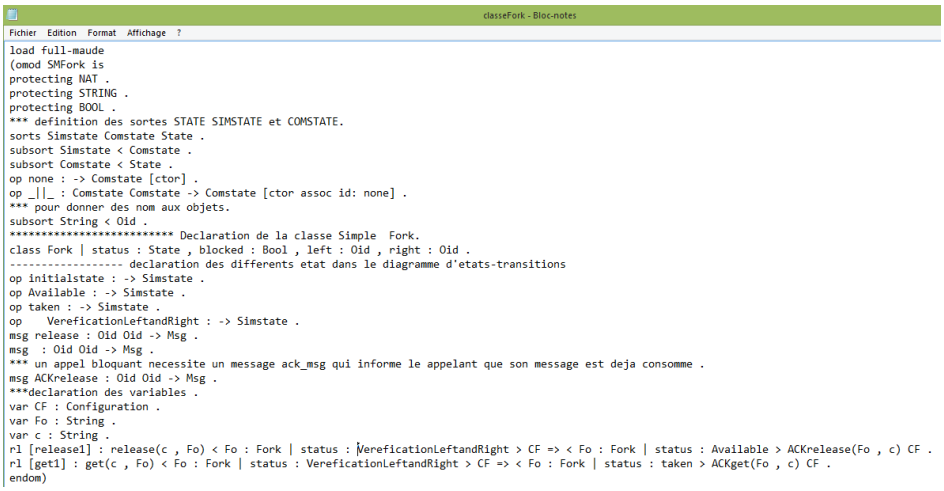


FIGURE 5.7 – Spécification Maude générée de Class Fork avec son propre état-transition

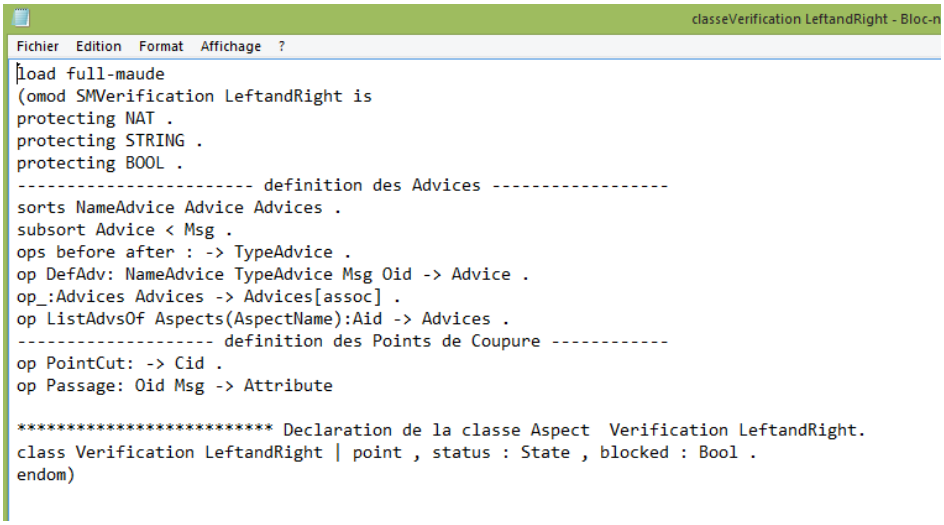


FIGURE 5.8 – Spécification Maude générée Class Aspect de Vérification

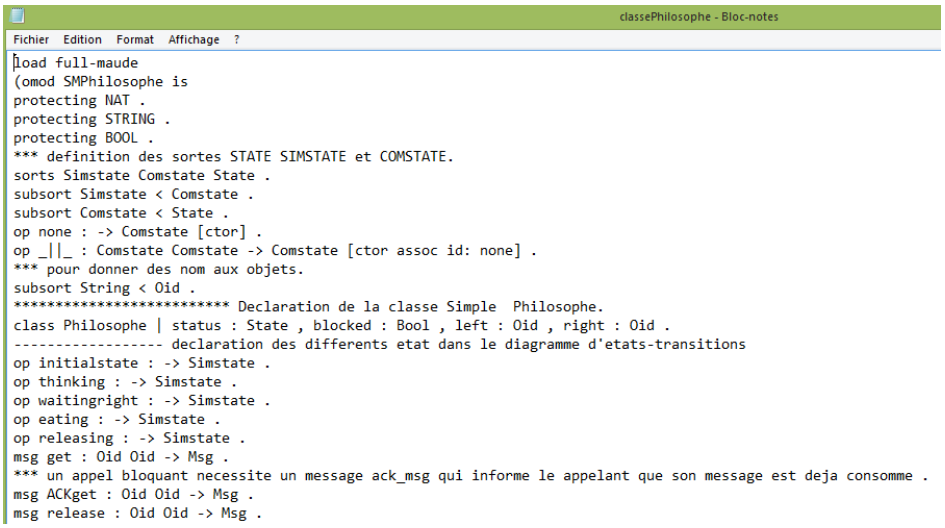


FIGURE 5.9 – Spécification Maude générée du problème du diner des philosophes

5.2 Étude de cas sur le processus de participation a une conférence

Pour illustrer notre approche, nous l'avons appliqué aussi sur le processus de participation à une conférence. Par conséquent, nous avons utilisé trois diagrammes orientés aspect : Classe, état-transition et communication .
Le premier est le diagramme Classe Orienté Aspect. Il représente un aspect qui permet la vérification des informations et les articles. Cet aspect permet de vérifier les informations d'un chercheur et de vérifier si le format de l'article est conforme ou non au format de la conférence aprés la soumission. Le deuxième est le diagramme d'état-transition Orienté Aspect. Il représente quatre aspects (Vérification de l'article, évaluation de la décision, OK de la version finale, Accusé de réception).
-Vérification de l'article : Cet aspect permet de vérifier la structure de l'article selon le format de la conférence.
-Evaluation de la décision : Cet aspect permet d'évaluer la décision des évaluateurs (reviewers) s'il y à un conflit.
-OK de la version finale : Cet aspect permet de contrôler la version finale si elle est corrigée ou non avant de publier l'article.
-Accusé de réception : Cet aspect permet de donner un accusé de réception si l'argent de l'enregistrement de l'article est disponible au niveau du compte de la conférence.
Le troisièmeest le diagramme de Communication Orienté Aspect. Il représente un aspect qui est la vérification d'information et de l'article. cet aspect permet de vérifier les informations d'un chercheur et de vérifier si le format de l'article est conforme ou non au format de la conférence aprés la soumission.

5.2.1 les diagrammes de classe, état-transition et communication orientés aspect

le diagramme de classe orienté aspect

Dans la Figure 5.10, nous présentons le diagramme de classe Orienté Aspect(OA).

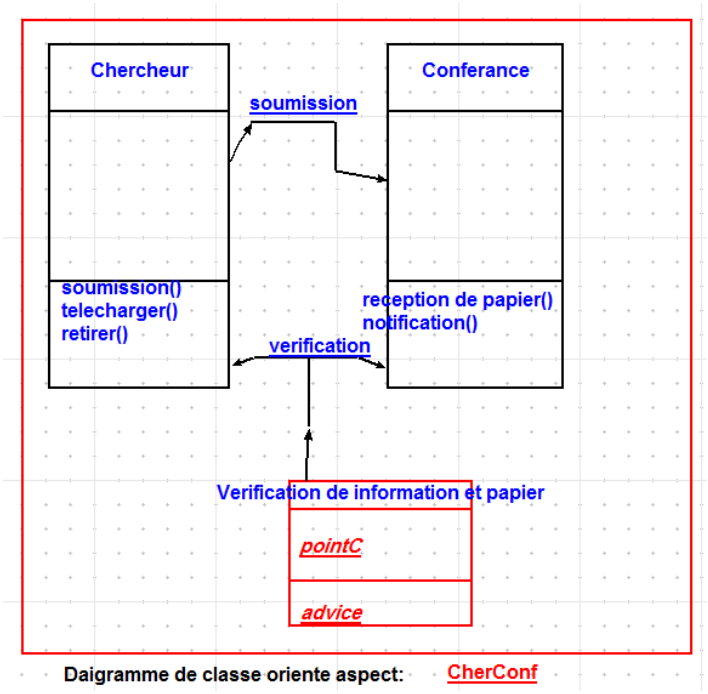


FIGURE 5.10 – le processus de participation à une conférence pour le diagramme de classe orienté aspect

le diagramme d'état-transition orienté aspect

Dans les Figures 5.11 et 5.12, nous présentons le diagramme d'état-transition Orienté Aspect(OA).

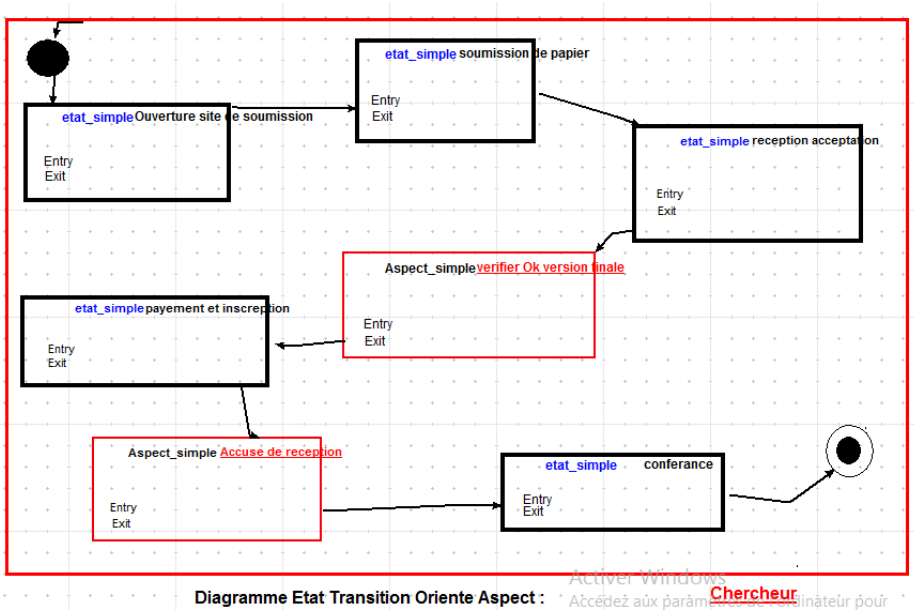


FIGURE 5.11 – le processus de participation à une conférence pour le diagramme d'état-transition orienté aspect"Chercheur"

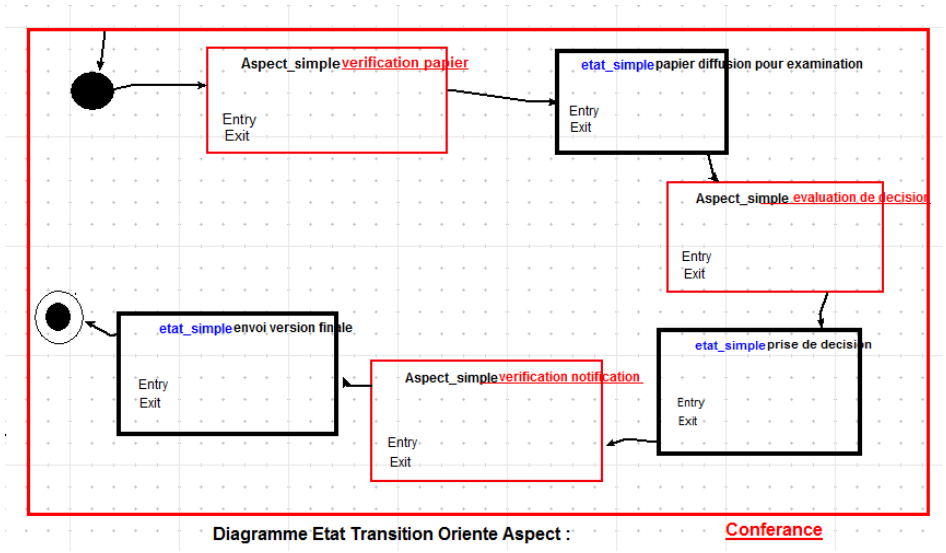


FIGURE 5.12 – le processus de participation à une conférence pour le diagramme d'état-transition orienté aspect"Conférance"

le diagramme de communication orienté aspect

Dans la Figure 5.13, nous présentons le diagramme de Communication Orienté Aspect(OA).

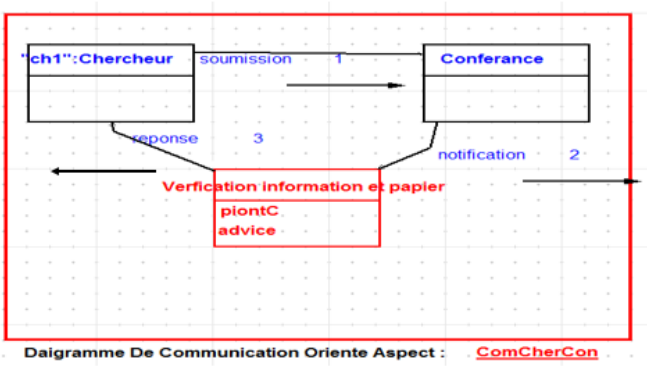


FIGURE 5.13 – le processus de participation à une conférence pour le diagramme de communication orienté aspect

5.2.2 Résultat Finale en Maude

Après l'application des règles de grammaire on obtient le Code Maude présenté dans les figures suivante[5.14, 5.15, 5.16, 5.17, 5.18, 5.19] :

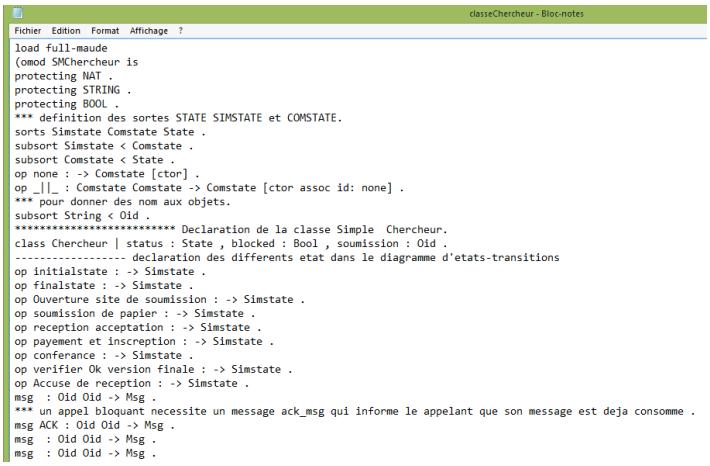


FIGURE 5.14 – Spécification Maude générée de Class Chercheur avec son propre état-transition

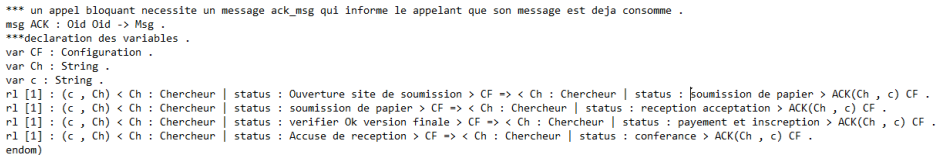


FIGURE 5.15 – Spécification Maude générée de Class Chercheur avec son propre état-transition"suit"

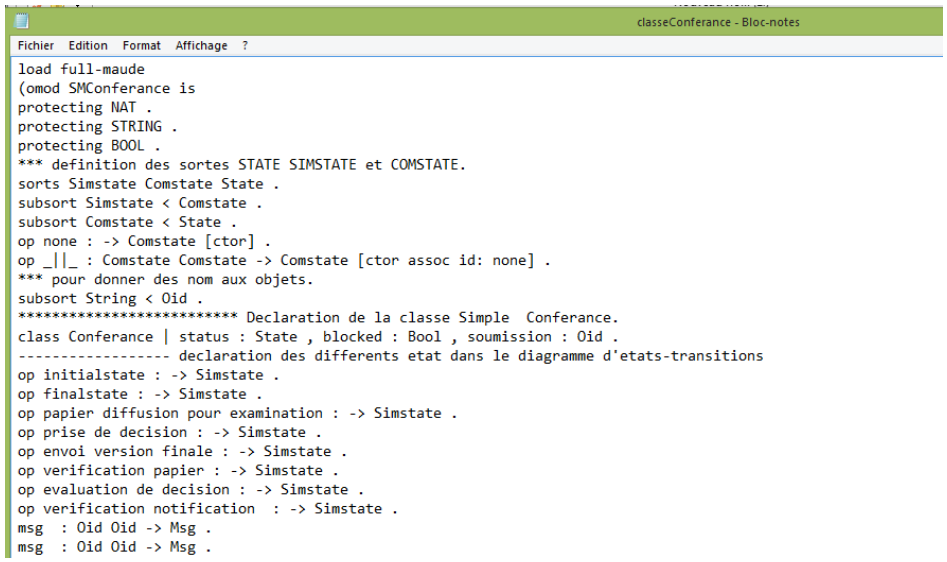


FIGURE 5.16 – Spécification Maude générée de Class Conférance avec son propre état-transition

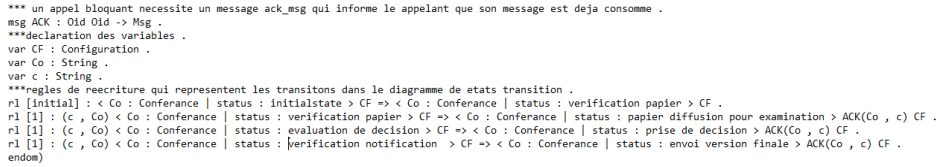


FIGURE 5.17 – Spécification Maude générée de Class Conférance avec son propre état-transition"suit"

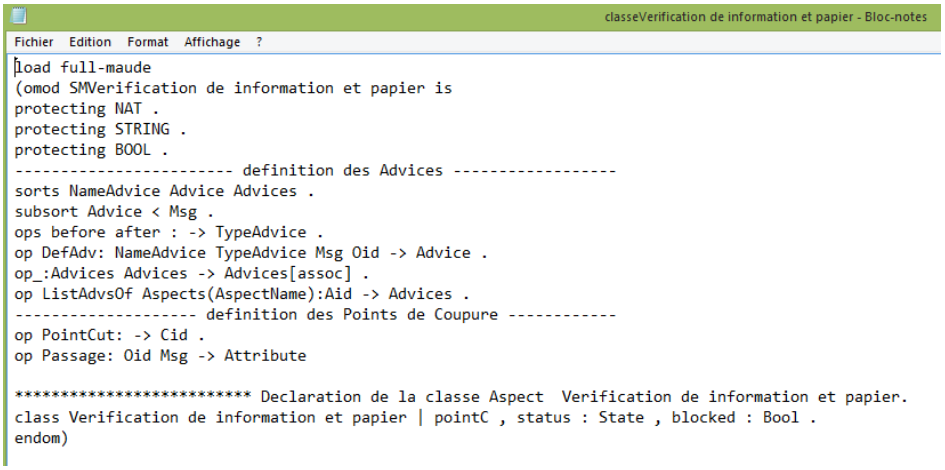


FIGURE 5.18 – Spécification Maude générée Class Aspect de Vérification

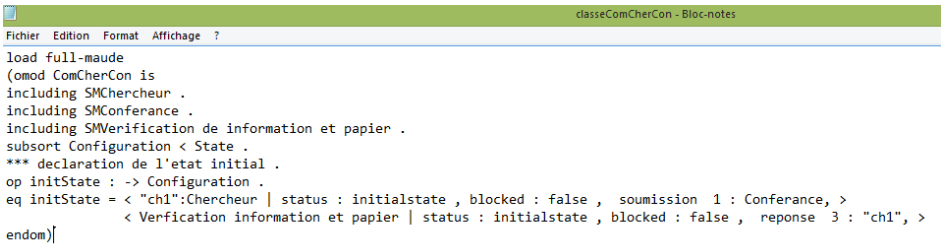


FIGURE 5.19 – Spécification Maude générée du processus de participation à une conférence

Conclusion

Dans ce chapitre nous avons présenté deux études de cas afin d'illustrer notre approche de transformation des diagrammes UML 2.0 orientés aspect Vers le langage Maude . Le premier sur le problème du diner des philosophes. Nous avons définis Des diagrammes de classe, état-transition et communication orientés aspect. L'application de la grammaire de graphes nous a permis d'obtenir automatiquement Code Maude. Le deuxiéme sur le processus de participation à une conférence. Finalement nous avons montré l'efficacité de notre approche a travers les résultats obtenus.

Conclusion et Perspectives

Conclusion

Le travail présenté dans ce mémoire s'inscrit dans le domaine de l'ingénierie dirigée par les modèles. Il se base essentiellement sur l'utilisation combinée de méta modélisation et de transformation de modèle. Plus précisément, la transformation de graphes est utilisée comme outil de transformation de modèles. Le résultat de notre travail est une approche pour transformer les diagrammes de classes ,de Communication et d'état Transition orientés aspect vers le Langage de spécification Formelle Maude. L'approche proposée est basée sur la transformation de graphes. Nous avons choisi la méthode de transformation de graphe parmi toutes les méthodes de transformation de modèle car elle est largement utilisée dans la littérature et de nombreux outils ont été développés pour rendre ces transformations automatiques et lisibles , parmi ces outils on a choisi l'outil ($AToM^3$) comme un bon outil de modélisation, multi formalismes, méta modélisation et de transformation de graphes. La transformation de graphes avec ($AToM^3$) est une opération simple et efficace de par leur sémantique. Il constitue la base pour l'intégration d'autres nouveaux outils.
Notre travail est réalisé dans deux étapes :
- La première étape consiste à proposer trois méta-modèles. Un pour les diagrammes de classes orientés aspect , la deuxième pour les diagrammes de communication orientés aspect et l'autre pour les diagrammes d'état de transition orientés aspect afin de générer un outil visuel permettant la manipulation de ces modèles.
- La deuxième étape sert à proposer une grammaire de graphes permettant la transformation de ces diagrammes orientés aspect vers le langage Maude.
De ce fait, l'outil ($AToM^3$) basé sur la méta- modélisation est choisi dans notre memoire, nous avons proposé une approche et un nouvel outil de transformation qui permet de transformer certains Diagrammes UML 2.0 orientés aspect vers le langage Maude en utilisant la grammaire de graphes. Nous avons proposé une grammaire de graphes qui effectue le processus de transformation. Finalement, Nous avons appliqué notre approche sur des études de cas et obtenu de bons résultats.

Perspectives

Dans la afin d'arriver à développer une approche totalement automatique, incluant tous les diagrammes UML 2.0, nous proposerons :
- Continuer la transformation des autres diagrammes UML 2.0 (diagramme de séquence orienté aspect, diagramme de cas d'utilisation orienté aspect, diagramme. . . , orientée aspect, etc. . . .) vers le langage de spécification Maude en utilisant la transformation de graphes et l'outil ($AToM^3$) .

- Amélioration des méta-modèles et de grammaire de graphe.
- Développer une approche totalement automatique incluant tous les diagrammes UML2.0 orienté-aspect Vers le langage Maude bassé sur la transformation de graphes et l'utilisation de l'environnement AToMPM.
- L'utilisation de LTL model-checker du langage Maude pour faire la vérification des modèles .

Références Bibliographiques

[Amroune, 2014] Amroune , Moammed.(2014).Vers une Approche Orientée Aspect d'Ingénierie des Besoins dans les Organisations multi Entreprises. Thèse de doctorat. Université de Toulouse, Toulouse.pp :15-20.

[Aouag, 2014] Aouag,Mouna. (2014).Des diagrammes UML 2.0 vers les diagrammes orientés aspect à l'aide de transformation de graphes. Thèse de doctorat, Université de Mentouri, Constantine.pp :8-43.

[Bahri, 2011] Bahri, Mohamed Rédha .(2011).Une approche intégrée Mobile-UML/Réseaux de Petri pour l'Analyse des systèmes distribués à base d'agents mobiles. Thèse de doctorat, Université de Mentouri, Constantine.pp : 67-68.

[Blanc, 2005] Blanc, X. (2005). Mda en action : Ingénierie logicielle guidée par les modèles. Eyrolles.

[Boubendir, 2011] Boubendir, Amel. (2011). Un cadre générique pour la détection et la résoulution des intéraction entre les aspects. Thèse de doctorat, Université de Mentouri, Constantine.pp :35-36.

[Brichau et D'Hondt, 2005] Brichau, J. et D'Hondt, T. (30 August 2005). Introduction to aspect-oriented software development, aosd-europe.

[Chama, 2011] Chama,Wafa. (2011). Une Approche basée transformation de graphes pour la génération de spécifications Maude à partir de diagrammes UML. Rapport de stage, Université de Mentouri Constantine, Constantine pp:7-9.

[Czarnecki et Helsen, 2003] Czarnecki, K. et Helsen, S. (2003). Classification of model transformation approaches. OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture.

[Czarnecki et Helsen, 2006] Czarnecki, K. et Helsen, S. (2006). Featurebased survey of model transformation approaches. IBM SYSTEMS JOURNAL, 45(03).

[Dehimi, 2014] Dehimi,Nardjess,Tissilia.(2014). Un Cadre Formel pour La Modélisation et L'analyse Des Agents Mobiles. Thèse de Doctorat. Université de Mentouri ,Constantine. pp :51-52.

[Douibi,2006] Douibi.Halima. (2006).Intégration de XML dans le cadre de la logique de réécriture. Mémoire de master. Université Mentouri Constantine, Constantine.pp :23-24.

[Hettab, 2009] Hettab, Abdelkamel. (2009).De M-UML vers les réseaux de Petri < Nested Nets > : Une approche basée transformation de graphes.Thèse de doctorat.Université de Mentouri, Constantine.pp :59-63.

[Kerkouche, 2011] Kerkouche, Elhillali. (2011).Modélisation Multi-Paradigme : Une Approche Basée sur la Transformation de Graphes. Thèse de doctorat. Université de Mentouri, Constantine.pp :8-52.

[Mérouani, 2011] Mérouani,Hamza.(2011).Validation de la Composition des Services Web à l'aide du Langage Maude.Mémoire de master. Université Larbi Ben M'Hidi Oum El Bouaghi, Oum El Bouaghi.pp :65-69.

[Muller, 2006] Muller, Pierre-Alain.(2006).De la modélisation objet des logiciels à la metamodélisation des langages informatiques. Thèse de doctorat.Université de rennes 1 ,Rennes. pp :20.

[Nouri,2020] Nouri,Zakarya.(2020). Spécification formelle des systèmes multi-agents normatifs - Une approche basée sur le langage MAUDE .Mémoire de master. Université d'Oum El Bouaghi, Oum El Bouaghi.pp :22-35.

[Otmane Rachedi, 2015] Otmane rachedi ,Soumeya.(2015). Apports des Approches de Séparation Avancée des Préoccupations : Une Etude Comparative Fondée sur les Modèles de Conception. Thèse de doctorat. Université de Badji mokhtar ,annaba. pp :16-20.

[Terchi et Mezahi, 2018] Terchi, Roqiya.et Mezahi, Bochra.(2018). Le Développement d'un Outil de Transformation des Modèles Orientés Aspect vers les Réseaux de Pétri, Basé sur la Transformation De Graphes.Mémoire de master. Centre Universitaire Abd Elhafid Boussouf Mila, Mila.pp :18-38.

[Wautelet et al, 2004] Wautelet, Yves .et Louvigny, Laurent .et Manuel, Kolp. (2004). Modélisation oriente-objet d'aspect opérationnel de bases de donnes siderurgique. IAG – ISYS (Unité de Systèmes d'Information), Université Catholique de Louvain,1 Place des Doyens, 1348 Louvain-la-Neuve, Belgique .

Spécification formelle des Modèles Orientés Aspect, une approche basée sur la Transformation De Graphes et le Langage Maude

Aouag Mouna, Dib Wiam, Saadaoui Amani.
Département de Mathématiques et Informatique
Centre Universitaire Abd elhafid boussouf Mila
Algérie
aouag.mouna@centre-univ-mila.dz,
wiamd123@gmail.com ,
saadaouiamani36@gmail.com .

Résumé :La modélisation en informatique est l'étape la plus importante dans le développement d'un logiciel. Elle facilite la compréhension du fonctionnement d'un système avant sa réalisation en produisant un modèle. Ils existent plusieurs méthodes de modélisation comme la Modélisation Orientée Aspect(MOA) et la Modélisation Orientée Objet(MOO). Avec l'utilisation de toute la puissance des langages orientés objet, l'approche orientée objet montré ces importances et efficacités depuis ces apparition dans les systèmes complexes. Mais avec l'évolution de l'informatique, les problèmes deviennent plus complexes, de nombreuses limites de cette approche ont été trouvées. Par ailleurs, la modélisation orientée aspect a montré son utilité dans la conception et le développement des systèmes complexes, pour cela il existe plusieurs modèles qui sont orientés aspects, mais les diagrammes UML2.0 Orientés Aspects ne possède pas de sémantique. En plus, il n'existe pas des outils qui permettent de vérifier et valider ces modèles. Donc, nous avons proposé une approche de transformation des modèles orientés aspect vers des modèles formels.
Dans ce mémoire, nous avons proposé une transformation des diagrammes de classes, etat-transition et communication orientés aspect vers le langage Maude en se basant sur le paradigme de la transformation de graphe. Notre approche consiste à proposer des méta modèles (les modèles orientés aspect et Maude), une grammaire de graphe et des règles pour la transformation entre deux formalismes différents. Finalement on va argumenter notre proposition avec des études de cas bien illustrées.

Mots clés: Le langage Maude, Grammaire de graphes, AToM^3 , Logique de réécriture, Modélisation orienté aspect, Transformation de graphes.

1. Introduction

Le langage de modélisation unifié UML est un langage standardisé compatible avec les méthodes de développement orienté objet. Il est utilisé pour spécifier et visualiser des systèmes logiciels depuis 1980. Il comprend un large ensemble de notations graphiques appelées diagrammes (treize diagrammes) qui permettent de représenter

respectivement les vues statique (structurelle) et dynamique (comportement) d'un système. Le diagramme de classes décrit une vue statique du système et modélise les relations entre les classes en utilisant les associations, la composition, l'héritage, etc… [1], [2], [3], [4].

Avec l'évolution dans le domaine de l'informatique, de nombreux problèmes deviennent plus complexes. Par conséquent, de nombreuses limites de La Modélisation Orientée Objet ont été trouvées telles que la duplication, le souci transversal, la résolution et la réutilisation des modèles. Pour cela, les développeurs et les programmeurs ont proposé La Modélisation Orientée Aspect pour dépasser les limites mentionnées ci-dessus.

Dans cet article, nous proposons une approche et un outil pour la transformation automatique de diagramme (classe, état-transition, communication) orientés aspect en Le langage Maude.

Le reste de l'article est organisé comme suit. La section 2 décrit les travaux connexes les plus pertinents et donne les différences entre leurs multiples approches. Dans la section 3 présente brièvement le système de réécriture et le langage Maude. Dans la section 4, nous donnons une brève introduction de l'outil AToM3. La section 5 détaille la traduction proposée en définissant les trois méta-modèles de diagrammes UML2.0 utilisés (Diagramme de classe , State Chart Diagramme, d'état-transition et Diagramme de communication) orienté aspect et en donnant les règles de la grammaire des graphes proposée , tandis que la section 6 décrit une étude de cas afin d'illustrer notre approche de la traduction. Enfin, nous donnons une conclusion et quelques perspectives dans la section 7.

2. Travaux connexes

Dans [5], les auteurs ont présenté quelques règles pour faire correspondre les diagrammes UML à leurs spécifications Maude équivalentes. Dans [6] , les auteurs ont ajouté le concept d'Aspect dans trois diagramme UML2.0 .La traduction est faite manuellement. Dans [7], l'auteur a présenté une autre approche pour transformer les diagrammes UML en leurs spécifications Maude équivalentes. La traduction est également faite manuellement. Dans cet article, nous proposons une approche automatique et un environnement d'outils qui transforment formellement les diagrammes UML2.0 orienté aspect en leurs spécifications Maude équivalentes à l'aide de l'outil de méta-modélisation AToM3 et des grammaires de graphes. Notre approche est inspirée des travaux présentés dans [8].

3. La logique de Réécriture et Maude

La logique de réécriture [9] a été introduite par José Meseguer permettant la spécification et la vérification simultanées de logiciels. Il est implémenté par plusieurs langages tels que Maude [10].

Maude est un langage de spécification et de programmation. Il est simple, expressif et a une mise en œuvre performante.

Maude définit trois types de modules : les modules fonctionnels, les modules système et les modules orientés objet.

Les modules fonctionnels permettent de définir les types de données et leurs propriétés par la définition de signatures et d'équations ; mais le comportement dynamique d'un système est défini par l'utilisation de lois de réécriture que nous introduisons dans les modules System, ces lois prennent la forme (1) .

$$R : [t] [t'] \text{ si } C \qquad (1)$$

Ce qui indique que, selon la règle R, le terme t se réécrit en t′ si une certaine condition C est vérifiée. La condition C est facultative, les règles peuvent donc être inconditionnelles. Enfin, les modules orientés objet ajoutent une syntaxe plus appropriée pour décrire le paradigme objet tel que les objets, les messages et les configurations. Maude propose « full Maude » pour accompagner cela ; en outre, il possède son propre vérificateur de modèle qui est utilisé pour vérifier les propriétés du système.

4. La Modélisation Oreinté Aspect

Est un nouveau paradigme, développé par Kiczales. G et son équipe en 1996 [10]. Il se concentre sur les concepts connus sous le nom d'aspects. Il permet la séparation d'un modèle de base (MB) et d'un modèle aspectuel (MA). La modélisation orientée aspect est une intégration du modèle aspect dans le modèle de base en fonction de certains points définis : Pointcut = Joinpoints et conseils.

Joinpoint (JP) : Placer dans le modèle où le conseil doit être inséré.

Pointcut (PC) : L'ensemble de JP, souvent une expression régulière et signifie le choix des JP pour l'application Advices.

Conseil (AD) : partie du modèle qui contient tout ou partie de l'aspect inséré à un point de jonction.

5. L'approche proposée

Les étapes de notre approche proposée sont les suivantes :

5.1. Méta-modélisation des diagrammes UML2.0 utilisés

Afin de traduire les diagrammes UML2.0 en spécifications Maude équivalentes, nous proposons trois méta-modèles : le premier pour le diagramme de classe orienté aspect , le deuxième pour le diagramme d'état-transition orienté aspect et le troisième pour le diagramme de communication orienté aspect. Ces méta modèles sont représentés par le formalisme UML2.0 Class Diagramme et les contraintes sont exprimées en code Python.

5.1.1. Méta-modèle de diagramme de classe orienté aspect

Le méta-modèle du diagramme de classes orienté aspect composé de huit classes (diagramclassaspect, Class-simple, Composition, Agrigation, Heritage (inheritance), Association, Association-naire, Aspect) et sept Associations. Dans la Figure, nous présentons l'outil généré pour la manipulation des diagrammes de classes orienté aspect

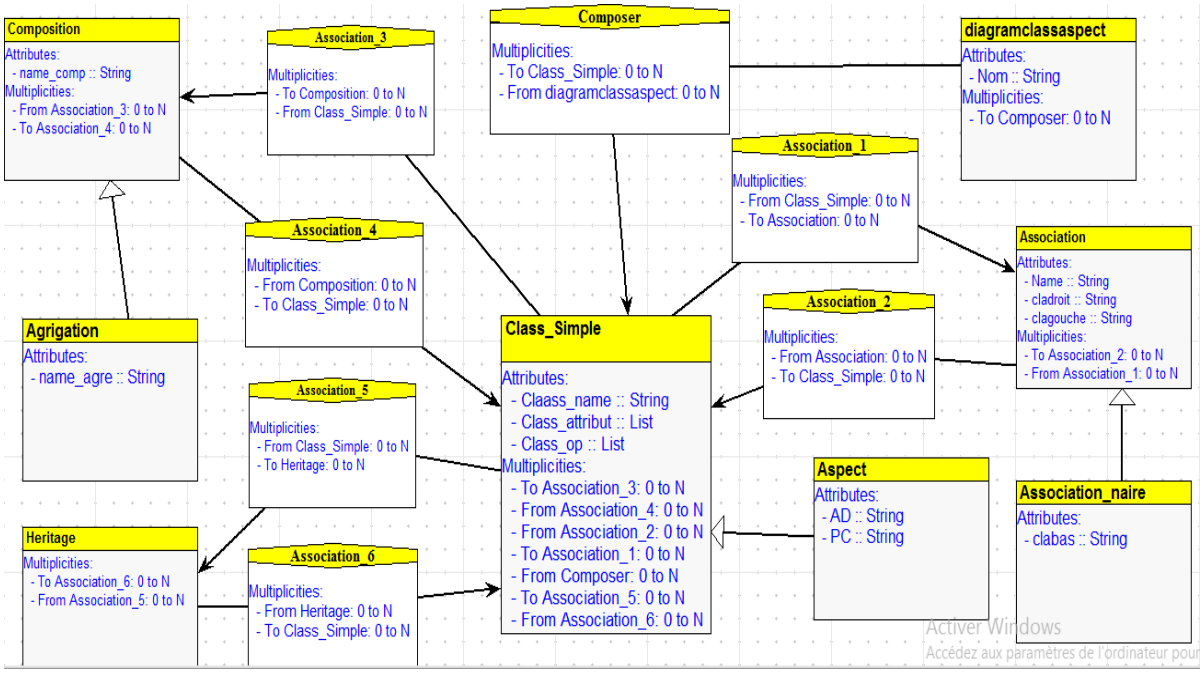


Figure 1 : Méta modéle pour le diagramme de classes orienté aspect.

Notre méta-modèle pour le diagramme de classes UML2.0 (voir Figure 1) est composé des classes suivantes :

a) **Diagramclassaspect :** cette classe représente le diagramme de classe orienté aspect. Elle posséde un attribut « Nom » de type String. Graphiquement, elle est représentée par un grand rectangle rouge.

b) **Class-simple** : cette classe représente les classe-simple. Elle posséde trois attributs « Claas – name » de type string, « Class – attribut »et « Class − op « de type list. Graphiquement, elle est représentée par une classe noire.

c) **Aspect** : est une classe qui hérite la classe « Class – Simple », elle hérite tous les attributs de la « Class – Simple » et plus deux attributs « AD »et « P C » de type string. Graphiquement, elle est représentée par une classe rouge.

d) **Association :** cette classe représente les associations entre deux classes. Graphiquement elle est représentée par un lien, elle posséde trois attributs « N ame » « cladroit » et « clagouche » de type string.

e) **Association-naire** : est une classe qui hérite la classe « Association »,elle hérite tous les attributs de la classe « Association » et plus l'attribut « clabase » de type string. Cette classe représente les associations entre plusieurs classes. Graphiquement elle est représentée par un lien.

f) **Composition** : cette classe représente la composition entre deux classes. Elle posséde l'attribut « name−comp » de type string. Graphiquement elle est représentée par un losange plain.

g) **Agrégation** : est une classe qui hérite la classe composition, cette classe représente l'agrégation entre deux classes. Elle posséde l'attribut « name−agre » de type string. Graphiquement elle est représentée par un losange vide.

h) **Heritage** : cette classe représente l'héritage entre deux classes. Graphiquement elle est représentée par une flache.

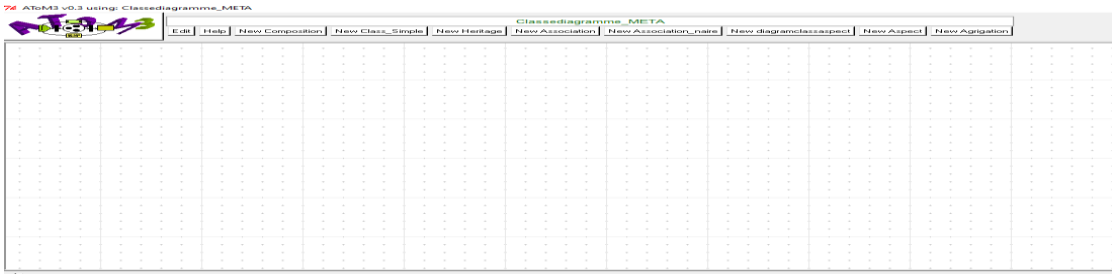


Figure 2 : l'outil généré pour les diagrammes de classes orientés aspect.

5.1.2. Méta-modèle de diagramme d'état-transition orienté aspect

Le méta-modèle du diagramme d'état-transition orienté aspect composé de sept classes (etat-initial, etat-final, etat-simple, etat-composit, Aspect-simple, Aspect-composit, diag-ET-OA) et dix Associations. Notre méta-modèle contient :

a) **diag-ET-OA** : Cette classe a un attribut qui est le Nom, elle représente les ´états dans le diagramme. Graphiquement, elle est représentée par un grand rectangle rouge.
b) **etat-simple** : Cette classe représenté les états simples (les étapes de la vie du système).
c) **etat-initial** : Cette classe représenté l'état initial(l' initialisation du système).
d) **etat-composit** : Cette classe représenté les états composites (état regroupant un ensemble d'´états).
e) **etat-final** : Cette classe représente l'état final du diagramme d'états-transitions (fin de vie du système).
f) **Aspect-simple** : est une etat qui hérite tous les attributs de « etat – simple » et plus deux attributs « AD » et « PC » de type string. Graphiquement, elle est représentée par une petite rectangle rouge.
g) **Aspect-composit** : est une etat qui hérite tous les attributs de « etat−composit » et plus deux attributs « AD » et « P C » de type string. Graphiquement, elle est représentée par une rectangle rouge.

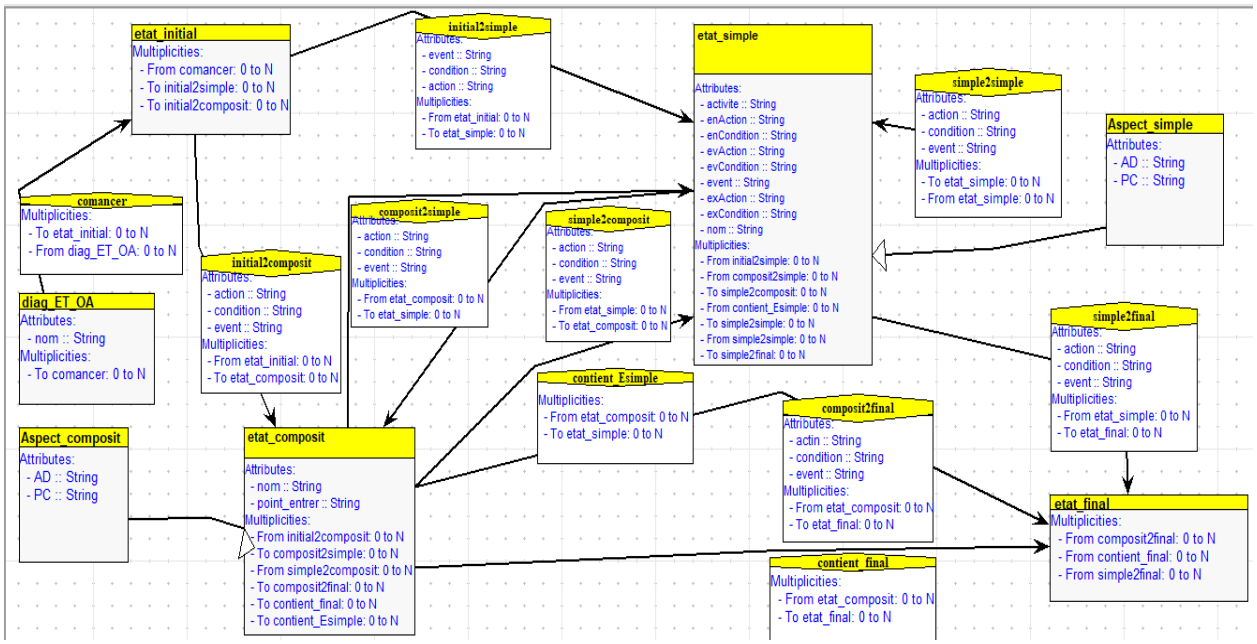


Figure 3 : Méta modèle pour le diagramme d'état-transition orienté aspect.

5.1.3. Méta-modèle de diagramme de communication orienté aspect

Le méta-modèle du diagramme de Communication orientée Aspect composé de trois classes : (DaigrammCommorienteAspect, OAspect, Objet) et deux Associations (contient , lien). Notre méta-modèle contient :

a) **DaigrammCommOrinteAspect** : Cette classe sert `a représenter le diagramme de communication, Graphiquement, elle est représentée par un grand rectangle noire.
b) **Objet :** Cette classe représenté les objets, chaque objet possède un nom objet et peut communiquer à travers des connecteurs.
c) **OAspect** : Cette classe représenté les objets ajoutés (les aspects) au DC et hérite tous ses attributs : multiplicités, associations à partir de la classe Objet.
d) Association :

- Contient : C'est une association de composition, qui permet de relier le DaigrammComm avec ses Objets.

- Lien : C'est une association simple, permettant la communication entre deux objets. possède des attributs de type String (condition ,nommsg,requrance,sequence) Autrement dit, elle représenté les messages envoyés ou reçus par un objet.

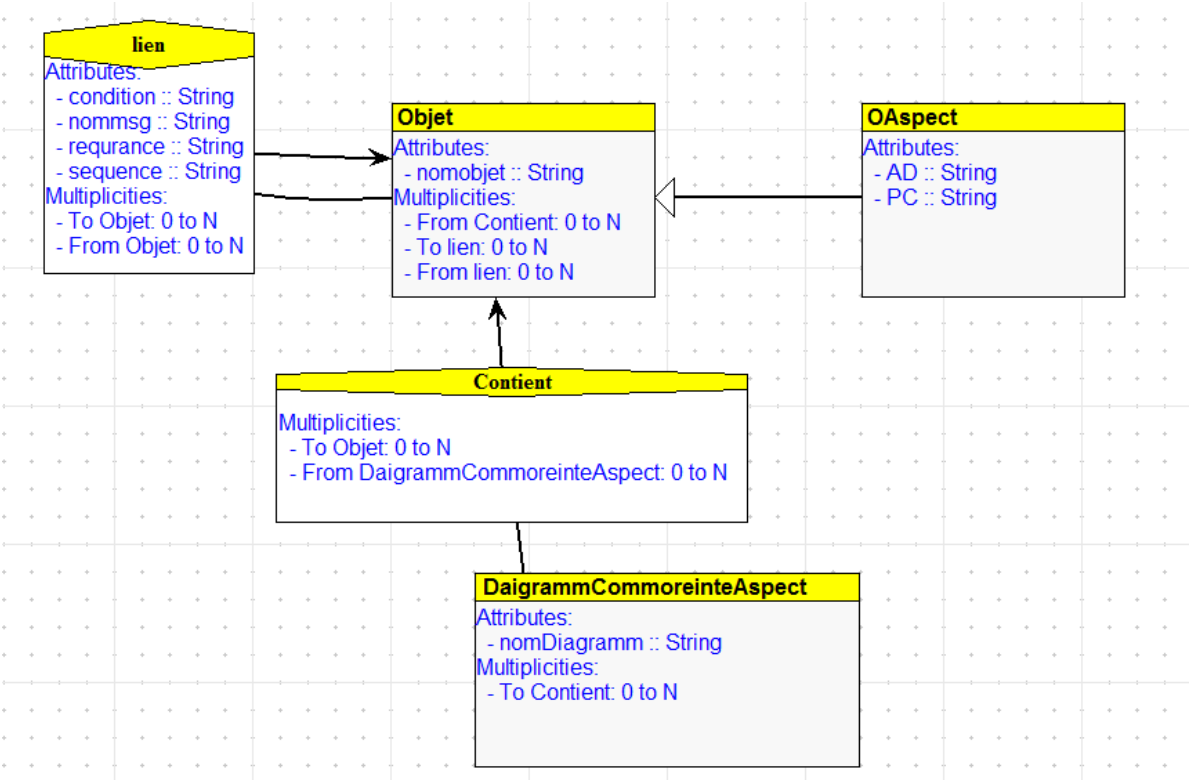


Figure 4 : Méta modèle pour le diagramme de communication orienté aspect.

5.2. Génération des Spécifications de Maude

Nous avons proposé une grammaire de graphes contenant 61 règles qui seront appliquées par ordre croissant (chaque règle à une priorité), aucune de ces règles ne changera les modèles UML car nous sommes concernés par la génération de code (Spécifications de Maude).

- Règles « 1, 2, 3,4 » : Ces règles sont appliquées pour localiser une classe qui n'a pas encore ´été traitée (Visité = = 0), et créer un fichier pour chacune.
- Règles « 5,6 » permettent de vérifier si une classe a sa propre State Machine pour ajouter les formats qui permettent de représenter les ´états simples, ´états composites, Aspect Simples et Aspect Composites, en fonction de la condition (nom_class) == (nom_stchart).nous avons déclaré une structure algébrique comme:

  sorts SIMSTATE COMSTATE STATE .
  subsort SIMSTATE < COMSTATE .
  subsort COMSTATE < STATE .
  op none : − > COMSTATE [ctor] .
  op_||_ : COMSTATE COMSTATE − > COMSTATE [ctor assoc comm id : none]

- Règles « 7,8 » permettent la récupération des informations associées aux classes (Simple/Aspect), que pour d´définir une classe, on peut utiliser la syntaxe suivante:

Class (nom_class) — Status : STATE, attr1 : (sort_attr1). . .attrn : (sort_attrn), (asso_name) : Oid.

- Règles « 9,10,11,12 » permettent la récupération des informations associées aux associations, et marquent l'association comme visitée (Asso.Visit = 1).
- Règles « 13,14,15,16,17,18 » permettent la récupération des informations associées aux associations .
- Règles « 19,20 » représentent la fin des Classes (Simple/Aspect).

➔ Ces règles sont appliquées pour sélectionner respectivement un état initial, un état final et un état (simple, composit, aspect) pour générer le code Maude correspondant.

- Règles « 21,22,23,24,25,26 »Ces règles permettent d'ajouter le code Maude appropriés des états initial, états final, états simple, états Composit, Aspect Simple, Aspect Composit.
- Règles « 27...39 » Ces règles permettent enregistre les évènements comme des messages, et le dernière règle (39) appliqué pour déclarer toutes les variables utilisées dans les règles de réécriture.
- Règles « 40...53 » Ces règles sont appliquées pour marquer la transition comme visitée et générer la spécification Maude correspondante.
- Règles « 54 , 55 » Ces règles sont appliquées pour marque la fin des modules orienté-objets/orienté-Aspect.
- Règle « 56 » Est appliqué pour localiser un diagramme de communication non précédemment traité (Vdiag == 0) et créer un nouveau fichier inclure tous les modules orientés Aspect.
- Règles « 57...60 » Ces règles sont appliquées pour sélectionner une Objet ou Objet Aspect (non traitée précédemment Comm = 0) et générer son code Maude équivalent.
- Règle « 61 » Est appliqué pour marquer la fin du module orienté Aspect qui est lié au diagramme de communication. Notre grammaire graphique a également une action finale qui efface toutes les variables globales.

6. Étude de cas

Pour illustrer notre approche, nous l'avons appliquée sur l'exemple du Processus de participation à une conférence. Nous proposons un chercheur et un Conférence: Chercheur est participer , Conférence Accepter la participation ou refusée , la figure présente des modèles UML qui représentent ce Processus. Pour traduire cette représentation graphique en son équivalent en code Maude dans notre framework, il suffit Commencez par exécuter le transformation qui permet d'exécuter notre grammaire des graphes définie dans la section précédente. Le résultat des fichiers générés automatiquement est illustré à la figure 6, à la figure 7 et à la figure 8 .

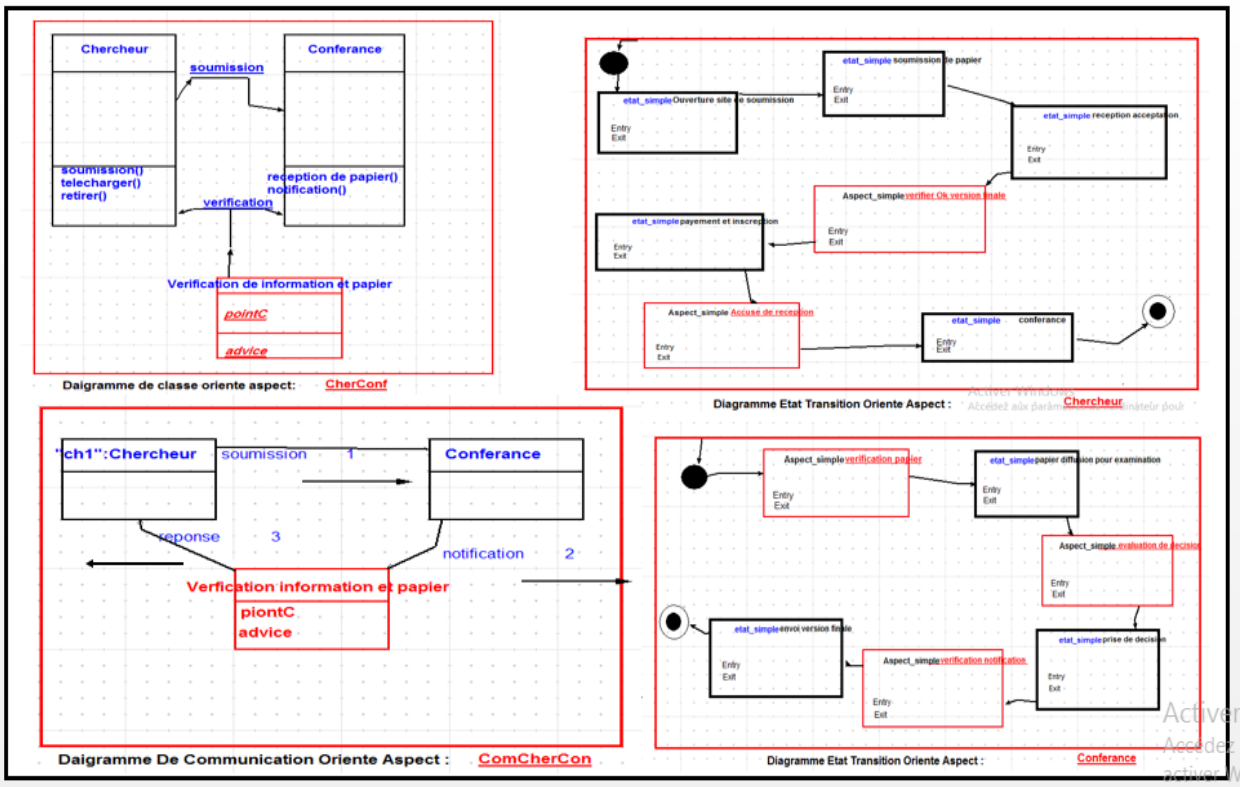


Figure 5 : Prossesus de participation à une conférence

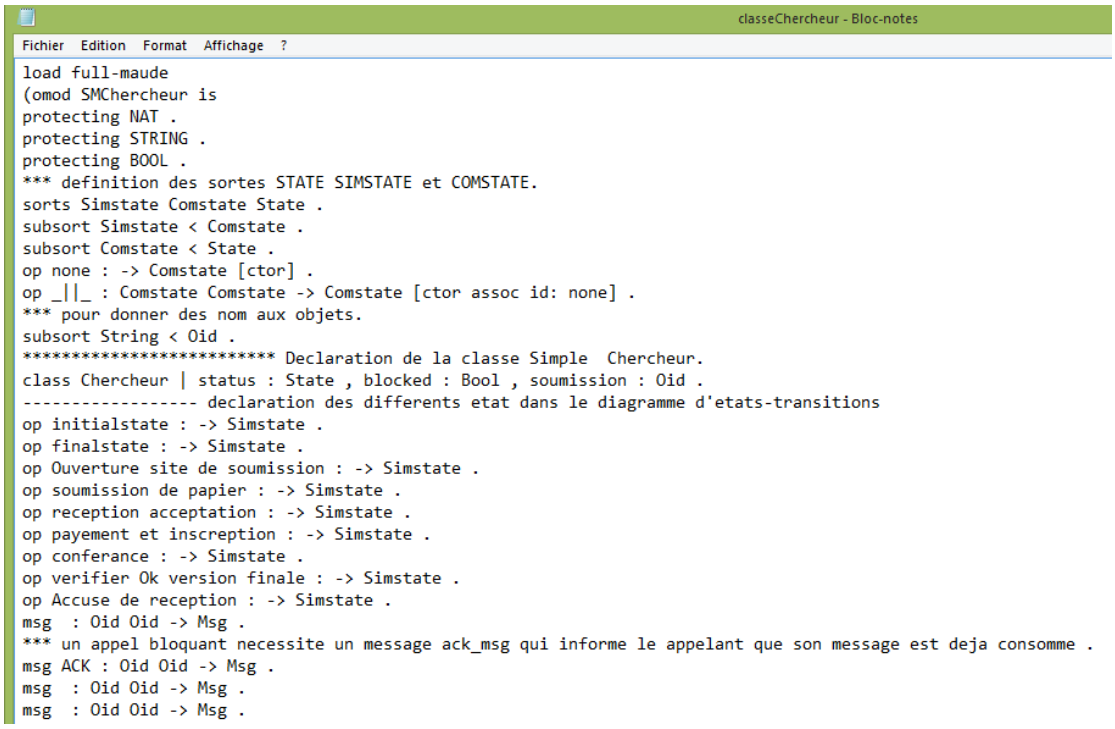


Figure 6 : Spécification Maude généré de Class Chercheur avec son propre état-transition

```
*** un appel bloquant necessite un message ack_msg qui informe le appelant que son message est deja consomme .
msg ACK : Oid Oid -> Msg .
***declaration des variables .
var CF : Configuration .
var Ch : String .
var c : String .
rl [1] : (c , Ch) < Ch : Chercheur | status : Ouverture site de soumission > CF => < Ch : Chercheur | status : soumission de papier > ACK(Ch , c) CF .
rl [1] : (c , Ch) < Ch : Chercheur | status : soumission de papier > CF => < Ch : Chercheur | status : reception acceptation > ACK(Ch , c) CF .
rl [1] : (c , Ch) < Ch : Chercheur | status : verifier Ok version finale > CF => < Ch : Chercheur | status : payement et inscreption > ACK(Ch , c) CF .
rl [1] : (c , Ch) < Ch : Chercheur | status : Accuse de reception > CF => < Ch : Chercheur | status : conferance > ACK(Ch , c) CF .
endom)
```

Figure 6.1 : Spécification Maude généré de Class Chercheur avec son propre état-transition 'Suit'

classeConferance - Bloc-notes

Fichier Edition Format Affichage ?

```
load full-maude
(omod SMConferance is
protecting NAT .
protecting STRING .
protecting BOOL .
*** definition des sortes STATE SIMSTATE et COMSTATE.
sorts Simstate Comstate State .
subsort Simstate < Comstate .
subsort Comstate < State .
op none : -> Comstate [ctor] .
op _||_ : Comstate Comstate -> Comstate [ctor assoc id: none] .
*** pour donner des nom aux objets.
subsort String < Oid .
************************** Declaration de la classe Simple  Conferance.
class Conferance | status : State , blocked : Bool , soumission : Oid .
------------------ declaration des differents etat dans le diagramme d'etats-transitions
op initialstate : -> Simstate .
op finalstate : -> Simstate .
op papier diffusion pour examination : -> Simstate .
op prise de decision : -> Simstate .
op envoi version finale : -> Simstate .
op verification papier : -> Simstate .
op evaluation de decision : -> Simstate .
op verification notification  : -> Simstate .
msg  : Oid Oid -> Msg .
msg  : Oid Oid -> Msg .
```

Figure 7 : Spécification Maude généré de Class Conférence avec son propre état-transition

```
*** un appel bloquant necessite un message ack_msg qui informe le appelant que son message est deja consomme .
msg ACK : Oid Oid -> Msg .
***declaration des variables .
var CF : Configuration .
var Co : String .
var c : String .
***regles de reecriture qui representent les transitons dans le diagramme de etats transition .
rl [initial] : < Co : Conferance | status : initialstate > CF => < Co : Conferance | status : verification papier > CF .
rl [1] : (c , Co) < Co : Conferance | status : verification papier > CF => < Co : Conferance | status : papier diffusion pour examination > ACK(Co , c) CF .
rl [1] : (c , Co) < Co : Conferance | status : evaluation de decision > CF => < Co : Conferance | status : prise de decision > ACK(Co , c) CF .
rl [1] : (c , Co) < Co : Conferance | status : verification notification  > CF => < Co : Conferance | status : envoi version finale > ACK(Co , c) CF .
endom)
```

Figure 7.1 : Spécification Maude généré de Class Conférence avec son propre état-transition 'Suit'

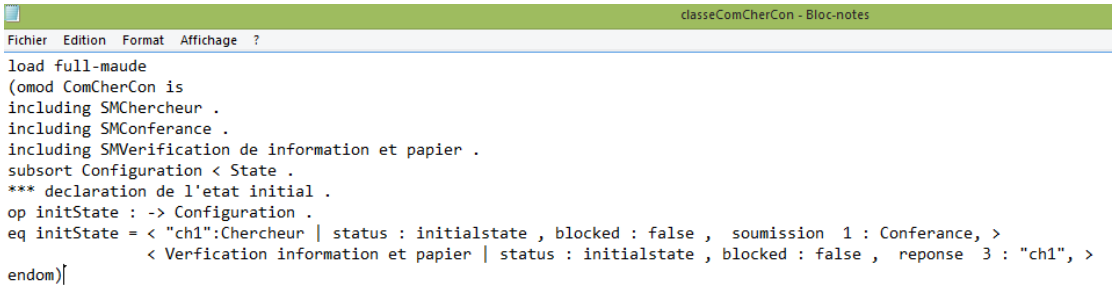


Figure 8 : Spécification Maude généré du processus de participation à une conférence

7. Conclusion

Dans cet article, nous avons proposé une approche et un outil de modélisation visuelle. Cette approche prend les applications modélisées en langage UML2.0 orienté Aspect et les traduit en un système de réécriture exprimé en langage Maude. Pour réaliser cette transformation, nous avons utilisé le formalisme des diagrammes de classes UML2.0 comme méta-formalisme et proposé trois méta-modèles pour les modèles d'entrée UML2.0, nous avons également proposé une grammaire des graphes pour générer du code Maude de manière graphique. L'outil de méta-modélisation AToM3 est utilisé. Dans un travail futur, nous prévoyons d'inclure la phase de vérification à l'aide du Maude LTL Model Checker et de donner un retour des résultats.

Les Références

[1] Cours « UML, le langage de modélisation d'objet unifié », en français http://uml.free.fr/index-cours.html

[2] UML, page web : http://laurent-audibert.developpez.com/Cours UML/html/CoursUML.html .

[3] Page Web UML : http://www.uml.org/

[4] Xavier, B., Isabelle, M., avec la contribution de Cédric, B : UML2 pour les développeurs. Cours avec exercices corrigés, éditeur :Eyrolles Paris (28/09/2006), ISBN : 2-212-12029-X.

[5] Gagnon, P., Mokhati, F., Badri, M. (2008). Application de la vérification de modèle aux modèles UML concurrents, Journal of Object Technology, 7 (1) 59– 84, janvier.

[6] Aouag,Mouna. (2014).Des diagrammes UML 2.0 vers les diagrammes orientés aspect à l'aide de transformation de graphes. Thèse de doctorat, Université de Mentouri, Constantine.pp :8-43

[7] Tibermacine, O. (2009). UML et Model Checking, Mémoire de Master dirigé par le Professeur A. Chaoui, Université El Hadj Lakhdar Batna, Algérie. (en français).

[8] Shama,Wafa. (2011). Une Approche basée transformation de graphes pour la génération de spécifications Maude à partir de diagrammes UML. Rapport de stage, Université de Mentouri Constantine, Constantine pp:7-9.

[9] Meseguer, J. (1992). Une théorie logique des objets concurrents et sa réalisation dans le langage Maude, G. Agha, P. Wegner et A. Yonezawa, éditeurs, Research Directions in Object-Based Concurrency. Presse du MIT, p. 314–390.

[10] Clavel, M., Duran, F., Eker, S., P., Lincoln, N. MartiOliet, Meseguer, J., Talcott, C. Manuel Maude (version 2.4). SRI International, http://maude.cs.uiuc.edu/ maude2-manual/maude-manual.pdf

[11] Kiczales, G. Lamping, J. Mendhekar, A. Maeda, C. Lopes, C. Loingtier, JM. Irwin,J : « Aspect Oriented Programming », publié dans les actes de la Conférence européenne sur la programmation orientée objet (ECOOP), Finlande. Springer-Verlag LNCS 1241. Juin 1997.

Formal Specification of Aspect Oriented Models, an approach based on Graph Transformation and Maude Language

Aouag Mouna, Dib Wiam, Saadaoui Amani.
Department of Mathematics and informatics .
University Centre Abd elhafid boussouf Mila
Algeria
aouag.mouna@centre-univ-mila.dz,
wiamd123@gmail.com ,
saadaouiamani36@gmail.com .

Abstract La Computer modelling is the most important step in software development. It facilitates the understanding of the functioning of a system before its realization by producing a model. There are several modeling methods such as Aspect Oriented Modeling (MOA) and Object Oriented Modeling (MOO).With the use of all the power of object-oriented languages, the object-oriented approach has shown these importance and efficiencies since these appearances in complex systems. But with the evolution of computing, the problems become more complex, many limitations of this approach have been found. In addition, aspect-oriented modelling has shown its usefulness in the design and development of complex systems, for which there are several models that are aspect-oriented, but the UML2.0 Aspect-oriented diagrams have no semantics. In addition, there are no tools to verify and validate these models. So, we proposed an approach to transforming Aspect oriented models towards formal models.

In this paper we proposed a transformation of aspect-oriented class diagrams, statechart and communication to the Maude language based on the graph transformation paradigm. Our approach consists in proposing Meta models (aspect-oriented models and Maude), a graph grammar and rules for the transformation between two different formalisms. Finally we will argue our proposal with well-illustrated case studies.

Keywords: Maude Language, Graph Grammar, AToM^3 , Rewriting Logic, Aspect Oriented Modeling, Graph Truncation.

1. Introduction

The UML unified modeling language is a standardized language compatible with object-oriented development methods.It has been used to specify and view software systems since 1980. It includes a large set of graphical notations called diagrams (thirteen diagrams) that represent the static (structural) and dynamic (behaviour) views of a system respectively. The class diagram describes a static view of the

system and models the relation between classes using associations, composition, inheritance, etc… [1], [2], [3], [4].

With the evolution in the field of computing, many problems become more complex. As a result, many limitations of Object Oriented Modeling have been found such as duplication, transversal concern, resolution and reuse of models.To do this, developers and programmers have proposed Aspect Oriented Modeling to exceed the limits mentioned above.

In this article, we propose an approach and a tool for automatic diagram transformation (class, state-transition, communication) oriented aspect in Maude language.

The rest of the article is organized as follows. Section 2 describes the most relevant related work and the differences between their multiple approaches. In section 3 briefly presents the rewriting system, Aspect oriented modeling and the Maude language. Section 4 details the proposed translation by defining the three meta-models of UML2.0 diagrams used (Class Diagram, State Chart and Communication Diagram) aspect-oriented and giving the rules of the proposed graph grammar, while section 6 describes a case study to illustrate our approach to translation. Finally, we provide a conclusion and some perspectives in Section 7.

2. Related Work

In [5], the authors presented some rules for matching UML diagrams to their equivalent Maude specifications. In [6] the authors added the concept of Aspect in three UML2.0 diagrams. The translation is done manually. In [7], the author presented another approach to transform UML diagrams into their equivalent Maude specifications. The translation is also done manually. In this article, we propose an automatic approach and a tool environment that formally transform aspect-oriented UML2.0 diagrams into their equivalent Maude specifications using the AToM3 meta-modeling tool and graph grammars.Our approach is based on the work presented in [8].

3. Rewritin Logic and Maude

The rewrite logic [9] was introduced by José Meseguer allowing the simultaneous specification and verification of software. It is implemented by several languages such as Maude [10].

Maude is a specification and programming language. It is simple, expressive and has an effective implementation.

Maude defines three types of modules: functional modules, system modules and object-oriented modules.

Functional modules allow defining data types and their properties by defining signatures and equations, but the dynamic behavior of a system is defined by the use of rewriting laws that we introduce in the System modules, these laws take the form (1). R: [t] [t'] if C (1)

This indicates that, according to the R rule, the term t is rewritten in t if a certain condition C is checked. Condition C is optional, so the rules can be unconditional. Finally, object-oriented modules add a more appropriate syntax to describe the object paradigm such as objects, messages and configurations.

Maude offers “full Maude” to accompany this , in addition, it has its own model checker that is used to verify the system properties.

4. Aspect -Oriented Modeling

Aspect Oriented Modeling is a new paradigm, developed by Kiczales.G and his team in 1996 [10]. It focuses on the concepts known as the Aspects. It allows the separation of a Base Model (BM) and an Aspectual Model (AM). Aspect Oriented Modeling is an integration of Aspect model in Base Model according to some defined points: Pointcut = $\sum$ Joinpoints and Advice.

Joinpoint (JP): Place in the model where the advice should be inserted.

Pointcut (PC): The whole of JP, often a regular expression and means the choice of JPs for the Advices application.

Advice (AD): part of the model that contains all or a part of aspect inserted at a joinpoint.

5. The Proposed Approach

The steps in our proposed approach are:

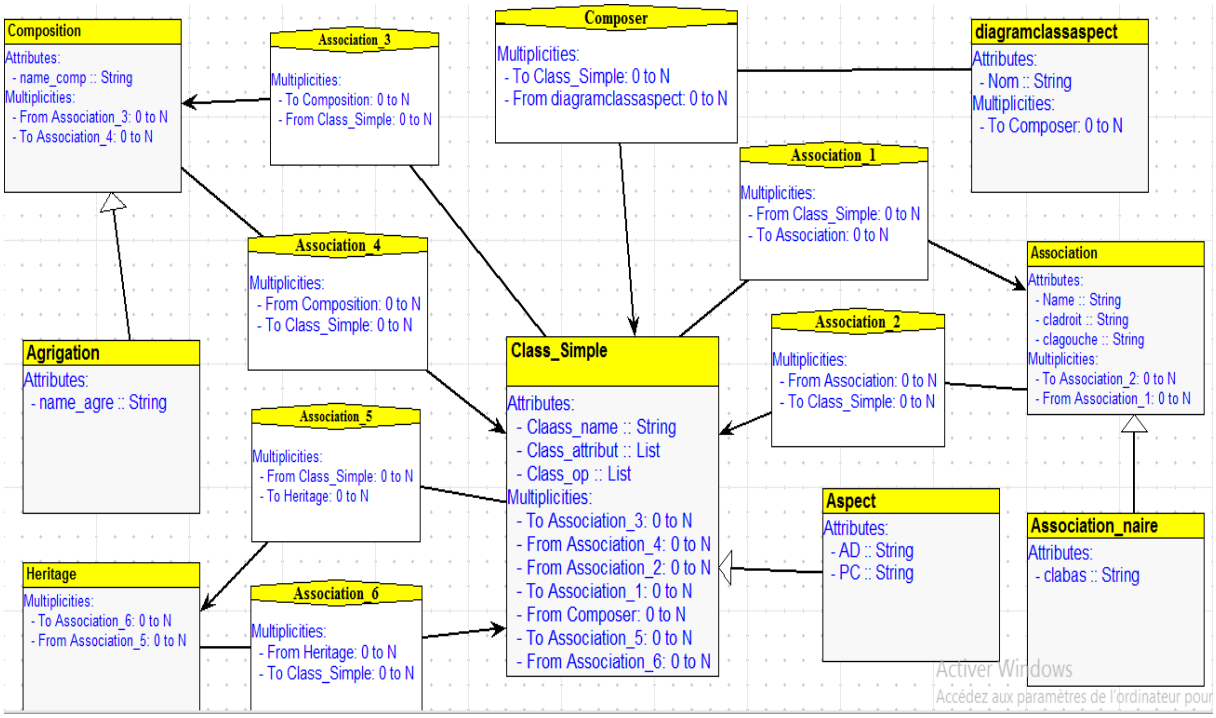
5.1. Meta-modelling of UML2.0 diagrams used

To translate UML2.0 diagrams into equivalent Maude specifications, we offer three meta-models: the first for the aspect-oriented class diagram, the second for the aspect-oriented state Machine diagram and the third for the aspect-oriented communication diagram. These meta models are represented by the UML2.0 Class Diagram formalism and the constraints are expressed in Python code.

5.1.1. Aspect-oriented class diagram meta-model

The meta-model of the class diagram oriented aspect consists of eight classes (diagramclassaspect, Class-simple, Composition, Agrigation, Heritage (inheritance), Association, Association-naire, Aspect) and seven Associations.

In the Figure 1, we present Meta model for aspect-oriented class diagrams



.**Figure 1 :** Meta model for aspect oriented class diagram

Our meta-model for the UML2.0 class diagram (see Figure 1) consists of the following classes:

a) **Diagramclassaspect :** this class represents the aspect-oriented class diagram. It has a "Name" attribute of type String. Graphically, it is represented by a large red rectangle.

b) **Class-simple :** this class represents the class-simple. It has three “Claas – name” attributes of type string, “Class – attribute” and “Class op” of type list. Graphically, it is represented by a black class.

c) **Aspect :** is a class that inherits the class "Class – Simple", it inherits all the attributes of the "Class – Simple" and more two attributes "AD" and "P C" of type string. Graphically, it is represented by a red class.

d) **Association:** this class represents associations between two classes. Graphically it is represented by a link, it has three attributes "Name" "cladroit" and "clagouche" of type string.

e) **Association-naire :** is a class that inherits the "Association" class, it inherits all the attributes of the "Association" class and more the "clabase" attribute of string type. This class represents associations between several classes. Graphically it is represented by a link.

f) **Composition:** this class represents the composition between two classes. It has the string type «name comp» attribute. Graphically it is represented by a plain diamond.

g) **Aggregation:** is a class that inherits the composition class, this class represents the aggregation between two classes. It has the string type «name agre» attribute. Graphically it is represented by an empty diamond.

h) **Heritage:** this class represents the inheritance between two classes. Graphically it is represented by a puddle.

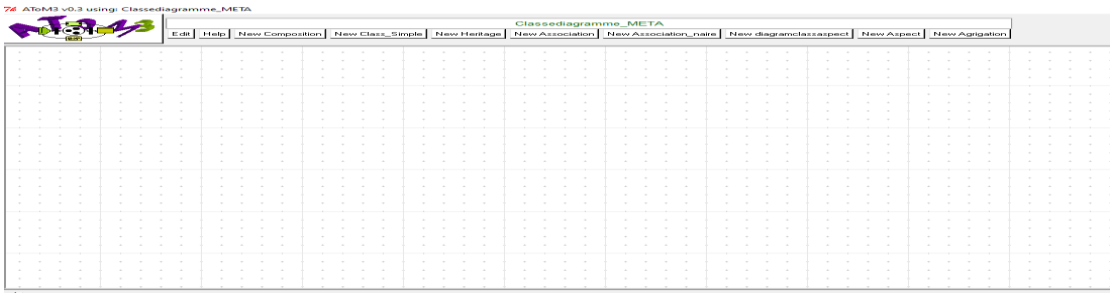


Figure 2 : The tool generated for aspect-oriented class diagrams.

5.1.2. Aspect-oriented stateChart diagram meta-model

The meta-model of the stateChart diagram oriented aspect composed of seven classes (initial-state, final-state, simple-state, composit-state, Aspect-simple, Aspect-composit, diag-ET-OA) and ten Associations. Our meta-model contains:

a) **diag-ET-OA :** This class has an attribute which is the Name, it represents the states in the diagram. Graphically, it is represented by a large red rectangle.
b) **simple-state :** This class represented the simple-state (life stages of the system).
c) **initial-state :** This class represents the initial state(the system initialization).
d) **composit-state :** This class represented composite states (state grouping a set of states).
e) **end-state :** This class represents the end-state of the stateChart diagram (end-of-life of the system).
f) **Aspect-simple :** is a state that inherits all the attributes of "simple state" and more two attributes "AD" and "PC" of type string. Graphically, it is represented by a small red rectangle.
g) **Aspect-composit :** is a state that inherits all the attributes of "composit state" and more two attributes "AD" and "P C" of type string. Graphically, it is represented by a red rectangle.

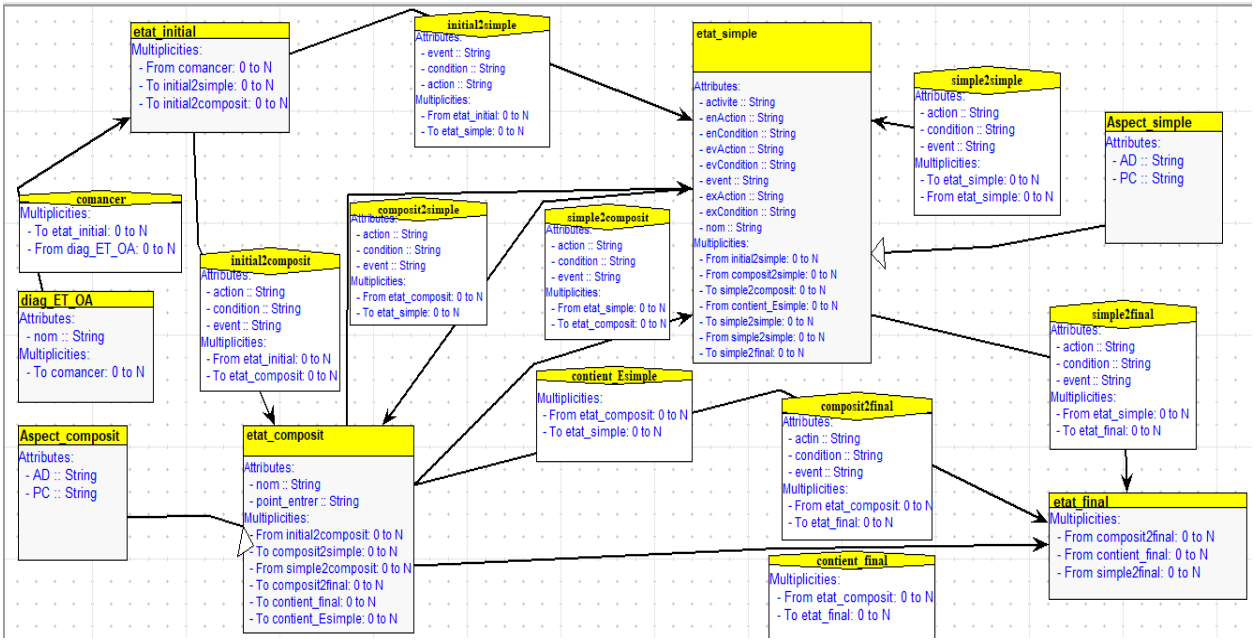


Figure 3 : Meta-model for the aspect-oriented stateChart diagram..

5.1.3. Aspect-oriented communication diagram meta-model

The Aspect-oriented Communication meta-model consists of three classes: (DaigrammCommorienteAspect, OAspect, Objet) and two Associations (contains , link). Our meta-model contains:

a) **DaigrammCommOrinteAspect** : This class is used to represent the communication diagram, Graphically, it is represented by a large black rectangle.
b) **Object :** This class represented objects, each object has an object name and can communicate through connectors.
c) **OAspect :** This class represents the objects added (aspects) to the DC and inherits all its attributes: multiplicities, associations from the Object class.
d) Association :

- Contains: It is a composition association, which allows to link the DaigrammComm with its Objects.

- Link: It is a simple association, allowing communication between two objects. has String attributes (condition ,nommsg,requrance,sequence) In other words, it represents the messages sent or received by an object.

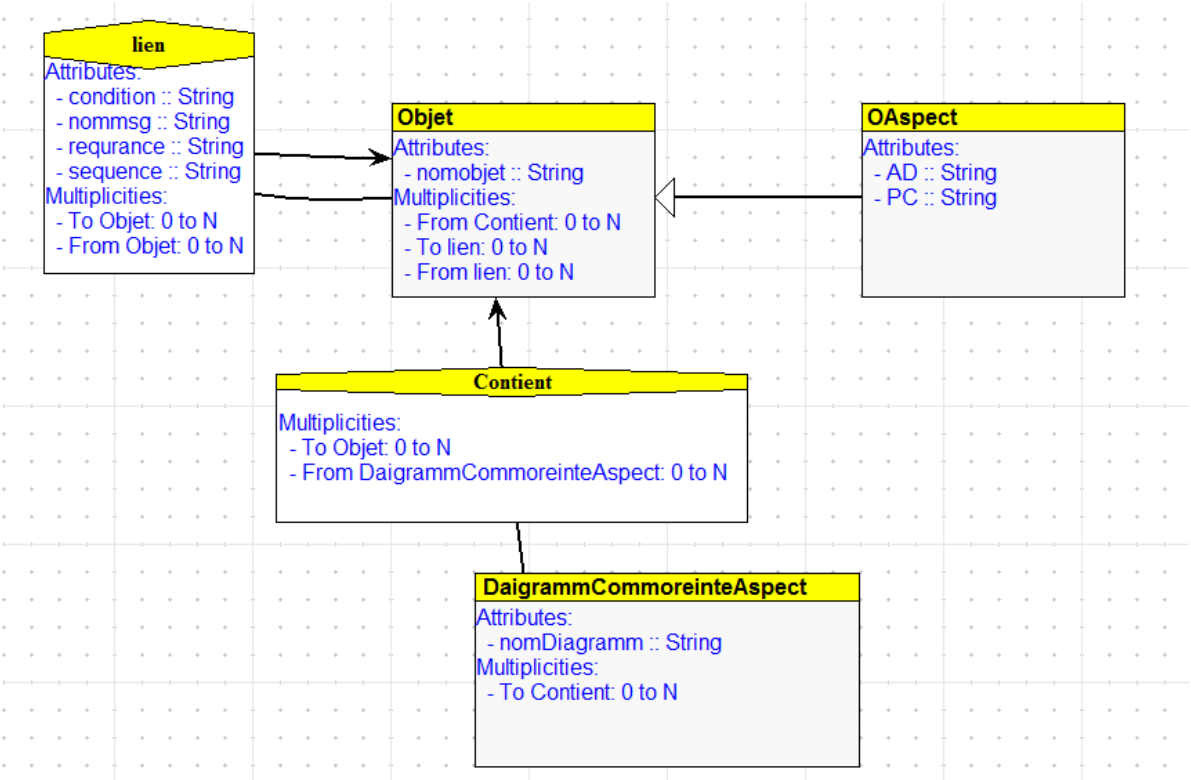


Figure 4 : Meta-model for the aspect-oriented communication diagram..

5.2. Generating Maude Specifications

We proposed a grammar of graphs containing 61 rules that will be applied in ascending order (each rule at a priority), none of these rules will change the UML models because we are concerned with code generation (Maude Specifications).

- Rules «1, 2, 3.4» : These rules are applied to locate a class that has not yet been processed (Visited = = 0) and create a file for each.
- Rules «5.6» : verify that a class has its own State Machine to add formats that represent simple states, composite states, Single Appearance and Composite Appearance, depending on the condition (filename) == (filename). we reported an algebraic structure like:

  sorts SIMSTATE COMSTATE STATE .
  subsort SIMSTATE < COMSTATE .
  subsort COMSTATE < STATE .
  op none : − > COMSTATE [ctor] .
  op_||_ : COMSTATE COMSTATE − > COMSTATE [ctor assoc comm id : none]

- Rules «7.8» : allow the retrieval of information associated with classes (Simple/Aspect), which to define a class, we can use the following syntax:

Class (nom_class) — Status : STATE, attr1 : (sort_attr1). . .attrn : (sort_attrn), (asso_name) : Oid.

- Rules «9,10,11,12» : allow the retrieval of information associated with associations, and mark the association as visited (Asso.Visit = 1)
- Rules «13,14,15,16,17,18» allow retrieval of information associated with associations.
- Rules «19,20» : represent the end of the Classes (Simple/Aspect).

➔ These rules are applied to select an initial state, an end state and a state (simple, composit, aspect) respectively to generate the corresponding Maude code.

- Rules «21,22,23,24,25,26» : These rules add the appropriate Maude code of initial states, end states, single states, Composit, Simple Appearance, Composit Appearance.
- Rules «27...39 » : These rules allow events to be recorded as messages, and the last rule (39) applied to declare all variables used in rewrite rules.
- Rules «40...53» : These rules are applied to mark the transition as visited and generate the corresponding Maude specification.
- Rules «54,55» : These rules are applied to mark the end of object-oriented/aspect-oriented modules.
- Rule «56» : is applied to locate a communication diagram not previously processed (Vdiag == 0) and create a new file to include all Aspect-oriented modules.
- Rules «57...60» : These rules are applied to select an Object or Aspect Object (not previously processed Comm = 0) and generate its equivalent Maude code.
- Rule «61» : is applied to mark the end of the Aspect-oriented module that is linked to the communication diagram. Our graphical grammar also has a final action that erases all global variables.

6. Case Study

To illustrate our approach, we applied it to the Conference Participation Process. We propose a Researcher and a Conference: Researcher is participating , Conference Accept participation or declined , figure 5 presents UML models that represent this Process. To translate this graphic representation into its equivalent in Maude code in our framework, simply Start by executing the transformation that allows us to execute our graph grammar defined in the previous section. The output of automatically generated files is shown in Figure 6, Figure 7 and Figure 8.

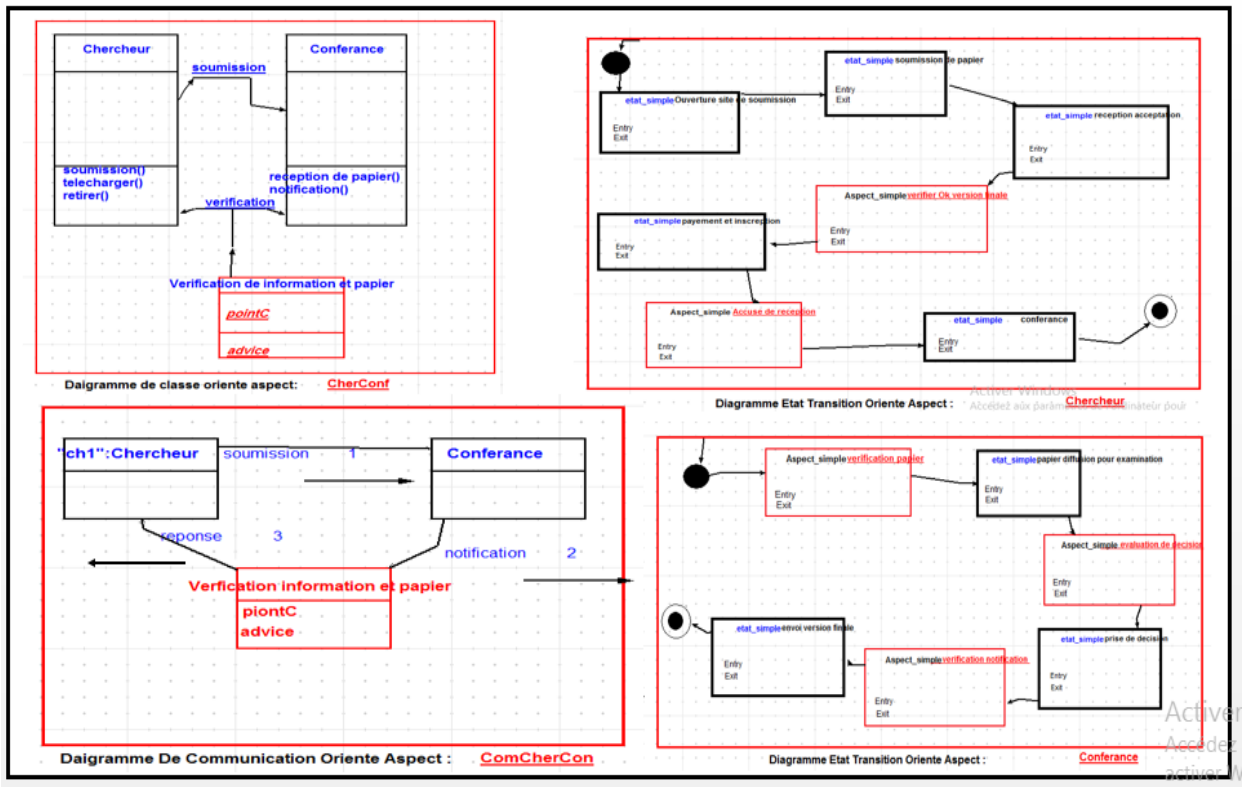


Figure 5 : Process of participation in a conference.

classeChercheur - Bloc-notes

Fichier Edition Format Affichage ?

```
load full-maude
(omod SMChercheur is
protecting NAT .
protecting STRING .
protecting BOOL .
*** definition des sortes STATE SIMSTATE et COMSTATE.
sorts Simstate Comstate State .
subsort Simstate < Comstate .
subsort Comstate < State .
op none : -> Comstate [ctor] .
op _||_ : Comstate Comstate -> Comstate [ctor assoc id: none] .
*** pour donner des nom aux objets.
subsort String < Oid .
************************** Declaration de la classe Simple  Chercheur.
class Chercheur | status : State , blocked : Bool , soumission : Oid .
------------------ declaration des differents etat dans le diagramme d'etats-transitions
op initialstate : -> Simstate .
op finalstate : -> Simstate .
op Ouverture site de soumission : -> Simstate .
op soumission de papier : -> Simstate .
op reception acceptation : -> Simstate .
op payement et inscreption : -> Simstate .
op conferance : -> Simstate .
op verifier Ok version finale : -> Simstate .
op Accuse de reception : -> Simstate .
msg  : Oid Oid -> Msg .
*** un appel bloquant necessite un message ack_msg qui informe le appelant que son message est deja consomme .
msg ACK : Oid Oid -> Msg .
msg  : Oid Oid -> Msg .
msg  : Oid Oid -> Msg .
```

Figure 6 : Maude specification generated from Class Researcher with its own stateChart

```
*** un appel bloquant necessite un message ack_msg qui informe le appelant que son message est deja consomme .
msg ACK : Oid Oid -> Msg .
***declaration des variables .
var CF : Configuration .
var Ch : String .
var c : String .
rl [1] : (c , Ch) < Ch : Chercheur | status : Ouverture site de soumission > CF => < Ch : Chercheur | status : soumission de papier > ACK(Ch , c) CF .
rl [1] : (c , Ch) < Ch : Chercheur | status : soumission de papier > CF => < Ch : Chercheur | status : reception acceptation > ACK(Ch , c) CF .
rl [1] : (c , Ch) < Ch : Chercheur | status : verifier Ok version finale > CF => < Ch : Chercheur | status : payement et inscreption > ACK(Ch , c) CF .
rl [1] : (c , Ch) < Ch : Chercheur | status : Accuse de reception > CF => < Ch : Chercheur | status : conferance > ACK(Ch , c) CF .
endom)
```

Figure 6.1 : Maude specification generated from Class Researcher with its own stateChart (followed)

classeConferance - Bloc-notes

Fichier Edition Format Affichage ?

```
load full-maude
(omod SMConferance is
protecting NAT .
protecting STRING .
protecting BOOL .
*** definition des sortes STATE SIMSTATE et COMSTATE.
sorts Simstate Comstate State .
subsort Simstate < Comstate .
subsort Comstate < State .
op none : -> Comstate [ctor] .
op _||_ : Comstate Comstate -> Comstate [ctor assoc id: none] .
*** pour donner des nom aux objets.
subsort String < Oid .
************************** Declaration de la classe Simple  Conferance.
class Conferance | status : State , blocked : Bool , soumission : Oid .
------------------ declaration des differents etat dans le diagramme d'etats-transitions
op initialstate : -> Simstate .
op finalstate : -> Simstate .
op papier diffusion pour examination : -> Simstate .
op prise de decision : -> Simstate .
op envoi version finale : -> Simstate .
op verification papier : -> Simstate .
op evaluation de decision : -> Simstate .
op verification notification  : -> Simstate .
msg  : Oid Oid -> Msg .
msg  : Oid Oid -> Msg .
```

Figure 7 : Maude specification generated from Class Conference with its own stateChart

```
*** un appel bloquant necessite un message ack_msg qui informe le appelant que son message est deja consomme .
msg ACK : Oid Oid -> Msg .
***declaration des variables .
var CF : Configuration .
var Co : String .
var c : String .
***regles de reecriture qui representent les transitons dans le diagramme de etats transition .
rl [initial] : < Co : Conferance | status : initialstate > CF => < Co : Conferance | status : verification papier > CF .
rl [1] : (c , Co) < Co : Conferance | status : verification papier > CF => < Co : Conferance | status : papier diffusion pour examination > ACK(Co , c) CF .
rl [1] : (c , Co) < Co : Conferance | status : evaluation de decision > CF => < Co : Conferance | status : prise de decision > ACK(Co , c) CF .
rl [1] : (c , Co) < Co : Conferance | status : verification notification  > CF => < Co : Conferance | status : envoi version finale > ACK(Co , c) CF .
endom)
```

Figure 7.1 : Maude specification generated from Class Conference with its own stateChart(followed)

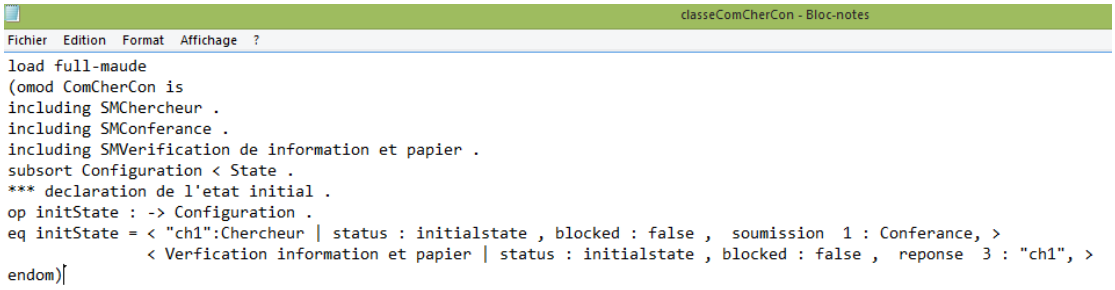


Figure 8 : Maude specification generated from the process of attending a conference

7. Conclusion

In this article, we have proposed a visual modeling approach and tool. This approach takes modeled applications in Aspect-oriented UML2.0 language and translates them into a rewrite system expressed in Maude language.
To achieve this transformation, we used the formalism of the UML2.0 class diagrams as meta-formalism and proposed three meta-models for the UML2.0 input models, we also proposed a grammar of the graphs to generate Maude code graphically.
The AToM3 meta-modelling tool is used. In future work, we plan to include the verification phsase using the Maude LTL Model Checker and provide feedback on the results.

References

[1] Cours « UML, le langage de modélisation d'objet unifié », en français http://uml.free.fr/index-cours.html

[2] UML, page web : http://laurent-audibert.developpez.com/Cours UML/html/CoursUML.html .

[3] Page Web UML : http://www.uml.org/

[4] Xavier, B., Isabelle, M., avec la contribution de Cédric, B : UML2 pour les développeurs. Cours avec exercices corrigés, éditeur :Eyrolles Paris (28/09/2006), ISBN : 2-212-12029-X.

[5] Gagnon, P., Mokhati, F., Badri, M. (2008). Application de la vérification de modèle aux modèles UML concurrents, Journal of Object Technology, 7 (1) 59– 84, janvier.

[6] Aouag,Mouna. (2014).Des diagrammes UML 2.0 vers les diagrammes orientés aspect à l'aide de transformation de graphes. Thèse de doctorat, Université de Mentouri, Constantine.pp :8-43

[7] Tibermacine, O. (2009). UML et Model Checking, Mémoire de Master dirigé par le Professeur A. Chaoui, Université El Hadj Lakhdar Batna, Algérie. (en français).

[8] Shama,Wafa. (2011). Une Approche basée transformation de graphes pour la génération de spécifications Maude à partir de diagrammes UML. Rapport de stage, Université de Mentouri Constantine, Constantine pp:7-9.

[9] Meseguer, J. (1992). Une théorie logique des objets concurrents et sa réalisation dans le langage Maude, G. Agha, P. Wegner et A. Yonezawa, éditeurs, Research Directions in Object-Based Concurrency. Presse du MIT, p. 314–390.

[10] Clavel, M., Duran, F., Eker, S., P., Lincoln, N. MartiOliet, Meseguer, J., Talcott, C. Manuel Maude (version 2.4). SRI International, http://maude.cs.uiuc.edu/ maude2-manual/maude-manual.pdf

[11] Kiczales, G. Lamping, J. Mendhekar, A. Maeda, C. Lopes, C. Loingtier, JM. Irwin,J : « Aspect Oriented Programming », publié dans les actes de la Conférence européenne sur la programmation orientée objet (ECOOP), Finlande. Springer-Verlag LNCS 1241. Juin 1997.