الجمهورية الجزائرية الديمقراطية الشعبية

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DR. YAHIA FARES DE MEDEA

FACULTE DES SCIENCES ET DE LA TECHNOLOGIE

Département du Génie Électrique et de l'Informatique



Projet de Fin d'Etudes

Présenté pour l'obtention du Diplôme
de Master en Génie Électrique

Spécialité : *Systèmes Intelligents et Communicants*

THEME

Modélisation et analyse des systèmes multi agents
Cas d'étude: Protocoles de communication

Proposé et dirigé par,

M. KHELDOUN Ahmed

Préparé et présenté par,

M. OULD ETHMANE Abdellahi

M. OULD MOHAMED MAHMOUD Brahim Salem

Dédicace

Je dédie ce modeste travail à mes chers parents

A la perle de ma vie et le parfum de mes jours ma très chère mère que

Dieu la protège et qu'elle a beaucoup sacrifié pour moi et qu'elle n'a

Jamais cessé de m'encourager

A l'âme de mon père qui m'a toujours soutenu

A mon cher ami, et mon binôme Brahim Salem

A mes frères, et ma sœur

A tous mes amis

A mes collègues de la section

SIC

OULD ETHMANE Abdellahi

Dédicace

Je dédie ce modeste travail à mes chers parents
A la perle de ma vie et le parfum de mes jours ma très chère mère que
Dieu la protège et qu'elle a beaucoup sacrifié pour moi et qu'elle n'a
Jamais cessé de m'encourager
A l'âme de mon père qui m'a toujours soutenu
A mon cher ami, et mon binôme Abdellahi/BELLAL
A mes frères, et mes sœurs
A tous mes amis
A mes collègues de la section SIC.

OULD MOHAMED MAHMOUD Brahim salem

Remerciements

Au terme de ce travail, nous tenons à remercier en premier lieu « Dieu » qui nous a donné la force pour achever ce travail.

Nous remercions vivement notre promoteur Mr .KHELDOUN

Nous tenons également à remercier tous ceux qui nous ont aidés à réaliser ce travail de près ou de loin.

Remerciements vont également à tous les membres de jury, qui font l'honneur de juger notre travail.

ملخص:

تتميز الأنظمة متعددة الوكلاء بسلوك معقد مرتبط بالتبادلات بين الكيانات المتجزئه التى توزع تبادل البيانات وتنـــسيق أنشطتها من أجل تحقيق الأهداف المشتركة.

في السنوات الأخـــيرة ركزت العديد من الدراسات والبحوث على مواصفات و تحليل الأنظمة متعددة الوكلاء.

تعتبر مخططات(AUML),تمد يد لمخططات(UML) لتمثيل الأنظمة متعددة الوكلاء من خلال أربعة محاور: وكيل، محيط، تنظيم و تبادل البيانات، مع ذالك يبقي (AUML) يفتقر إلي دلالات منهجية، وبالتالي فإنه لايمكن من التحقق من الخصائص المطلوبة في البروتوكولات التفاعلية.

الهدف من هذ العمـــــل هو استخدام طريقة منهجية لتحليل و تمثيل ابروتوكولات الاتصال في الأنظمة لمتعددة الوكلاء.

كلمات مفتاحية : الأنظمة المتعددة الوكلاء، بروتوكولات الاتصال، تبادل البيانات.

Résumé :

Les systèmes multi-agents (SMA) se caractérisent par un comportement complexe lié à l'interaction entre les entités réparties qui échangent des données et coordonnent leurs activités afin d'atteindre des objectifs communs. Ces dernières années, de nombreux efforts de recherche se sont focalisés sur la spécification et la vérification des SMA. Agent UML (AUML), a été révélée utile pour la spécification des systèmes multi-agents. AUML étend les diagrammes UML pour modéliser les SMA en quatre vues, l'agent, l'environnement, l'interaction et l'organisation. Cependant, AUML manque une sémantique formelle et, en conséquence, il ne permet pas de vérifier les propriétés requises de protocoles d'interaction. L'objectif de ce travail est d'utiliser une méthode formelle afin de vérifier/analyser les protocoles de communication dans les SMA.

Mots clés : SMA, Protocole de communication, Interaction, UML, AUML.

Abstract :

Multi-agent systems (MAS) are characterized by a complex behavior related to the interaction between the distributed entities which need to coordinate in order to reach a common objectives. These last years, numerous research efforts focused on the specification and verification of the MAS. Agent UML (AUML), was revealed useful for the specification of the multi-agent systems. AUML extends UML diagrams in order to model MAS in four sights, the agent, the environment, the interaction and the organization. However, AUML misses formal semantics, as a consequence, it does not allow to verify the properties required by communication protocols. The objective of present project is to use a formal method to verify/ analyze the communication protocols in MAS.

Keywords: MAS, Communication protocols, Interaction, UML, AUML.

Table des matières

LISTE DES FIGURES ..... **I**

LISTE DES TABLEAUX ..... **II**

LISTE DES ABREVIATIONS ..... **III**

INTRODUCTION GENERALE **1**

CHAPITRE I : GENERALITE SUR LES SYSTEMES MULTI-AGENTS **2**

1. Introduction 3

2. Notion d'agent 3

2.1. Définitions 3
2.1.1. Définition 1 3
2.1.2. Définition 2 4
2.1.3. Définition 3 4
2.2. Caractéristiques des agents 5
2.3. Classification des agents 7
2.3.1. Les agents réactifs 7
2.3.2. Les agents cognitifs 7
2.3.3. Les agents hybrides 7

3. Système Multi-Agents 8

3.1. Définition 8
3.2. Caractéristiques des systèmes multi-agents 9
3.3. Champs d'application des systèmes multi-agents 9
3.4. Interaction dans un système Multi-agents 9
3.4.1. Coordination 10
3.4.2. Coopération 10
3.4.3. Négociation 10
3.5. Communication dans les systèmes Multi-agents 10
3.5.1. Modes de communication 11

4. Protocoles d'interaction dans des systèmes multi-agents 12

4.1. Protocoles de FIPA 13

5. Langages de communication 13

5.1. KQML 13
5.2. FIPA-ACL 14

6. Gestion de conversation 14

7. Méthodologies de développement des SMA 15

8. Conclusion 15

Chapitre II : GENERALITE SUR LES RESEAUX DE PETRI 16

1. Introduction 18

2. Concepts et définitions 18

2.1. Réseau de Petri 18
2.2. Représentation d'un RdP 18
2.2.1. Représentation graphique 18
2.2.2. Représentation matricielle 19
2.3. Marquage 20

3. Sémantique d'un réseau de Petri 20

3.1. Franchissabilité et franchissement 20
3.2. Graphe d'accessibilité 21

4. Propriétés des réseaux de Petri 22

4.1. Vivacité 22
4.1.1. Quasi-vivacité d'une transition 22
4.1.2. Vivacité d'une transition pour un réseau marqué 22
4.1.3. Vivacité d'un réseau 22
4.2. Blocage 22
4.3. Caractère borné d'un réseau 23
4.4. Conflit 23
4.4.1. Conflit structurel 23
4.4.2. Conflit effectif 24

5. Réseaux de Petri récursifs 24

5.1. Définition 24
5.2. Syntaxe 25
5.3. Marquage étendu 25
5.3.1. Définition 25
5.4. Transition franchissable 26
5.4.1. Définition 26
5.4.2. Franchissement d'une transition élémentaire 26
5.4.3. Franchissement d'une transition abstraite 27
5.4.4. Franchissement d'une étape de coupure 28

6. Les RECATNets 29

6.1. Définition 29
6.2. Exemple d'un RECATNet 30

7. Conclusion 31

CHAPITRE III : TRANSFORMATION D'AUML VERS RECATNETS 32

1. Introduction 34

2. Description d'AUML 34

3. Les Diagrammes AUML 35

3.1. Le diagramme de séquence AUML 35

4. Modélisation des protocoles d'interactions entre agents 37

4.1. Premier Niveau 37
4.2. Deuxième Niveau 38
4.3. Troisième Niveau 38

5. Transformation du diagramme AUML vers RECATNet 39

6. Application sur FIPA-Broking Protocol 43

7. Conclusion 47

CHAPITRE IV : VERIFICATION ET VALIDATION 48

1. Introduction 49

2. Schéma général de vérification 49

3. Logique de réécriture & Maude 50

3.1. Logique de réécriture 50
3.2. Maude 51

4. Sémantique des RECATNets en termes de la logique de réécriture 51

4.1. Spécification de la structure des états des RECATNets 52
4.2. Règles de réécriture associées à un RECATNet 55
4.2.1. Règles élémentaires 55
4.2.2. Règles abstraites 56
4.2.3. Règles de coupure 56

5. ETUDE DE CAS (FIPA-BROKING-PROTOCOL) 57

5.1. Analyse par graphe d'accessibilité 57
5.1. Analyse par Maude 58

6. Conclusion 65

CONCLUSION GENERALE 66

Références Bibliographiques 67

Annexe 70

LISTE DES FIGURES

Figure I.1. *Capacités de perception et d'action de l'agent dans un environnement.* 4
Figure I.2. *Architecture d'un SMA.* 8
Figure I.3. *Communication via un Tableau Noir.* 11
Figure I.4. *Communication par envoi de messages.* 12
Figure II.1.*Réseau de Petri.* 19
Figure II.2. *Représentation matricielle d'un RdP.* 19
Figure II.3. *Réseau de Petri marqué.* 20
Figure II.4. *Graphe d'accessibilité de réseau de Petri de la figure II.3.* 21
Figure II.5. *Exemple d'un RdP vivant.* 22
Figure II.6. *RdP borné.* 23
Figure II.7. *Transition avec / sans conflit structurel.* 23
Figure II.8. *Conflit effectif.* 24
Figure II.9. *Exemple d'un RdPR simple.* 25
Figure II.10. *Etat du réseau de Petri récursif : Arbre de marquage.* 26
Figure II.11. *Franchissement d'une transition élémentaire.* 27
Figure II.12. *Franchissement d'une transition abstraite.* 27
Figure II.13. *Franchissement d'une étape de coupure.* 28
Figure II.14. *Transition élémentaire* 30
Figure II.15. *Transition abstraite.* 30
Figure II.16. *Exemple d'un RECATNet.* 31
Figure III.1. *Différents branchements en AUML.* 36
Figure III.2. *Exemple d'échange de message dans AUML.* 36
Figure III.3. *Paramètres du package AUML.* 37
Figure III.4. *Notation du Template en AUML* 38
Figure III.5. *FIPA-Broking-Protocol: AUML.* 44
Figure III.6. *FIPA-Broking-Protocol: RECATNet.* *45*
Figure IV.1. *Schéma général de vérification* 49
Figure IV.2. *Forme générale du marquage étendu associé à un RECATNet.* 53
Figure IV.3. *Graphe d'accessibilité de FIPA-Broking-Protocol* 57
Figure IV.4. *Exécution de la commande 'search' selon l'hypothèse 1 du premier test* 59
Figure IV.5. *Partie du graphe d'accessibilité de FIPA-Broking générée par la commande 'show search graph' selon l'hypothèse 1 du premier test* 60
Figure IV.6. *Résultat de la commande 'search' selon l'hypothèse 2 du premier test* 61
Figure IV.7. *Partie du graphe d'accessibilité de FIPA-Broking générée par la commande 'show search graph' selon l'hypothèse 2 du premier test* 62
Figure IV.8. *Vérification par model-checher de système Maude.* 63
Figure IV.9. *Contre-exemple généré par le model-checker de système Maude.* 64

LISTE DES TABLEAUX

Tableau I.1. *Comparaison entre agent cognitif et agent réactif* 7
Tableau III.1. *Entité agent* 40
Tableau III.2. *Envoi/Réception (synchrone).* 40
Tableau III.3. *Représentation d'une condition en RECATNet* 41
Tableau III.4. *Appel d'un sous protocole* 42
Tableau III.5. *Transitions et leurs sémantiques* 46

LISTE DES ABREVIATIONS

AUML	Agent Unified Modeling Language
FIPA	Foundation for Intelligent and Physical Agent
FIPA-ACL	FIPA Agent Communication Langage
RECATNet	Recursive Extended Concurrent Algebraic Term Net
KQML	Knowledge Query and Manipulation Language
LTL	Linear Temporel Logic

Introduction Générale

INTRODUCTION GENERALE

Le domaine des SMA (Système Multi-Agents) peut être vu comme le confluent de plusieurs disciplines de recherche: l'intelligence artificielle pour les aspects décisionnels de l'agent et les systèmes distribués pour les interactions entre agents et enfin du génie logiciel pour l'approche de modélisation orienté-agent et la création de composants logiciels autonomes. Les SMA s'inspirent également d'études issues d'autres disciplines connexes notamment la sociologie, la psychologie sociale, les sciences cognitives et bien d'autres. Ces fondations interdisciplinaires expliquent la richesse des SMA par rapport aux approches antécédentes, tel que l'orienté objet et les systèmes experts.

L'approche multi-agents est justifiée par : la coopération; la résolution de problèmes complexes; l'efficacité; la fiabilité; et la réutilisation. Elle repose sur un ensemble de concepts tels que la communication, l'organisation, la coopération, la coordination entre des entités plus ou moins intelligentes et autonomes. L'ensemble des concepts de l'approche multi-agents font de celle-ci une approche très riche et très puissante en termes de modélisation.

Agent UML (AUML), C'est révélée utile pour la spécification des systèmes multi-agents. AUML étend les différents diagrammes UML pour modéliser le SMA en quatre vues, l'agent, l'environnement, l'interaction et l'organisation. Cependant, AUML manque une sémantique formelle et, en conséquence, il ne permet pas de vérifier les propriétés requises de protocoles d'interaction.

Dans ce travail, nous nous sommes intéressé à l'utilisation d'une méthode formelle afin de vérifier/analyser les protocoles de communications dans les SMA décrits par le langage AUML. Par conséquent, Le présent mémoire est structuré en quatre chapitres de la manière suivante :

- Chapitre1 : dans ce chapitre nous présentons un ensemble de concepts sur l'agent et le SMA.
- Chapitre2: en ce qui concerne ce chapitre, nous exposons la théorie derière le formalisme de résaux de Petri et plus particuliérement les RECATNets.
- Chapitre3: ce chapitre décrit le langage AUML, et présente nos règles de transformation de modéle AUML vers RECATNet.
- Chapitre4: au cours de ce dernier chapitre, nous abordons la partie vérification et validation en utilisant le système Maude.

CHAPITRE I : GENERALITE SUR LES SYSTEMES MULTI-AGENTS

1. Introduction

Les systèmes multi-agents (SMA) constituent un domaine issu de la rencontre de l'intelligence artificielle (IA) et des systèmes distribués (SD). Ce paradigme a vu le jour pour remédier aux insuffisances et enrichir l'approche classique de l'IA afin de résoudre certains problèmes qui sont distribués de manière inhérente tels que par exemples la gestion du trafic aérien, la gestion des réseaux électriques, ... etc.

Dans ce chapitre, nous présentons d'abord la notion d'agent et les définitions que lui sont attribuées ensuite nous parlons de ses propriétés et ses types, puis on met l'accent sur le concept des SMA et leurs champs d'application ainsi que les interactions dans les SMA.
En fin on termine le chapitre par les interactions entre les SMA ainsi que la méthodologie de développement des SMA.

2. La notion d'agent

Dans cette section, nous exposerons tout d'abord les principales définitions du concept agent, ainsi que ses caractéristiques. Nous verrons ensuite ses différents types.

2.1. Définitions

Il existe dans la littérature plusieurs définitions du concept "Agent" [**23**] [**33**] [**7**] [**36**]. Elles ne sont pas contradictoires les unes avec les autres, elles diffèrent selon le type domaine pour lequel est conçu l'agent. Nous citons, les définitions les plus courantes:

2.1.1. Définition 1

Selon Russell & Norvig [**33**], un agent est une entité capable de percevoir son environnement par des capteurs et d'agir sur celui-ci grâce à des effecteurs. Cette idée est illustrée par la Figure-I.1.

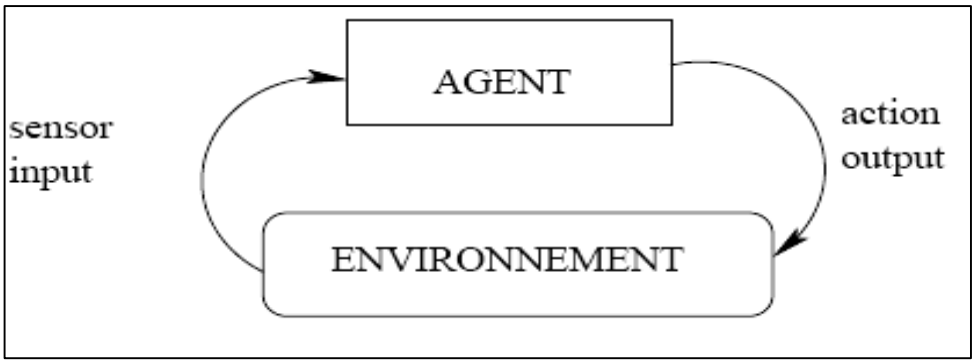


Figure I.1. *Capacités de perception et d'action de l'agent dans un environnement.*

Cette définition est encore générale, il n'y a aucune contrainte sur les actions de l'agent ni sur son comportement, il reste encore difficile de distinguer un agent des autres programmes ordinaires.

2.1.2. Définition 2

Selon Ferber [**13**], un agent est une entité autonome, physique[1]ou virtuelle[2], qui est capable d'agir sur son environnement sur lequel il dispose d'une représentation partielle, et qui peut communiquer avec d'autres agents, et éventuellement se reproduire et dont son comportement vise à la satisfaction de ses besoins et de ses objectifs à partir de tous les autres éléments (Perceptions, représentations, actions, communications et ressources) dont il dispose.

2.1.3. Définition 3

Selon Jennings & Wooldridge [**23**][**24**], un agent est une entité informatique située dans un environnement, qui est capable d'agir de manière autonome et flexible de façon à remplir ses objectifs, c'est à dire ceux qui lui ont été assignés lors de sa conception. Après avoir présenté l'état de l'art, nous avons adopté la première définition.

[1]: Réelle, objet réel qui existe et agit dans le monde réel ; ce peut être un robot.
[2] : Abstraite, des entités qui n'existent pas physiquement, plongées dans un environnement modélisé ; un logiciel.

2.2. Caractéristiques des agents

À partir des différentes définitions du concept "**Agent** ", on peut extraire les propriétés Clés d'un agent : l'agent doit être autonome, flexible, situé dans un environnement il perçoit, il agit et il communique. Nous essayons maintenant de clarifier ces notions :

- Autonomie

Il est difficile de définir la notion d'autonomie d'une manière précise, [**23**] lui accorde le sens suivant : Le système peut agir sans une intervention directe d'humains ou d'autres agents, et que le système possède le contrôle total sur ses propres actions et sur son état interne. On pourrait dire ainsi que le moteur d'un agent, c'est lui-même. Il a toujours la possibilité de répondre par l'affirmative ou le refus à des requêtes provenant des autres agents. L'agent donc n'est pas guidé par l'extérieur mais par ses tendances.

- Flexibilité

L'agent doit également être flexible, la flexibilité peut être vue comme une forme de l'intelligence. Etre flexible signifie que l'agent est [**23**]:

- ❖ Réactif

L'agent doit être capable de percevoir son l'environnement et de répondre à temps aux changements qui peuvent affecter cet environnement.

- ❖ Proactif

L'agent n'agit pas seulement d'une manière réactive (en fonction de son environnement) mais il doit avoir un comportement orienté objectifs et que l'agent peut prendre des initiatives.

- ❖ Social

L'agent est capable d'interagir avec les autres agents intelligents et humains pour qu'il puisse atteindre ses propres objectifs et aider les autres dans leurs activités.

- Perception

La perception est un moyen de recevoir de l'information d'un environnement qui est souvent complexe et évolutif. L'agent doit percevoir directement son environnement par ses propres capteurs.

- Action

La définition du mot action, dépend souvent du domaine de système conçu. L'agent est capables d'agir, et non pas seulement de raisonner comme dans les systèmes d'IA classique. L'action, repose sur le fait que les agents accomplissent des actions qui vont modifier l'environnement des agents et donc leurs prises de décision futures [**13**].

- Communication

Un agent fait un acte délibéré de transfert d'informations vers un ou plusieurs autres agents. La communication entre agents est l'un des modes principaux d'interaction existant entre les agents. L'interaction ne se résume pas à la communication, elle a d'autres formes : la négociation, la coopération, la coordination. En plus des propriétés clés citées, l'agent peut avoir éventuellement d'autres propriétés, telles que par exemple :

- Rationalité

Un agent rationnel est un agent qui agit d'une manière lui permettant d'obtenir le plus de succès possible dans la réalisation des tâches qu'on lui a assignées. Il dispose de critères d'évaluation de ses actions. Il suit le principe suivant:
"Si un agent sait qu'une de ses actions lui permet d'atteindre un de ses buts, il la sélectionne".

- Mobilité **:** capacité de déplacement dans un environnement [**12**].
- Bienveillance**:** les agents ne doivent pas avoir des objectifs contradictoires, et chaque agent essaye toujours de faire ce qui lui a été demandé.
- Adaptation : un agent adaptatif est capable d'apprendre en fonction de son expérience passée et de son évolution.
- Sécurité **:** un agent doit se protéger contre les attaques extérieures par rapport aux Autres Agents, à ses environnements d'exécution et pendant son déplacement sur le réseau pour les agents mobiles [**12**].

2.3. Classification des agents

Les agents sont généralement classés selon deux courants de penser : l'école cognitive et l'école réactive.

2.3.1. Les agents réactifs

Ce sont des agents qualifiés de non intelligents, ils répondent d'une façon opportune aux modifications de leurs environnements résultants des stimuli externes, les agents réactifs agissent en fonction de ces dernières sans nécessiter de compréhension de leurs univers ni de leurs buts [**6**].

2.3.2. Les agents cognitifs

L'agent dispose d'une capacité de raisonnement, il possède des informations et des savoir-faire nécessaires à la réalisation de sa tâche et à la gestion des interactions avec son environnement et les autres agents. L'agent possède un but à atteindre à l'aide des plans explicites. Les différences entre agent cognitif et agent réactif.

Tableau I.1. *Comparaison entre agent cognitif et agent réactif*

Système d'agents cognitifs	**Système d'agents réactifs**
Représentation d'agents cognitifs	Pas de représentation explicite
Peut tenir compte de son passé	Pas de mémoire de son historique
Agent complexe	Fonctionnement stimulus /réaction
Petit nombre d'agent	Grand nombre d'agent

2.3.3. Les agents hybrides

Chacune des approches **réactive** et **cognitive** est plus ou moins appropriée selon le type de problème. Pour la plupart des problèmes, il est souhaitable de concevoir des systèmes hétérogènes comprenant les deux types d'agents, ou d'avoir des agents conciliants à la fois des aspects réactifs et des aspects cognitifs : on parlera dans ce cas d'agent hybride [**35**]. Différentes approches ont été proposées pour combiner les deux comportements.

3. Système Multi-Agents

3.1. Définition

Les systèmes Multi-Agents (SMA) sont des systèmes distribués conçus et implantés idéalement comme un ensemble d'agents interagissant, le plus souvent, selon des modes de coopération, de concurrence et de coexistence.

Jacques Ferber définit les systèmes Multi-Agents comme des systèmes composés des éléments suivants [**13**]:

- ✓ Un environnement **E**, c'est-à-dire un espace disposant généralement d'une métrique.
- ✓ Un ensemble d'objets **O** situé dans **E.** Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.
- ✓ Un ensemble **A** d'agents, qui représentent les entités actives du système.
- ✓ Un ensemble de relations **R** qui unissent des objets (et donc des agents) entre eux.
- ✓ Un ensemble d'opérations **Op** permettant aux agents de **A** de percevoir, produire, consommer, transformer et manipuler des objets de **O**.
- ✓ Des opérateurs chargés de représenter l'application de ces opérations et la réaction de l'environnement envers les tentatives de modification.

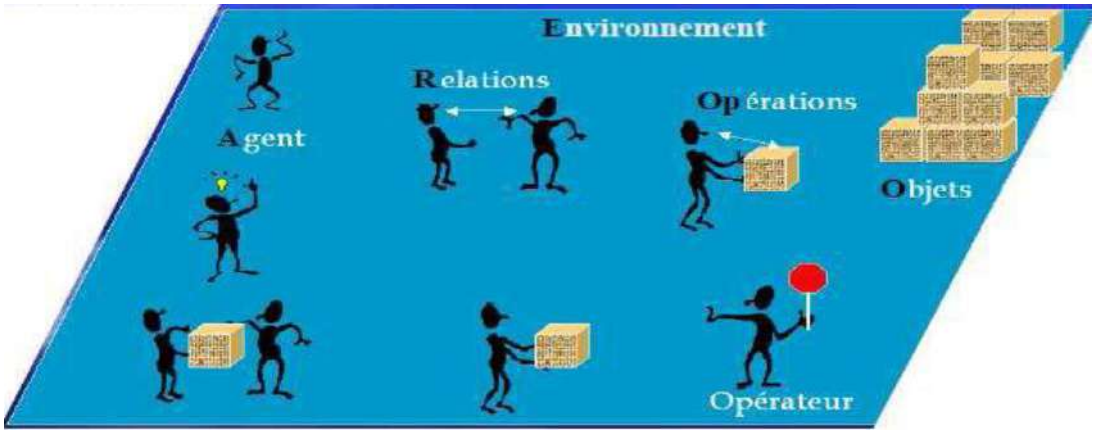


Figure I.2. *Représentation d'un SMA.*

3.2. Caractéristiques des systèmes multi-agents

Un système Multi-agents possède généralement les caractéristiques suivantes:

- Chaque agent travaille dans le but d'accomplir ses tâches.
- Chaque agent est capable de communiquer et d'interagir avec d'autres agents.
- Un agent coopère avec les autres agents lorsque nécessaire.
- Un agent est capable de coordonner ses activités avec les autres agents pour accéder à des ressources et à des services partagés dont il a besoin (pour réaliser ses buts).
- Les agents ont un but commun.
- Chaque agent a une vue partielle du SMA [**3**].

3.3. Champs d'application des systèmes multi-agents

Les domaines d'application de la technologie d'agents sont très vastes, des exemples d'applications sont :

- Assistance du commerce électronique : les agents sont utilisés pour les négociations modérées, en comparant des propositions, etc.
- Simulation des systèmes décentralisés: agents comme un outil de modélisation (Biologie, règlement de la circulation etc.)
- Les Robots : les agents sont personnifiés dans les robots et réagissent réciproquement dans le vrai monde.

3.4. Interaction dans un système Multi-agents

Une interaction « est une mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques. Les interactions s'expriment ainsi à partir d'une série d'actions, dans les conséquences exercent en retour une influence sur le comportement futur des agents » [**13**]**.**

Nous allons donner par la suite quelques définitions de la coopération, la coordination et la négociation, ces types d'interactions étant parfois difficiles à différencier.

3.4.1. La coordination

La coordination est une propriété de système Multi-agent qui exécute des activités dans un environnement partage.

Le degré de coordination est une mesure par laquelle on évitait des activités étranges par la réduction de réclamation de ressources, éviter le blocage et de maintenir le système sain.

3.4.2. La coopération

La coopération est une caractéristique très importante des SMA. En effet, une résolution distribuée d'un problème ou le partage des tachés et de résultats est les résultats d'interactions coopératives entre les différents agents qui disposent d'un ensemble des caractéristiques qui leur rendent coopératifs.

Ferber a donné une définition plus précise de la coopération : « On dira que deux agents coopèrent, ou encore qu'ils sont en situation de coopération, si l'une des deux conditions est vérifiée » :

- L'ajout d'un nouvel agent permet d'accroître d'une manière différentielle les performances du système.
- L'action des agents sert à éviter ou à résoudre des conflits potentiels ou actuels. c'est surtout le dernier point qui rend la négociation une interaction coopérative dans le cas où elle interviendrait pour résoudre les conflits.

3.4.3. La négociation

La négociation définie comme le processus d'améliorer les accords (en réduisent les inconsistances et l'incertitude) sur des points communs [**17**].

3.5. Communication dans les systèmes Multi-agents

La communication représente la base de réalisation de tous les modes d'interaction qu'on a vue précédemment, soit la négociation ou bien la coordination.

Une communication dans un SMA peut être définie comme une forme d'action locale d'un agent vers d'autres agents.

3.5.1. Modes de communication

On peut distinguer deux modes de communication ; la communication directe et la communication indirecte.

- Communication indirecte : dans ce mode de communication les agents communiquent par signaux via l'environnement. Ces signaux, une fois interprétés, vont produire des effets sur les agents**.** Ce type de communication est spécifique aux agents réactifs.
- Communication directe: l'échange direct est réalisé volontairement en direction d'un individu ou groupe d'individus soit :

• Par partage d'informations: les agents communiquent en utilisant la technique du tableau noir[3]ou « *Blackboard »* [**32**] [**33**] [**8**]. Dans cette technique, on suppose l'existence d'un espace où les agents peuvent partager et échanger les informations relatives à la résolution de problème.

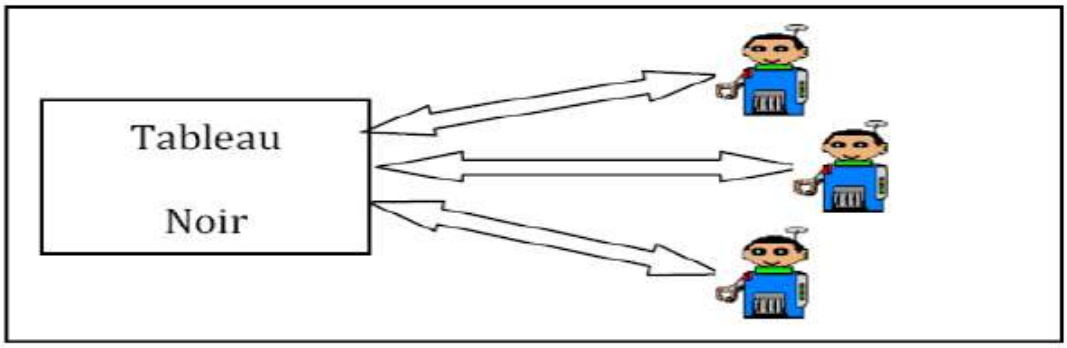


Figure I.3. *Communication via un Tableau Noir.*

• Par envoi de messages : la communication se fait par échange de messages entre Les agents, ce mode de communication provient initialement du domaine des acteurs [**19**]. Cette approche s'est inspirée de l'interaction sociale telle que nous la trouvons dans d'autres contextes comme la communication entre les humains.

[3] : Ce type de communication n'existe pas dans les SMA où l'on dispose que d'une vision partielle de systèmes alors que la communication par partage d'information suppose l'existence d'une base partagée **sur** laquelle les agents viennent lire et écrire.

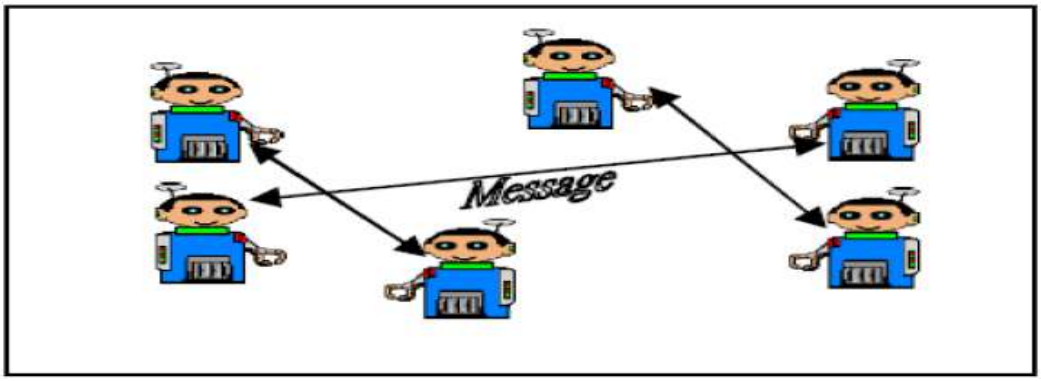


Figure I.4. *Communication par envoi de messages.*

4. Les protocoles d'interaction dans des systèmes multi-agents

L'interaction entre les agents est plus qu'un simple échange de messages un des aspects d'interaction montre une conversation basée sur un échange partagé et conventionné de messages. Les conversations entre agents dans les SMA sont souvent structurées selon des schémas typiques appelés protocoles d'interaction. Les protocoles d'interaction permettent de décrire explicitement les enchaînements conversationnels lors des communications entre les agents. Ils représentent un schéma commun de conversation utilisé pour exécuter une tâche, une stratégie de haut niveau gouvernant les interactions entre les agents tout en permettant de faciliter leur dialogue. Un protocole précise qui peut dire quoi à qui et les réactions possibles à ce qui est dit. Les protocoles d'interaction permettent de définir une séquence causale des messages communiqués entre les agents et décrivent comment les agents doivent réagir aux messages reçus durant les interactions. Les aspects de ces protocoles d'interaction diffèrent selon le type des agents. Il existe différents types de protocoles d'interaction, nous citons quelques-uns :

- Les protocoles de coordination
- Les protocoles de coopération
- Les protocoles de négociation

Les protocoles d'interaction peuvent être modélisés en utilisant différents formalismes tels que :

- ✓ les réseaux de Petri
- ✓ les langages formels
- ✓ les langages de description de protocoles

La définition d'une architecture logicielle générique permettant l'interopérabilité entre les agents dans des environnements ouverts et dynamiques semble intéressante pour l'exploitation des protocoles d'interaction. Plusieurs groupes de recherche ont déjà défini leurs propres modèles. Les modèles existants sont : les modèles de FIPA, General Magic, KAoS, OMG, ZEUS le modèle des laboratoires de British Telecom, et le modèle de l'université de Calgary. Nous utilisons dans ce travail de protocoles de FIPA.

4.1. Les protocoles de FIPA

FIPA spécifie un environnement d'existence et de fonctionnement des agents ainsi qu'une infrastructure physique de déploiement des agents. FIPA définie une plate-forme des agents qui représente l'infrastructure dans laquelle les agents peuvent être déployés, un système de gestion des agents décrit tel les agents peuvent gérer la création, la suppression, la suspension, la reprise, l'authentification et la migration des autres agents tout en assurant un service des pages blanches ainsi qu'un service des pages jaunes.

5. Langages de communication

L'intérêt des langages de communication est de faire l'échange et l'interprétation des messages et l'interopérabilité entre les agents. Ces langages se focalisent essentiellement sur la manière de décrire exhaustivement des actes de communication d'un point de vue syntaxique et sémantique. Deux langages de communication entre agent, FIPA-ACL et KQML [**14**] [**25**], ont émergé des efforts de standardisation de la communauté des SMA, portés par des besoins d'interopérabilité logicielle.

5.1. KQML

Le premier langage qui a été introduit est KQML (**Knowledge Query and Manipulation Language**). A l'origine, KQML a été développé pour échanger des informations et des connaissances entre des systèmes à base de connaissances. Il a été ensuite repris dans [**14**] pour décrire les messages échangés entre les agents.

5.2. FIPA-ACL

FIPA-ACL (FIPA Agent Communication Language), est proposé dans le cadre d'un travail de standardisation mené au sein l'organisation FIPA[4] *(Foundation of Intelligent Physical Agents).* Ce langage est fondé sur vingt et un actes communicatifs, exprimés par des performatifs, qui peuvent être groupés selon leurs fonctionnalités de la façon suivante

- Passage d'information : Inform, Inform-if, Inform-ref, Confirm, Disconfirm,
- Réquisition d'information : Query-if, Query-ref, Subsidie,
- Négociation : Accept-proposal, Cfp, Propose, Reject-proposal,
- Distribution de tâches (ou exécution d'une action) : Request, Request-when, Agree, Refuse, Request-whenever
- Manipulation des erreurs : Failure, Not-understood.

6. Gestion de conversation

Une conversation est une séquence d'actes de communication échangés entre deux ou plusieurs agents. La communauté SMA distingue essentiellement deux approches différentes pour gérer les conversations [**22**]:

- Dans la première approche, l'agent construit son propre plan de communication au moment où il en a besoin pour réaliser sa tâche. Ainsi, l'interaction n'est pas définie a priori, elle est émergente. Les connaissances et les buts des agents régissent l'interaction en spécifiant le message à envoyer (acte de communication, agents destinataires, le contenu du message,...etc.).

Cette approche donne à l'agent plus de flexibilité dans son interaction. Cependant, dans cette approche l'agent doit avoir suffisamment de connaissances sur la sémantique des messages et qu'il doit être doté d'un mode de raisonnement lui permettant de mener ses interactions.

- Dans la deuxième approche, l'interaction se conforme à un enchaînement de messages spécifiés à l'avance. Les règles qui régissent l'échange de messages forment le protocole d'interaction. De plus, le protocole d'interaction définit le type des messages qui doivent être échangés. L'agent interactif doit se conformer aux règles de conversation dictées dans le protocole. Comparée à la première approche, l'interaction basée sur les protocoles d'interaction ne nécessite pas d'architecture complexe au sein de l'agent.

[4] : http://www.fipa.org.

7. Définitions de la méthodologie de développement des SMA

La construction de SMA impose l'intégration des méthodes et techniques de différentes aires de connaissance: du génie logiciel pour structurer le processus de développement, de l'intelligence artificielle pour doter les programmes de capacité pour traiter des situations imprévues et prendre des décisions et de la programmation concurrente et distribuée pour traiter la coordination de tâches exécutées dans différentes machines. En effet, le développement de SMA se complique, c'est un défi en lui-même.

La recherche dans le domaine de développement des SMA inclut des contributions sur les architectures d'agents, sur des plateformes multi-agents, et sur des méthodologies multi-agents.

8. Conclusion

Dans ce chapitre, nous avons représenté les concepts agent et le système multi-agents, nous avons défini leurs caractéristiques, et les notions fondamentales de ce domaine. Il est intéressant de noter, que l'ensemble des concepts de l'approche multi-agents font de celle-ci une approche très riche et très puissante en termes de modélisation. Néanmoins, le manque crucial de côté formel en est le principal inconvénient. Pour y remédier, un axe de recherche est en pleine expansion, il s'agit d'associer à l'approche multi-agents une approche formelle. Pour plusieurs raisons, l'association de formalisme réseaux de Petri à l'approche SMA semble être un domaine d'investigation très prometteur.

Dans le prochain chapitre, nous allons présenter le formalisme de réseaux de Petri.

CHAPITRE II : GENERALITE SUR LES RESEAUX DE PETRI

1. Introduction

Un réseau de Petri (RdP) est un outil graphique et mathématique servant à représenter des systèmes à fonctionnement séquentiel dans plusieurs domaines (informatique, industriels,…). Cette théorie est apparue en 1962, dans la thèse de doctorat de l'allemand Carl Adam Petri (Communication avec automatismes) à l'université de Darmstadt [**34**].

2. Concepts et définitions

2.1. Réseau de Petri

Un réseau de Petri est un quadruplet

$$\mathcal{R} = (\mathbf{P}, \mathbf{T}, \mathbf{Pre}, \mathbf{Post}) \qquad \textbf{(II.1)}$$

Où :

- P est un ensemble fini de places.
- T est un ensemble fini de transitions.
- Pre et Post: sont deux fonctions partielles de P×T dans ℕ.

Pre : P×T→ ℕ est l'application places précédentes.

Post : P×T→ ℕ est l'application places suivantes.

2.2. La représentation d'un RdP

Un réseau de Petri peut être représenté par deux façons à savoir : représentation graphique et matricielle.

2.2.1. La représentation graphique

Selon cette représentation, un RdP est vue comme un graphe biparti (voir la figure II.1) où les places et les transitions sont relies entre aux (place →transition ou transition→ place) par des arcs orientés [**34**].

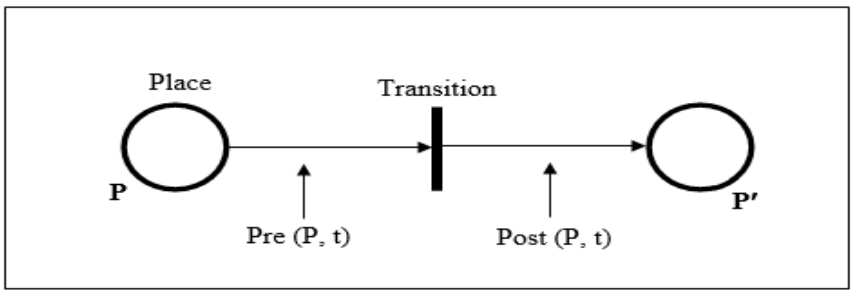


Figure II.1. *Réseau de Petri.*

2.2.2. La représentation matricielle

On utilise la notation : $\mathbf{C = Post - Pre}$ **(II.2)**

Où C'est en général appelé matrice d'incidence du réseau de Petri.

Les matrices Pre et Post ont la même dimension | P |x| T | où :

- | P | : représente le nombre de places de réseau de Petri,
- | T | : représente le nombre de transitions de réseau de Petri.

La matrice Pre contient les évaluations des arcs qui vont des places vers des transitions, et Post concerne les arcs des transitions vers les places [**34**].

On peut passer de la représentation graphique vers la représentation matricielle ou l'inverse comme illustre la figure II.2.

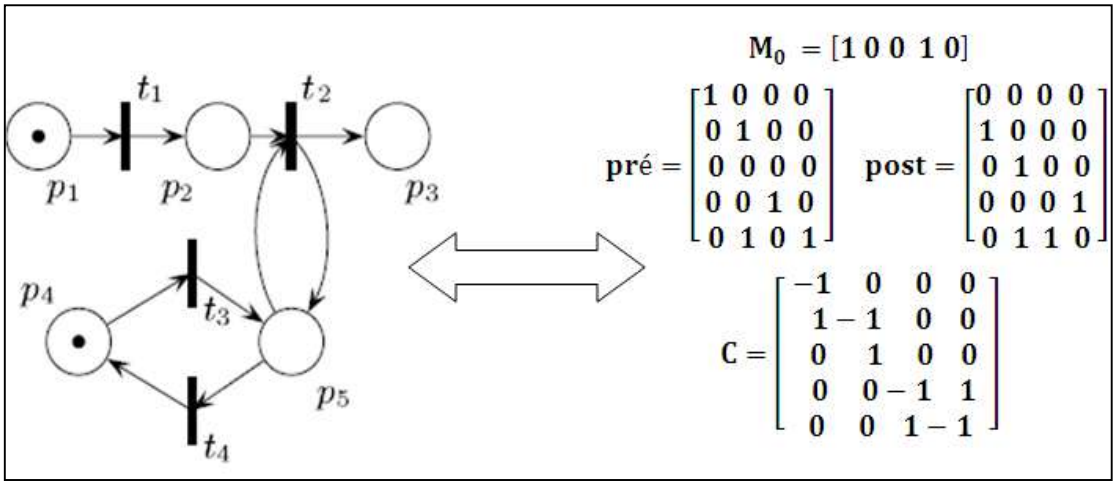


Figure II.2. *Représentation matricielle d'un RdP.*

2.3. Marquage

On appelle marquage une distribution de jetons sur les places. Le marquage initial noté *M0* est la distribution initiale de jetons dans le réseau à l'instant initial. Un marquage définit l'état du système.

Le marquage d'un RdP est une application $M : P \rightarrow N$ donnant pour chaque place le nombre de jetons qu'elle contient. Le marquage de la place *pi* est noté par *M (pi)* qui est un nombre entier.

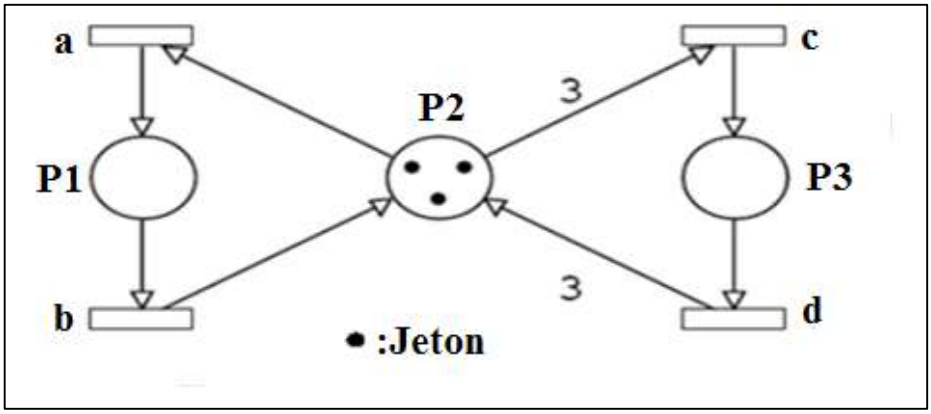


Figure II.3 *Réseau de Petri marqué.*

3. Sémantique d'un réseau de Petri

3.1. Franchissabilité et franchissement

Une transition t est franchissable à partir d'un marquage M (dénoté M[t>) si pour toute p :

$$\mathbf{M(p) \geq Pre(p, t).} \qquad \mathbf{(II.3)}$$

Le marquage M', atteint en franchissant la transition t, est calculé pour toute place p, par l'équation suivante :

$$\mathbf{M'_p = M_p - Pre\ (p, t) + post\ (p, t)} \qquad \mathbf{(II.4)}$$

Le franchissement de t est noté : M[t>M' le fait que t est franchissable à partir de M et que M'est le marquage obtenu.

Une séquence de franchissement « s » est une suite de transitions {t1, t2, …, tn} qui permet, à partir d'un marquage M, de passer au marquage M' par le franchissement successif des transitions définissant la séquence [**34**].

3.2. Graphe d'accessibilité

L'évolution temporelle d'un réseau de Petri marqué N =< R, M0 > peut être décrite par un graphe de marquage représentant l'ensemble fini des marquages accessibles A(R;M0) et d'arcs correspondant aux franchissements des transitions faisant passer d'un marquage à l'autre (voir la figure II.4).

Tel que un arc relie deux marquages M_i et M_j si et seulement s'il existe une transition t ∈T telle que Mi $\xrightarrow{t}$Mj [**34**].

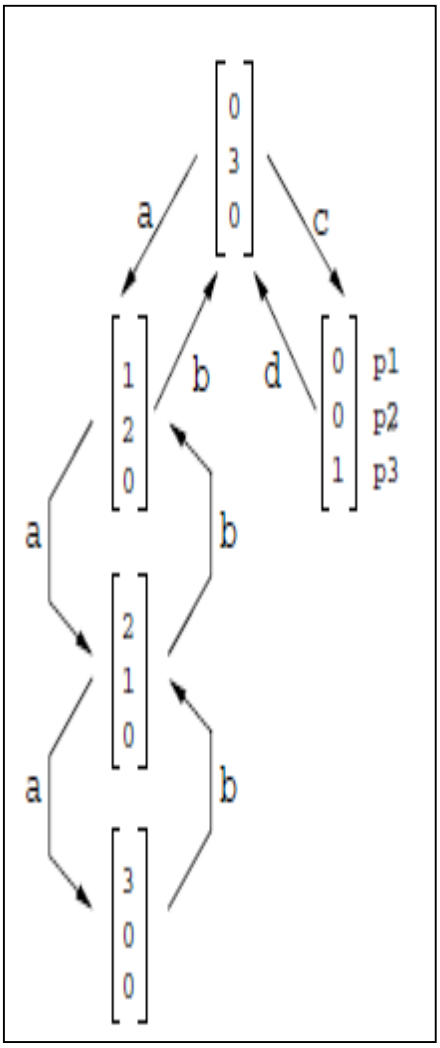


Figure II.4. *Graphe d'accessibilité de réseau de Petri de la figure II.3.*

4. Propriétés des réseaux de Petri

4.1. Vivacité

4.1.1. Quasi-vivacité d'une transition

La quasi-vivacité d'une transition signifie que depuis le marquage initial, cette transition peut être franchie au moins une fois. Autrement dit **[11]** :

$$\mathbf{t \in T\ est\ quasi{-}vivante} \leftrightarrow \exists \mathbf{M} \in (\mathbf{R}, \mathbf{M0})\ \mathbf{tel\ que}\ M[\mathbf{t}> \qquad \mathbf{(II.5)}$$

4.1.2. Vivacité d'une transition pour un réseau marqué (R, M0):

$$\mathbf{t \in T\ est\ vivante} \leftrightarrow \exists \mathbf{M} \in \mathbf{AR}, \mathbf{M}, \mathbf{t\ est\ quasi{-}vivante\ pour\ M} \qquad \mathbf{(II.6)}$$

4.1.3. Vivacité d'un réseau

Un réseau de Petri marqué N = <R, M0> est vivant si et seulement si toutes ses transitions sont vivantes [**11**] (voir la figure II.5).

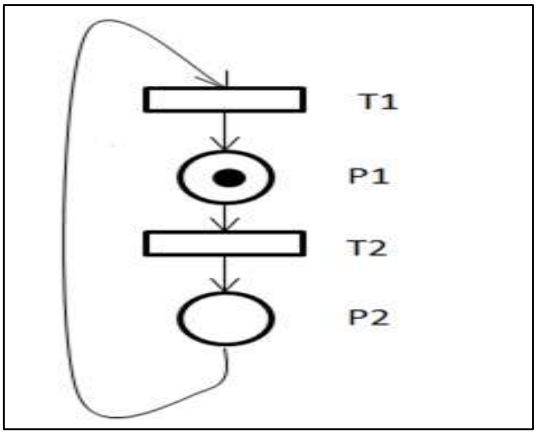


Figure II.5. *Exemple d'un RdP vivant.*

4.2. Blocage

Un blocage (ou état puits) est un marquage pour lequel aucune transition est validée. Un réseau de Petri est sans blocage si et seulement si depuis tout marquage accessible il est possible de franchir au moins une transition.

$$\mathbf{M\ \acute{E}tat\ puits} \rightarrow \forall\ \mathbf{t} \in \mathbf{T}\ \mathbf{pre}\ *, \mathbf{t} > \boldsymbol{M} \qquad \mathbf{(II.7)}$$

L'absence de blocage dans un RdP implique que le réseau a toujours la possibilité d'évoluer.

4.3. Caractère borné d'un réseau

Cette propriété définit et caractérise la possibilité pour une place d'accumuler une quantité bornée ou pas de jetons au cours de l'évolution d'un Réseau. Autrement dit :

$$\textbf{P est k-bornée} \leftrightarrow \forall M \in A(R, M0), M(p) \leq k \qquad \textbf{(II.8)}$$

Un réseau marqué est borné si toutes ses places sont bornées. Les réseaux 1-bornés sont appelés des réseaux saufs [**11**]. La figure II.6. Illustre un exemple d'un RdP 2-bornés.

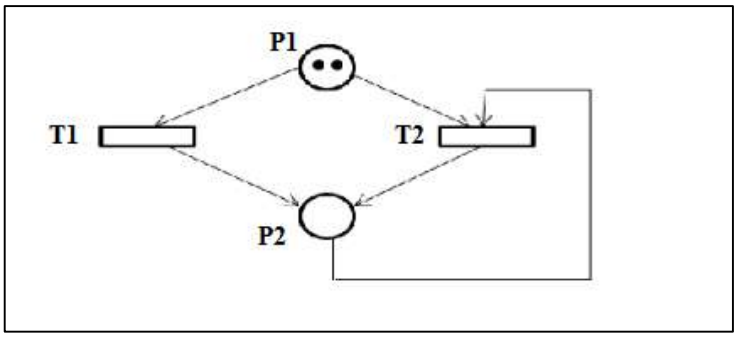


Figure II.6. *RdP borné.*

4.4. Conflit

4.4.1. Conflit structurel

Deux transitions T1 et T2 sont en conflit structurel si et seulement Si elles ont au moins une place d'entrée en commun [**2**] (voir la figure II.7).

$$\exists p \; Pre(P, T1) \,.\, Pre(P, T2) \neq 0 \qquad \textbf{(II.9)}$$

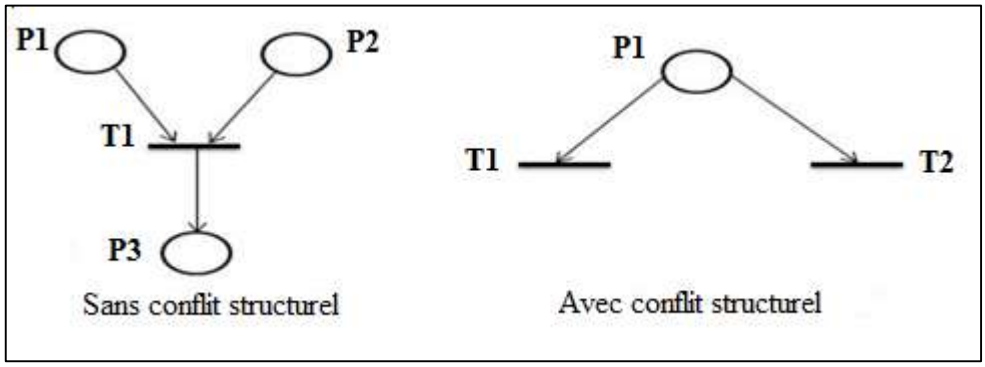


Figure II.7. T*ransition avec / sans conflit structurel.*

4.4.2. Conflit effectif

Elles sont en conflit effectif pour un marquage M si et seulement si T1 et T2 en conflit structurel et que [**2**] (voir la Figure II.8) :

$$\mathbf{M \geq Pre}(.\ , \mathbf{T1}) \qquad \mathbf{(II.10)}$$

$$\mathbf{M \geq Pre}(.\ , \mathbf{T2}) \qquad \mathbf{(II.11)}$$

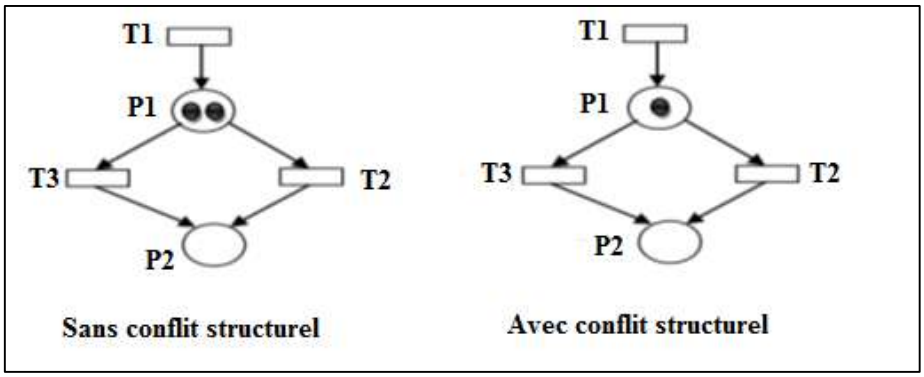


Figure II.8. *Conflit effectif.*

5. Réseaux de Petri récursifs

5.1. Définition

Le modèle des réseaux de Petri récursifs est une extension des réseaux des Petri ordinaire, qui introduit par Haddad et al. [**20**][**15**]. Et utilisés depuis pour la modélisation et vérification de systèmes concurrent où la création et l'annulation Dynamique de processus est requise tel que les systèmes multi-agents [**15**].

- La particularité de RdPR est distingué deux type de transitions :
  - **Transition élémentaire :** représenté par un rectangle simple comme les transitions de Réseaux de Petri ordinaire. Son franchissement entraîne l'exécution de l'action élémentaire qui lui est associée.
  - **Transition abstraite :** représenté par un rectangle à double bord. Son franchissement donne lieu à l'activation d'un RdPR. Dans un RdPR l'état en cours d'un système est décrit par un arbre de processus appelé marquage étendu.

5.2. Syntaxe

Un RdPR est défini par un n-uplet R=<P, T, I, W-, W+, Ω, ϒ > où :

- P est un ensemble fini de places.
- T est un ensemble fini de transitions où P∩T =Ø. Une transition de T peut être élémentaire (T_{el}) ou abstraite (T_{ab}).
- I est un ensemble fini d'indices, généralement appelé les indices de terminaison.
- W- est la fonction des préconditions de P× T dans N.
- W+ est la fonction de la post-condition de P × (Tel ∪ (Tab× I)) dans N.
- Ω est la fonction du marquage initial défini de T_{ab} à N^P. Elle associe à chaque transition abstraite T_{ab} un marquage ordinaire.
- ϒ est une famille indexée de représentations effective d'ensembles semi linéaires de marquages finaux.

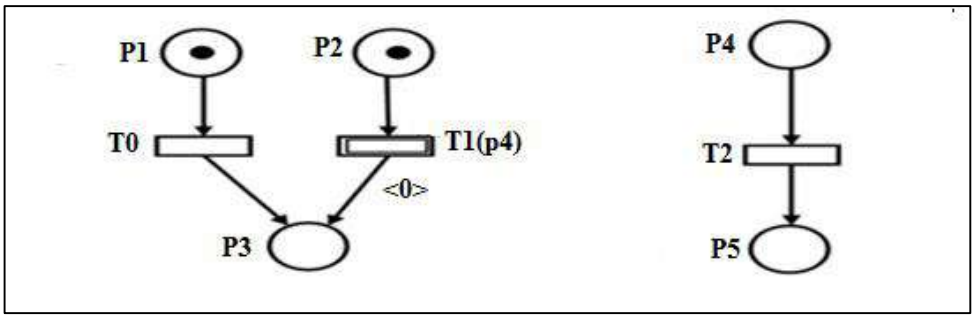


Figure II.9. *Exemple d'un RdPR simple.* **[**20**]**

5.3. Marquage étendu

5.3.1. Définition

Le marquage étendu d'un RdPR R= < P, T, I, W-, W+, Ω, ϒ > est un arbre marqué Tr= < V, M, E, A > Où :

- V est l'ensemble fini des nœuds de l'arbre. lorsque l'arbre Tr n'est pas vide, $V_0 \in$ V dénote la racine de l'arbre.
- M : V →N^P est une application qui associe à chaque nœud un marquage ordinaire.
- E ⊆ V *V est l'ensemble des arcs.
- A : E →Tab est une application associant une transition abstraite à chaque arc de l'arbre.

La figure II.10 représente une partie de l'arbre de marquage crée par la séquence de franchissement des transitions de RdPR de la figure II.9. [**15**]

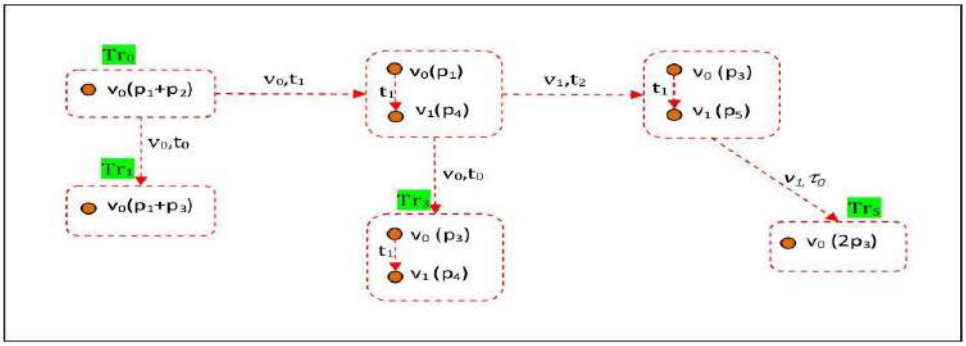


Figure II.10. *Etat du réseau de Petri récursif : Arbre de marquage.*

5.4. Transition franchissable

5.4.1. Définition

Une transition t est franchissable dans un nœud v (ou dans le marquage M(v)) d'un marquage étendu tr ≠ ⊥ (dénotée par Tr $\xrightarrow{t,v}$) si ∀p ∈P, M(v) (p) ≥W^-(p, t).

Une étape de coupure τ_i (avec i ∈I) est franchissable dans v (ou dans le marquage M(v)) (dénote par Tr $\xrightarrow{t,v}$) si M(v) ∈Υ_i.

5.4.2. Franchissement d'une transition élémentaire

Le franchissement d'une transition élémentaire t dans un nœud v d'un marquage étendu Tr = < V, M, E, A > conduit à un marquage étendu Tr' = < V', M', E', A' >

(noté $Tr \xrightarrow{t,v} Tr'$) tel que :

Pour (E″ {(v, v') ∈ E et défini} et V″ = {v`∈V | (v, v`) ∈ E″}

- ∀p ∈ P, M(v) (p) ≥ W^-(p, t).
- V' = V | (∪v ∈v" succ (v')) E'=E ∩ (v'*v').
- ∀ e ∈ E', À(e)= A(e).
- ∀ v' ∈V' | {v}, M'(v') =M (v').
- ∀ p ∈ P, M'(v). (p) =M(v) (p)-W^-(p, t)+ W^+ (p, t)+$\sum_{e\in E''} W^+$ (p, A(e)).

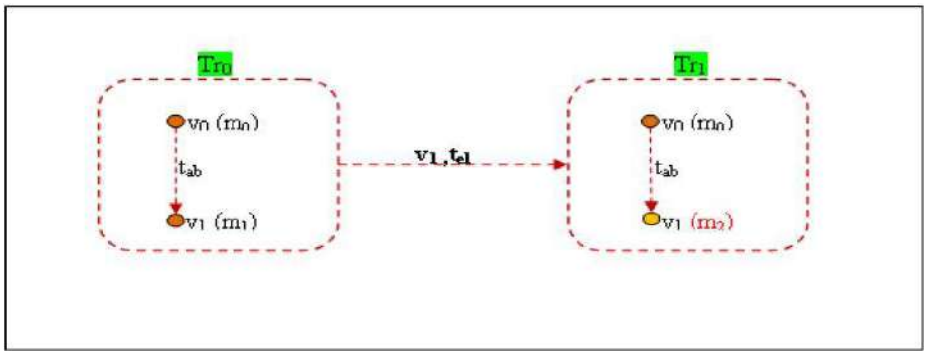


Figure II.11. *Franchissement d'une transition élémentaire.*

5.4.3. Franchissement d'une transition abstraite

Le franchissement d'une transition abstraite t dans u nœud v d'un marquage étendu Tr = < V, M, E, A > conduit au marquage étendu Tr' = < V', M', E', A' >

(Noté Tr $\xrightarrow{v,t}$ Tr') si et seulement si :

- Soit v' un nouvelle identificateur dans l'arbre Tr'.
- ∀p ∈ P, M(v) (p) ≥ W^-(p, t).
- V' = V| (∪ (v') E'=E U {(v, v'}, ∀ e ∈ E', À(e)= A(e), A'((v,v'))=t.
- ∀ v″ ∈V{v}, M' (v″) =M(v″).
- ∀ p ∈ P, M'(v)(p) =M(v)(p)-W^-(p,t).
- M'(v')= Ω(t) le nouveau nœud est associé par le marquage initial de t.

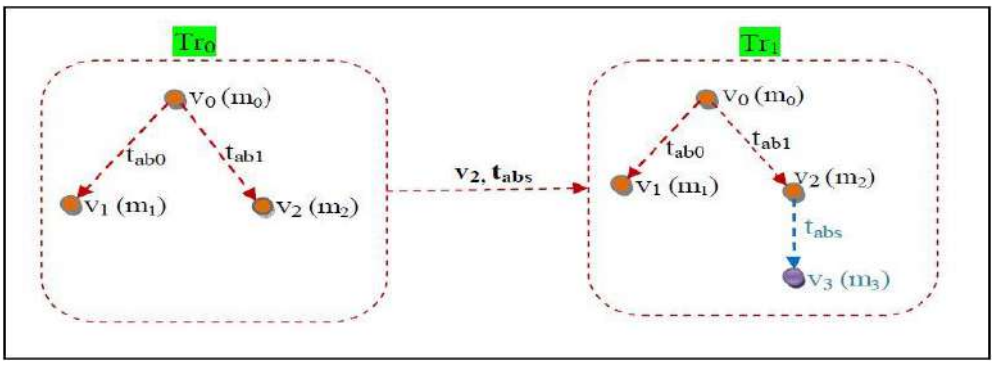


Figure II.12. *Franchissement d'une transition abstraite.*

5.4.4. Franchissement d'une étape de coupure

Le franchissement d'une étape de coupure d'un nœud de l'arbre élimine le sous-arbre issu de ce nœud, et produit les jetons en sortie de la transition abstraite qui a généré ce nœud.

Formellement, le franchissement d'une étape de coupure τ_i par un nœud v d'un Marquage étendu Tr = <V, M, E, A> survient quand M(v) ϵ Υ_i mène au marquage Étendu Tr' = <V', M', E', A'> (dénoté par $Tr \xrightarrow{\tau_i, v} Tr'$) Ssi :

- (v = v0) alors Tr' = ϕ,
- (v ≠ v0) alors :
- V' = V \ Succ(v), E' = E ∩ (V'× V'), ∀ e ∈ E', A'(e)= A(e),
- ∀ v'∈ V \ {pred(v)}, M'(v') = M (v'),
- ∀ p ∈ P, M' (pred(v)) (p) = M(pred(v))(p)+W+(p,A(pred(v),v),i).// i représente l'indice de terminaison associé à Yi.

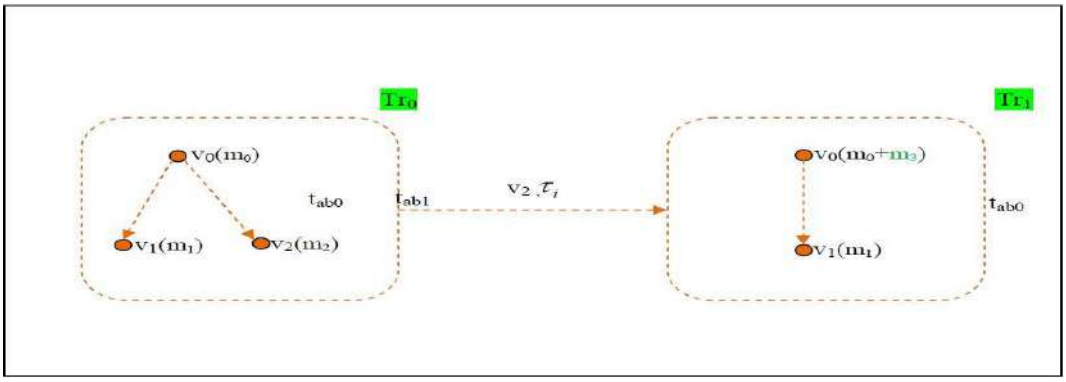


Figure II.13. *Franchissement d'une étape de coupure.*

6. Les RECATNets

6.1. Définition

Les RECATNets (Récursive Extended Concurrent Algebraic Term Nets) [**4**], [**18**], sont une catégorie de réseaux de Petri algébriques de haut niveau, introduisant la notion de récursivité (premièrement introduit dans les réseaux de Petri [**20**]).

Les RECATNets offrent un mécanisme récursif pratique pour modéliser la création et la suppression dynamiques de processus concurrents ainsi que leur synchronisation.

Ils ont deux types de transitons :

- Les transitions élémentaires (représentées par un rectangle simple, voir figure II.14)
- Les transitions abstraites (représentées par un rectangle à double bord, voir la figure II.15).
- Chaque transition abstraite est associée à un marquage de départ représenté dans un cadre adjacent à cette transition.

Dans les RECATNets, chaque place est associée à une sorte et les jetons qui y résident sont des multi-ensembles de termes algébriques étant de la même sorte que cette place. De plus, les arcs reliant les places et les transitions d'un RECATNets sont étiquetés par des multi-ensembles de termes algébriques construits à partir de la signature multi-sorte et de l'ensemble des variables associées à ce réseau. Un arc entrant (p, t), dans un RECATNets, reliant une place p à une transition t (élémentaire ou abstraite) est étiqueté par les deux inscriptions IC(p, t) (*Input Condition*) et DT(p, t)(*Destroyed Tokens*). L'expression IC(p, t) précise les conditions sur le marquage de la place d'entrée p pour le franchissement de la transition t. L'expression DT(p, t) précise les termes détruits de la place p lorsque la transition t est franchie. Une transition t peut être aussi étiquetée par une inscription de garde TC(t) (*Transition Condition*). Cependant, les arcs sortant d'une transition dans un RECATNets sont étiquetés par des expressions différentes selon si cette transition est élémentaire ou abstraite.

Pour une transition élémentaire t, un arc sortant (t, p′) la reliant à une place p′ est étiqueté par l'inscription CT(p′, t) (*Created Tokens*). Par contre, pour une transition abstraite t, un arc sortant (t, p′) la reliant à une place p′ est étiqueté par une expression ICT (p′, t, i) (*Indexed Created Tokens*). Ces deux expressions précisent les jetons à créer dans la place de sortie p′ de la transition t.

On note que dans la représentation graphique des transitions d'un RECATNets donnée dans les Figures.II.14 et II.15, les places p et p′ sont respectivement de sorte s et s′ et de capacité c et c′. Pour une transition donnée t, on note sur son arc d'entrée (p, t) la condition d'entrée IC (p, t) en gras et italique pour la distinguer de l'expression DT (p, t). Par simplification, dans le cas où IC (p, t) =DT(p, t), une seule de ces inscriptions est précisée sur l'arc (p, t).

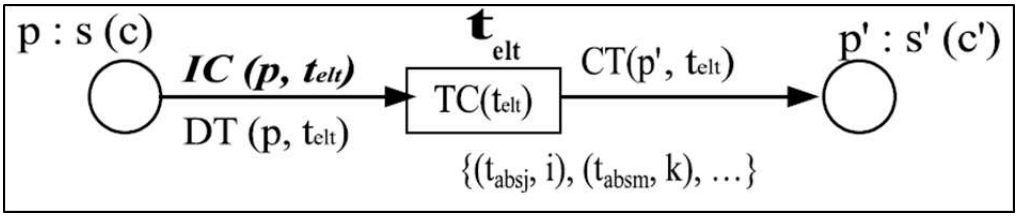


Figure II.14. *Transition élémentaire*

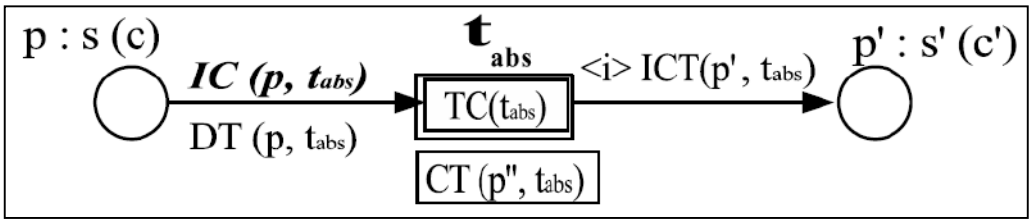


Figure II.15. *Transition abstraite.*

6.2. Exemple d'un RECATNet

La figure II.17 modélise une simple interaction entre un client et un serveur. Le client envoie sa requête sauvegardée dans la place *Pstart* au serveur à travers le franchissement de la transition abstraite *SendReq*. A la réception d'une requête, le serveur traite cette dernière et envoie une réponse vers le client. Deux réponses peuvent être envoyées par le serveur :

- Réponse OK : dans le cas où la requête a été traitée sans anomalie chez le serveur
- Réponse NotOK : dans le cas où le serveur a trouvé des anomalies lors du traitement de la requête. Par conséquent, deux états de terminaison peuvent être distingués au niveau de serveur représentés par deux marquages finaux Y_0 et Y_1.

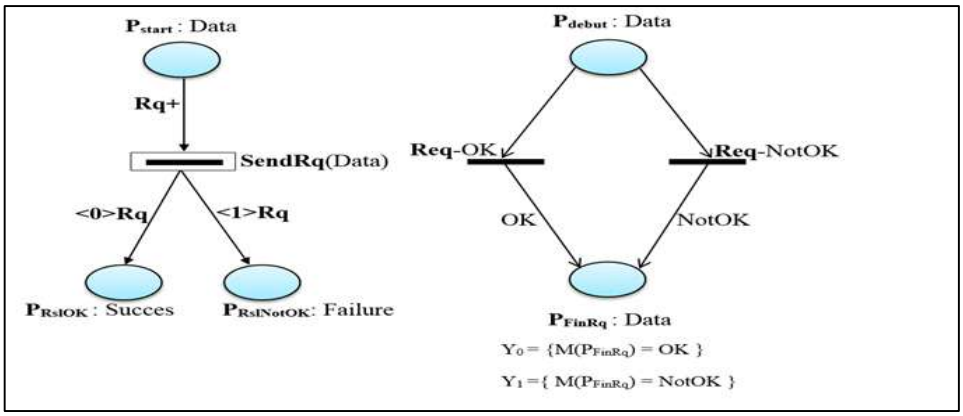


Figure II.16. *Exemple d'un RECATNet.*

7. Conclusion

Les réseaux de Petri (RdP) est une méthode qui se base sur une théorie mathématique très solide, ce qui permettant de l'utiliser comme un outil de modélisation, d'analyse et de simulation des systèmes complexes.

Les RdP disposent de nombreuses techniques de vérification automatique des propriétés des modèles. Il est possible de démontrer de nombreuses propriétés fort utiles : le caractère k-borné, le blocage, la vivacité, etc. Par conséquent, le RdP semble un modèle formel très intéressant, pour tester la fiabilité des systèmes électroniques.

Nous avons introduit le modèle des réseaux de Petri récursifs pour lequel nous avons vérifié certain propriété.

Le RdPR cache la limitation des réseaux de Petri ordinaires. Les réseaux de Petri récursifs sont définis en enrichissant les réseaux de Petri avec des mécanismes de création et suppression dynamiques de processus.

Enfin, nous avons présenté le modèle RECATNets qui sont de leur tour une catégorie plus puissante de RdPR vu leur aspect algébrique de vérifier les systèmes concurrents.

CHAPITRE III : TRANSFORMATION D'AUML VERS RECATNETS

1. Introduction

Dans ce chapitre nous intéressons au formalisme AUML (**Agent Unified Modeling Language)** qui s'appuie sur UML, c'est à dire il étend UML pour décrire les spécificités des SMA. Puis on propose un ensemble des règles de transformations du formalisme AUML vers RECATNet, et enfin on clôt le chapitre par un exemple d'application FIPA-Broking-Protocol.

2. Description d'AUML

AUML « Agent UML » est un projet relativement jeune proposé par Bauer, Odell et al en 1999 **[37]** et les premiers travaux de spécification ont débuté en 2003. AUML vise à proposer un ensemble de notations mieux adaptées au paradigme multi-agent, en étendant le langage UML.

AUML a été développé par l'équipe FIPA appelée FIPA Modeling Technical Committee (Modeling TC) qui a pour but les objectifs suivants :

- Permettre aux développeurs de mieux comprendre comment modéliser des applications orientées agent.
- Être en accord avec les spécifications existantes de FIPA (Fondation for Intelligent Physical Agents) et OMG (Object Management Group)**.**
- Avoir un aperçu rapide du déroulement d'un protocole. rappelons que les protocoles d'interaction du modèle FIPA-ACL sont spécifiés en AUML **[1].**
- Représentation du comportement interne à un agent **[21].**

Notons, qu'AUML est un formalisme de modélisation qui est actuellement difficile à cerner **[30]** ; et qu'il concerne les diagrammes de séquence pour modéliser les protocoles d'interaction entre agents ainsi que les diagrammes de classe pour modéliser les agents.

3. Les Diagrammes AUML

L'extension des diagrammes UML concerne :

- Le diagramme de classes, qui représente la structure statique du SMA.
- Les diagrammes d'interactions comme le diagramme de séquence.

Ceux-ci permettent de modéliser le comportement dynamique du système.

Dans ce mémoire nous nous focalisons sur le diagramme de séquence en étant le diagramme le plus utilisé pour la modélisation des protocoles d'interactions entre les agents.

3.1. Le diagramme de séquence AUML

Le diagramme de séquence est défini comme suit en UML: le diagramme de séquence est un diagramme qui montre les interactions des objets disposés dans la séquence temporelle. En particulier, il montre les objets participant à l'interaction et la séquence de messages échangés**.** En effet, le diagramme de séquence en AUML est initialement adopté par la FIPA pour exprimer des protocoles d'interaction des agents. Ainsi, on considère des agents et non des objets, donc il faut lire agents au lieu des objets dans la précédente définition. Les diagrammes de séquence dans les systèmes Multi-agents sont des schémas qui expriment l'échange de messages dans les protocoles **[16].** Donc les déférences qui existent entre le diagramme de séquence en UML et en AUML sont :

a) Dans le diagramme de séquence AUML il existe une spécification plus riche du rôle d'un agent **[21]**.

b) L'extension du diagramme de séquence donne des étiquettes (acte de communication), aux flèches au lieu du message style orienté objet **[21].**

c) L'ajout de nouveaux types de branchements dans les diagrammes de séquence AUML afin de prendre en compte l'indéterminisme du comportement d'un agent. La figure III.1 montre les trois types de branchement: branchement ET, branchement OU et branchement XOR **[37].**

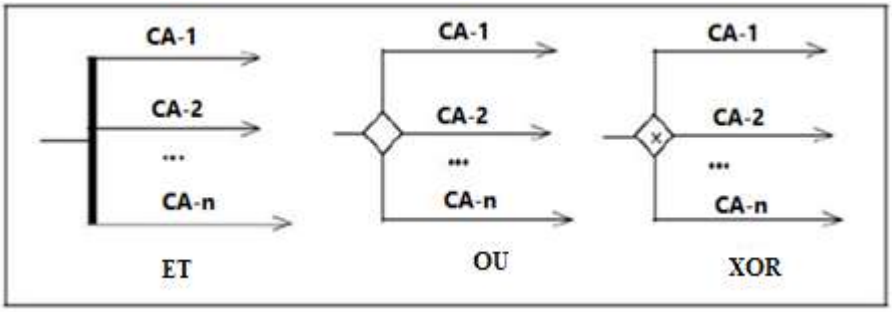


Figure III.1. *Différents branchements en AUML.*

Le diagramme de séquence AUML se compose de deux dimensions:

- ❖ Verticale : pour le temps.
- ❖ Horizontale : pour les différents rôles ou bien pour les agents jouant un rôle spécifique**.**

Dans les diagrammes de séquence AUML, les messages sont ordonnés selon un axe de temps déroulant selon la dimension verticale de haut en bas. Cependant, l'ordre des messages est implicitement défini par l'axe des temps **[16].**

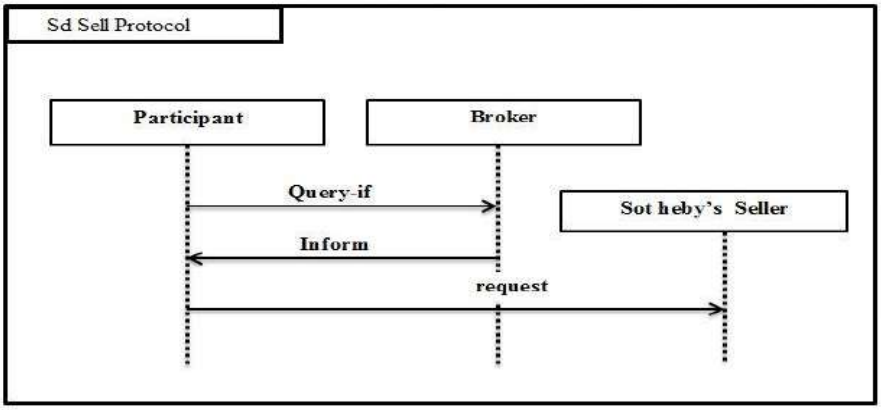


Figure III.2. *Exemple d'échange de message dans AUML.*

Un message définit une communication particulière entre deux lignes de vie. Les expéditeurs et les récepteurs d'un message peuvent être sur la même ligne de vie ou non **[16].** Deux situations doivent être prises en considération dans la ligne de vie de l'expéditeur et la même chose pour la ligne de vie du récepteur.

L'acte de communication est annoté dans le diagramme de séquence AUML sur la flèche dont l'expéditeur est l'origine de la flèche et le récepteur est à sa pointe **[30].** Ainsi, le langage et l'ontologie sont écrits dans un commentaire lié au diagramme. Ainsi, en raison de clarté le contenu du message n'est pas toujours inclus, mais peut être ajouté à la droite de la flèche**.**

4. Modélisation des protocoles d'interactions entre agents

A-UML modélise les protocoles d'interaction entre agents (ou AIP : Agent Interaction Protocol) sur trois niveaux de définition :

4.1. Premier Niveau

Ce niveau permet de représenter le protocole global **[21]** c'est-à-dire ne fournit qu'une spécification de base sur le protocole. En effet, on retrouve dans ce niveau : le diagramme de séquence, Package et Template « gabarit »**.**

- Le diagramme de séquence

Est détaillé précédemment dans la partie diagramme AUML.

- Package

Le package en AUML est représenté par un rectangle aux bordures pleines avec un onglet en haut à gauche auquel il est écrit le mot clé « sd » (séquentiel diagramme) suivit du nom du protocole **[30].**

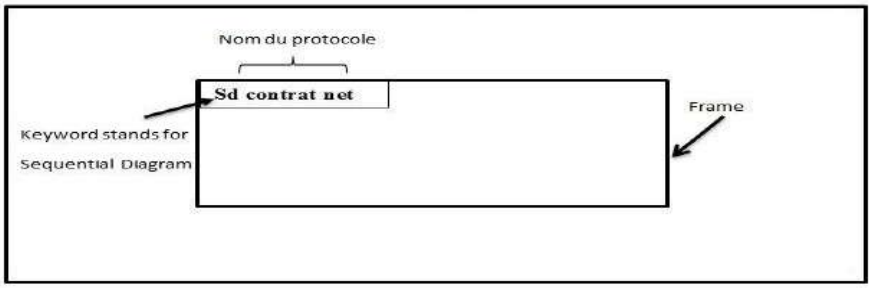


Figure III.3. *Paramètres du package AUML***.**

- Template « Gabarit »

Une Template « Gabarit » est un élément paramétrable, dont les paramètres sont attachés quand le nouveau modèle est produit **[21].** L'utilisation des Template est important car au lieu de définir une composante réutilisable. La personnalisation des paquets devrait être incluse**.** En effet, les paramètres du Template sont divisés en trois catégories : les paramètres de rôle, les contraintes et les actes de communication **[37].**

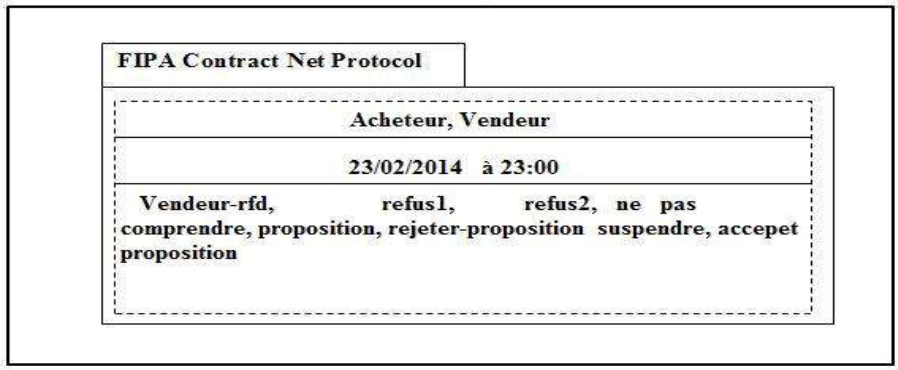


Figure III.4. *Notation du Template en AUML*

Le Template est représenté à travers un package avec un trait interrompus dans le coin haut droit du package **[37].**

4.2. Deuxième Niveau

Le deuxième niveau représente les interactions entre les agents grâce à des diagrammes d'interactions : diagrammes de séquences, diagrammes de collaborations, diagrammes d'activités ou bien diagrammes d'états/transitions**.**

4.3. Troisième Niveau

La modélisation d'un protocole exige, la représentation intérieure qui survient dans un agent lors de l'exécution d'une procédure **[21].** Pour cela le troisième niveau est consacré pour la représentation correspondant aux processus internes aux agents qui sont principalement modélisés par des diagrammes d'états/ transitions **[37].**

Dans notre travail nous nous intéressons aux deux premiers niveaux pour la modélisation des interactions entre les agents.

5. Transformation du diagramme AUML vers RECATNet

Pour faciliter la transformation du modèle AUML des interactions des agents en RECATNet correspondant, nous avons mis une hypothèse et définis les règles de traduction suivantes :

- Une transition dans le RECATNet peut correspondre à une émission / réception d'un acte ou à un traitement. Un traitement est l'exécution d'une certaine activité de l'agent.
- Une place représente l'état de l'agent (agent attente) ou message stock
- Le type d'agent est l'ensemble de tous les agents impliqués dans l'interaction.
- Les types d'actes sont= {Failure, Refuse, Agree, ...}.
- Un sous-RECATNet comprend une place de démarrage, un marquage initial et d'un ensemble des transitions à franchir. Le type de ces transitions peut être élémentaire, final ou abstrait (voir le chapitre II).
- Un message échangé entre deux agents est représenté avec la synchronisation place (place intermédiaire ou place Interface) et deux liens (le premier relie la transition de l'émission avec la place de la synchronisation et le second relie la transition intermédiaire avec la transition de réception).
- Une transition finale, constitue la fin d'une interaction.
- Les transitions abstraites sont utilisées pour modéliser une récurrence ou sous-protocole entièrement développé dans le schéma AUML.

Enfin, voici quelques règles permettant de faire la transformation graphique du modèle AUML vers RECATNet.

Les transitions utilisées sont tous de types élémentaires, sauf celle mettant fin à une interaction qui est de type final ou celle qui fait appel à un sous-RECATNet qui est de type abstraite.

❖ **Règle1 : Entité agent**

Tableau III.1. *Entité agent*

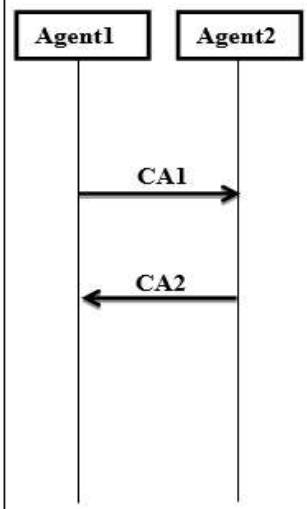
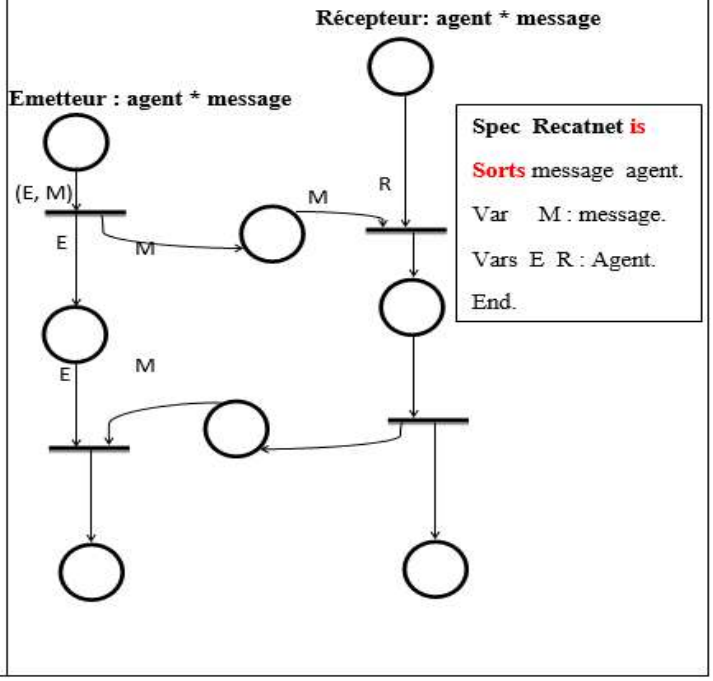
AUML	RECATNet
	

Description :

Chaque entité agent ou classe /agent dans le diagramme AUML est représenté dans le RECATNet par une place. Cette place peut contenir à la fois un ou plusieurs agents de types agent.

❖ **Regle2 : Envoi/Réception (synchrone)**

Tableau III.2. *Envoi/Réception (synchrone).*

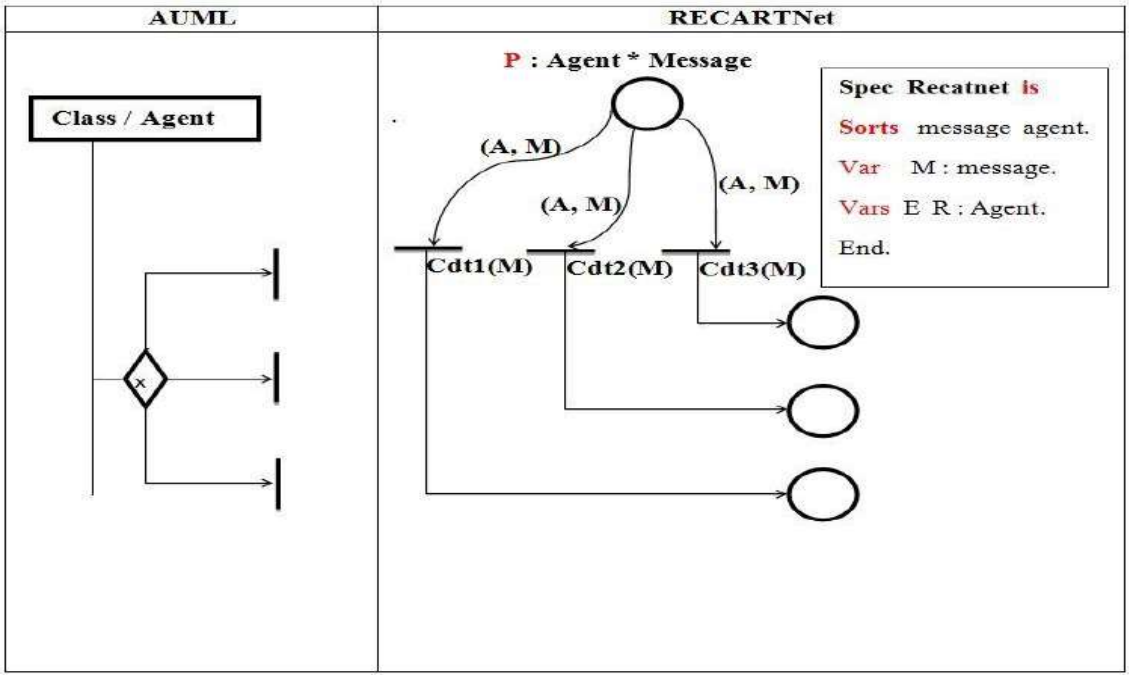
AUML	RECATNET
	

Description :

Cette règle explique le déroulement d'un échange des messages entre deux agents.
Comme on a vu dans la règle1, l'agent et le message sont représentés par des places. L'envoi d'un message est représenté par une transition représentant l'action de l'émission et une place intermédiaire notant que l'élément envoyé est de type message, ensuite l'agent reste en attente jusqu'à ce qu'il reçoive une réponse de la part de destinateur.
De son tour le recevant du message doit formuler une réponse puis la retourner à l'émetteur de la même manière que l'émetteur avait envoyé son message.

- **Règle3 : Condition**

Tableau III.3. *Représentation d'une condition en RECATNet*

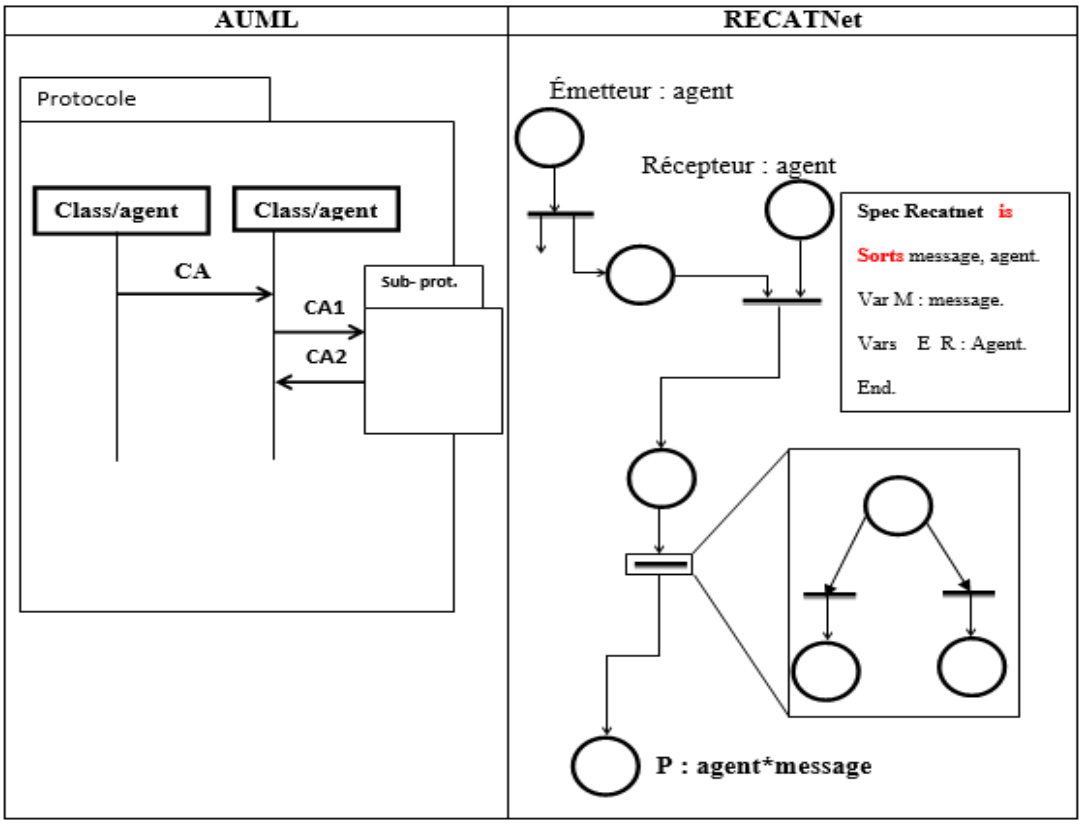


Description :

Dans cette règle on montre le mécanisme de la sélection d'une destination, en utilisant la décomposition de la représentation XOR AUML en trois cas :
Le premier cas si le message est de type Cdt1.
Le deuxième s'il est de type Cdt2 et
Le troisième si de type Cdt3. Enfin l'agent reste en attente jusqu'à ce qu'il reçoive une réponse.

❖ **Règle4 : Transition abstraite**

Tableau III.4. *L'appel d'un sous protocole*



Description :

Dans le schéma ci-dessus on a montré que l'appel d'un sous-protocole se fait à travers une transition abstraite, qui va donner lieu à son tour à l'execution d'un sous-RECATNet c'est-à-dire à un nouveau RECATNet que va nous retourner lui aussi un resultat après un certain traitement via aussi la transition abstraite.

6. Application sur FIPA-Broking Protocol

Le protocole de courtage FIPA [**38**] vise à appuyer les interactions de courtage en matière de médiation de systèmes multi-agent. Un agent d'échange est un agent qui fournit des services facilitant la communication entre les agents à l'aide de connaissances sur leurs besoins et leurs capacités.

Rappelons que les types d'actes possibles sont : **Request, Refuse, Agree, Failure-FNM, Failure-FC, Failure-IDP, Failure-FB, Reply-RMSP.**

Le protocole travaille comme suit:

❖ Un agent initiateur (Initiator).envoie un acte comme un **Request** vers l'agent d'échange(Broker).

❖ Lorsque l'agent d'échange reçoit l'acte qui a été envoyé, il peut répondre par **Refuse** (demande rejetée et l'interaction se termine) ou par **Agree** (demande acceptée) en cas d'acceptation, l'agent d'échange commence à chercher des agents qui peuvent accepter le **Request** envoyé par l'initiateur. S'il ne trouve pas d'agents ou si des problèmes de médiation se pose, il retourne un **FNM** (failure-no-match: no-potentiel agents, failure-request: problem of mediation…) Et l'interaction s'arrête. Dans le cas contraire, l'agent d'échange commence une interaction avec l'agent potentiel et envoie un **FC** (failure connexion) si la communication avec l'agent visé s'est interrompue sinon il retourne inform:**IDP** (inform-proxy-done), indiquant que la médiation a réussi et la communication se passe bien, et fait appel à un sous-protocole, à travers **PCA** (proxy-communicative-act) permettant à l'agent cible de traiter l'acte reçu. L'agent cible retourne sa réponse en utilisant l'acte **RM** (Reply-Message).

Enfin, l'agent d'échange renvoie à l'initiateur la réponse de l'agent cible, Reply : **RMSP** (Reply-Message-Sub-Protocol) dans le cas où le sous-protocole se termine avec succès et Failure : **FB** (Failure-Broking) pour le cas échéant. Puis l'interaction se termine.

- Le modèle AUML représentant notre protocole FIPA-Broking-Protocole

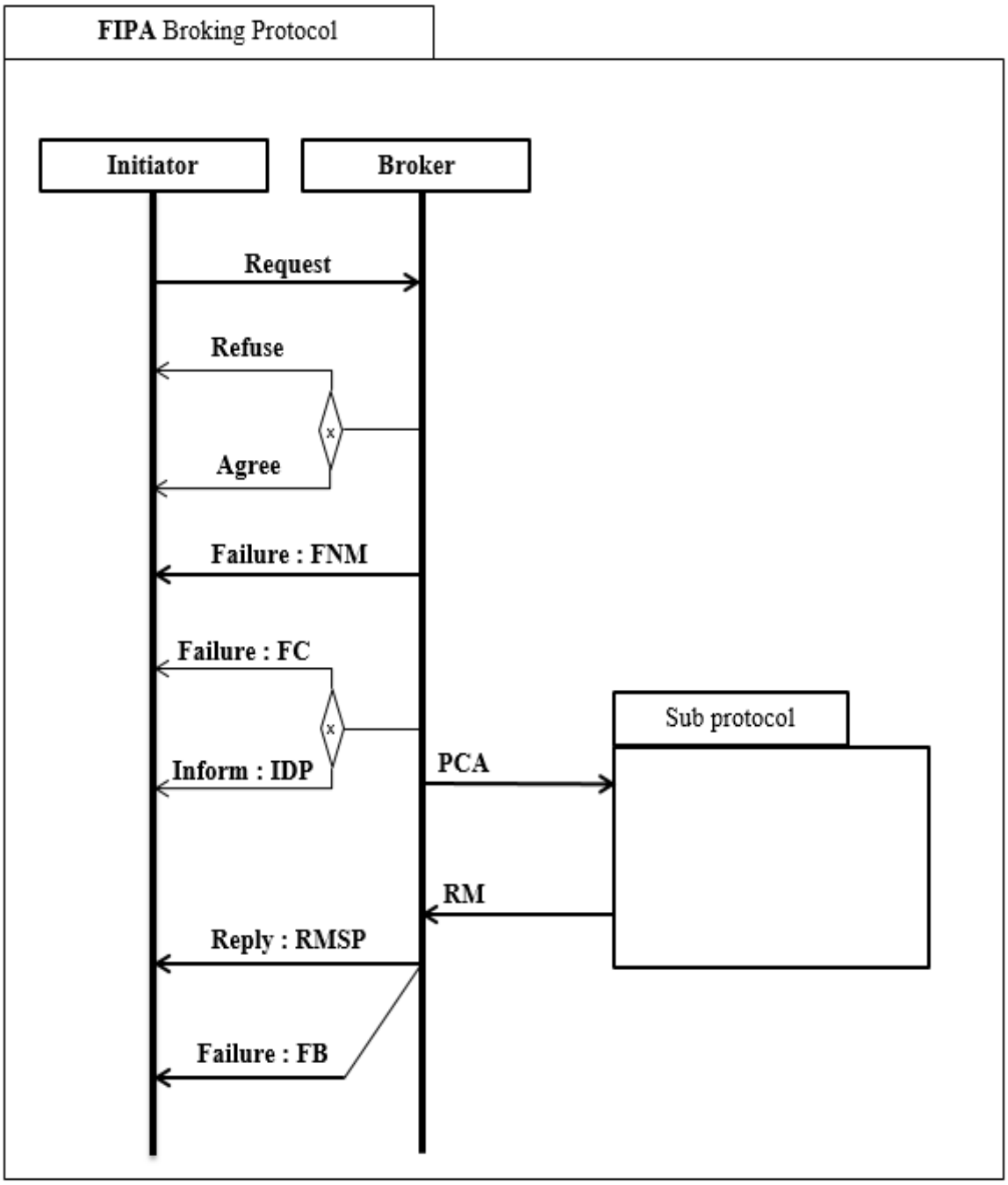


Figure III.5. *FIPA-Broking-Protocol: AUML.*

- RECATNet dérivé de FIPA-Broking-Protocole

Selon les règles définies précédemment nous obtenons le modèle de RECATNet suivant :

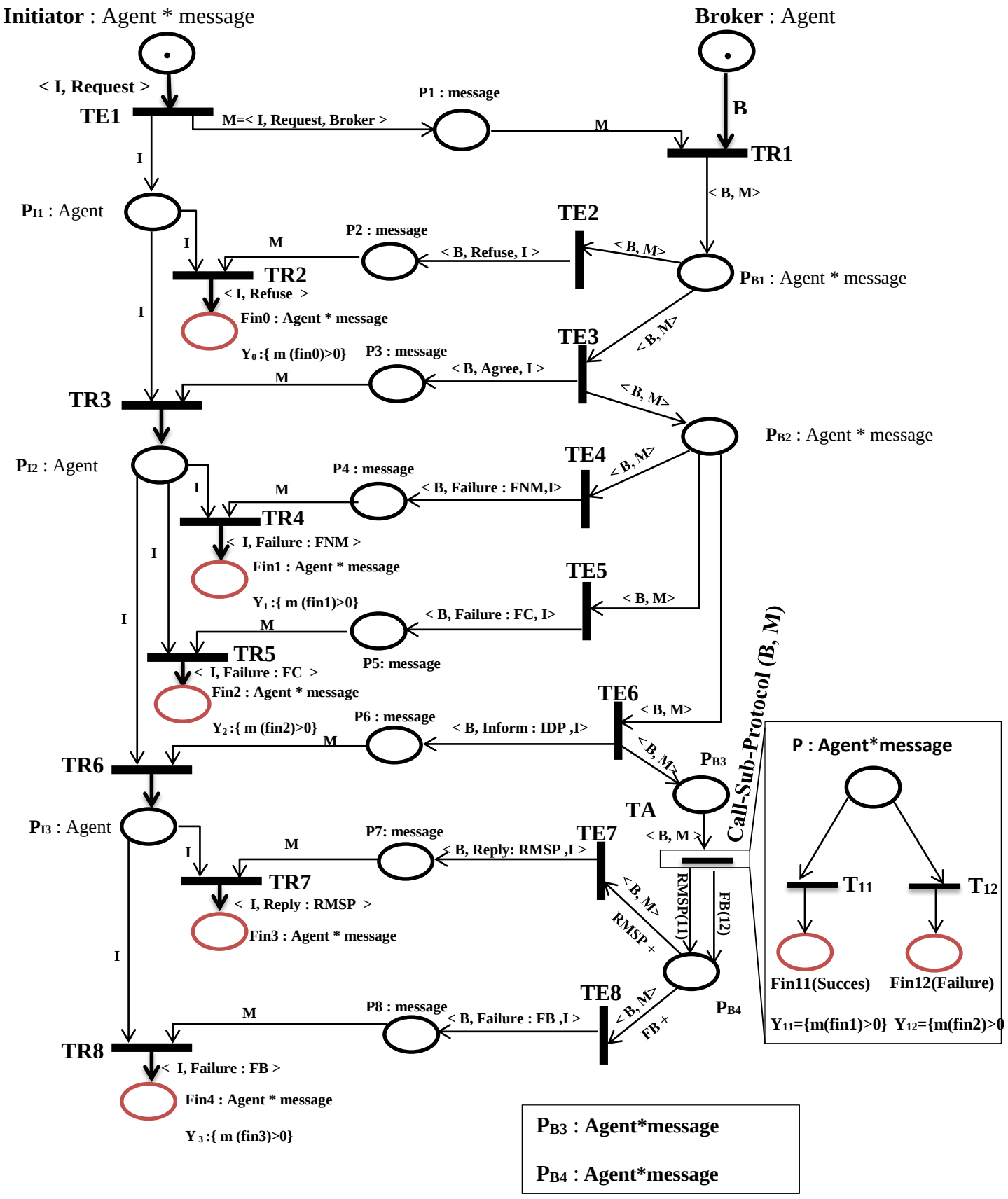


Figure III.6. FIPA-Broking-Protocol: RECATNet.

Voici la spécification algébrique définit les différents types utilisés dans notre exemple illustré dans la figure III.5:

```
Spec RECATNet is

Sorts Agent Message Flag.

Ops   Initiator Broker: -> Agent.

Ops    Request Refuse Agree   Failure_FNM   Failure_FC   Failure_IDP
Failure_FB      Reply_RMSP: Flag.

Ops <_, _, _>    Agent   Agent   Flag -> Message.

Var M   : Message.

Vars   I  B   : Agent.

End.
```

Le tableau ci-dessous montre la sémantique de transitions figurées dans le modèle de RECATNets de la figure III.6.

Tableau III.5. *Les transitions et leurs sémantiques*

Nom	**Sémantique**
TE1	Send Request
TR1	Recieve Refuse
TE2	Send Refuse
TE3	Send Agree
TE4	Send Failure : FNM
TE5	Send Failure : FC
TE6	Send Inform : IDP
TE7	Send Reply : RMSP
TE8	Send Failure : FB
TR2	Recieve Agree
TR3	Recieve Failure : FNM
TR4	Recieve Failure : FC
TR5	Recieve Inform : IDP
TR6	Recieve Reply : RMSP
TR7	Recieve Failure : FB
TR8	Recieve Failure : FB
TA	Call- sub-Protocol

7. Conclusion

Dans ce chapitre nous avons présenté le formalisme AUML : les principaux diagrammes et les différentes notations utilisées par ce formalisme qui ne sont pas complètement standardisées. Puis nous avons proposé quelques règles de transformations permettant la transformation du modèle AUML vers RECATNet.

Enfin, nous avons terminé le chapitre par un exemple de FIPA-Broking-Protocol sur lequel nous avons appliqué nos règles de transformations proposées.

L'objectif de ce chapitre suivant est de dédié à l'analyse des RECATNets obtenus par l'application de nos règles de transformation.

CHAPITRE IV : VERIFICATION ET VALIDATION

1. Introduction

Dans ce chapitre, nous présentons la logique de réécriture et son langage Maude. Puis on parle de la sémantique de RECATNet en terme de la logique de réécriture et enfin on donne un exemple de FIPA-protocole c'est ce lui de FIPA-Broking Protocole, et Son analyse en langage de réécriture Maude et on termine le chapitre par une conclusion.

2. Schéma général de vérification

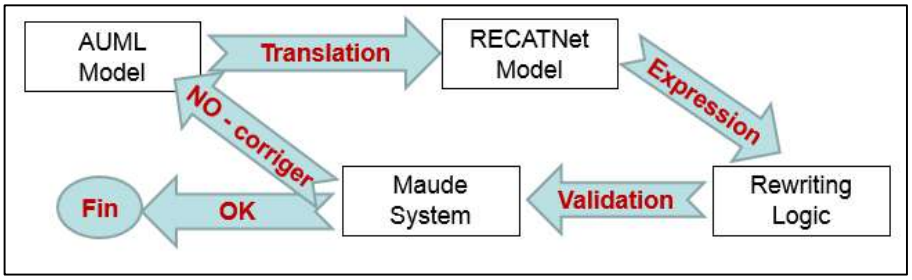


Figure IV.1. *Schéma général de vérification*

Ce schéma représente les étapes de validation/vérification d'un protocole par le système Maude, Ces étapes sont :

- **Etape 1 :** Génération du modèle AUML du protocole étudié.
- **Etape 2 :** La transformation du modèle AUML vers le modèle RECATNets.
- **Etape 3 :** L'écriture du modèle RECATNets en terme de la logique de réécriture.
- **Etape 4 :** La validation par le système Maude.
- **Etape 5 :** Soit 'Fin' si la validation marche correctement, ou bien retour à l'étape 1 dans le cas inverse.

3. Logique de réécriture & Maude

3.1. Logique de réécriture

La logique de réécriture est une logique qui permet de raisonner d'une manière correcte sur les systèmes concurrents non-déterministes ayant des états et évoluant en termes de transitions. Elle explique n'importe quel comportement concurrent dans un système avec une sémantique de 'vraie concurrence', par des déductions dans cette logique elle permet aussi de décrire les systèmes concurrents. Cette logique unifie plusieurs modèles formels qui expriment la concurrence.

La logique de réécriture a été introduite par Meseguer [**28**].

Dans la logique de réécriture, chaque système concurrent, est représenté par une théorie de réécriture $\Re = (\Sigma, E, L, R)$ structure statique est décrite par la signature (Σ, E), tandis que sa structure dynamique est décrite par les règles de réécriture R. Une conséquence intéressante des définitions de la logique de réécriture est qu'une théorie $\Re = (\Sigma, E, L, R)$ devient une spécification exécutable du système concurrent qu'elle formalise. Une théorie de réécriture $\Re$ est une 4-tuple $\Re = (\Sigma, E, L, R)$, tel que (Σ, E) est une signature ; est un ensemble des sortes et opérations, et E est un ensemble de Σ-équations. La signature (Σ, E) est une théorie équationnelle qui décrit la structure algébrique particulière des états du système (multi-ensemble, arbre binaire, etc...) qui sont distribués selon cette même structure.

L'ensemble $R \subseteq L \times (T_{\Sigma,E}(X))^2$ est l'ensemble des pairs tel que le premier composant est une étiquette et le second est un pair des classes d'équivalence des termes modulo les équations E, avec $X = \{x1,\ldots,xn,\ldots\}$ un ensemble comptable et infini des variables. Les éléments de R s'appellent les règles conditionnelles de réécriture. Ils décrivent les transitions élémentaires dans un système concurrent. Chaque règle de réécriture correspond à une action pouvant se produire en concurrence avec d'autres actions.

Une théorie de réécriture est une description statique d'un système concurrent. Sa sémantique est définie par un modèle mathématique qui décrit son comportement.

Le langage de la logique de réécriture Maude [**29**], [**9**] est l'un des langages les plus puissants dans la spécification formelle, la programmation et la vérification des systèmes concurrents.

3.2. Maude

Maude [**29**] est un langage de spécification et de programmation basé sur la logique de réécriture. Maude est simple, expressif et performant. Maude offre peu de constructions syntaxiques et une sémantique bien définie.
Maude est un langage qui supporte facilement le prototypage rapide et représente un langage de programmation avec des performances compétitives. Dans le langage Maude, deux niveaux de spécification sont définis. Un premier niveau concerne la spécification du système tandis que le second est lié à la spécification des propriétés [**9**], [**10**].

4. Sémantique des RECATNets en termes de la logique de réécriture

Dans cette section nous présentons la méthode adoptée pour donner une sémantique de réécriture aux RECATNets. Nous avons pu voir dans la partie précédente que la logique de réécriture permet de décrire naturellement et intuitivement la sémantique des réseaux de Petri et des réseaux de Petri algébriques. En effet, cette logique permet de capturer correctement la structure algébrique de ce formalisme et les comportements atomiques et concurrents dans ces réseaux selon le mode de sémantique choisi (entrelacement ou vraie concurrence). En cela, la logique de réécriture offre un cadre sémantique puissant unifiant un grand nombre de classes de réseaux de Petri.

En décrivant la sémantique des ECATNets récursifs en termes de la logique de réécriture, on bénéficie :

- D'un cadre opérationnel où peut être spécifié le comportement de tout RECATNet.

Il existe actuellement des outils et des systèmes de réécritures qui supporte correctement le caractère opérationnel de cette logique tels que ELAN ou encore le système Maude ce qui nous permettra d'obtenir rapidement des prototypes exécutables des spécifications produites.

- D'un cadre théorique dans lequel toute réécriture correspond formellement à une preuve de déduction permettant ainsi d'établir des procédures d'analyse correctes pour les RECATNets.
- De plus, étant donné le caractère unificateur de la logique de réécriture, on peut facilement spécialiser la sémantique de réécriture donnée aux RECATNets pour décrire, en termes de la logique de réécriture, toutes sous-classes de ce modèle (comme les réseaux de Petri récursifs séquentiels) ou toutes superclasses de ce dernier. Puisque nous exprimons la Sémantique des RECATNets en termes de la logique de réécriture, chaque RECATNet RN est

décrit par une théorie de réécriture conditionnelle $\mathfrak{R}RN = (\Sigma RN, ERN, LRN, RRN)$ où la théorie équationnelle sous-jacente $(\Sigma RN, ERN)$ axiomatise la structure des états du RECATNet et où chaque transition est décrite par une règle de réécriture de RRN. Dans ce qui suit-on adopte la syntaxe de l'outil Maude pour présenter des théories équationnelles ou des théories de réécriture particulières décrivant la structure de l'espace des états des RECATNets ainsi que leurs transitions.

4.1. Spécification de la structure des états des RECATNets

On rappelle que l'état d'un RECATNet est un arbre dynamique de processus marqués Tr (un marquage étendu) où chaque nœud (processus) est associé à un marquage ECATNet ordinaire décrivant l'état interne de ce processus. La figure IV.2. Montre la structure générale d'un marquage étendu d'un RECATNet.

Pour décrire d'une manière simple et intuitive la structure algébrique d'un marquage étendu Tr d'un RECATNet RN par une théorie équationnelle, l'idée première est de spécifié chaque processus Th d'un tel arbre par un terme de la forme d'un triplet [MTh, tabs, ThreadChilds] de sorte Thread où :

- Le sous-terme MTh (de sorte Marking) représente le marquage interne de ce noeud.

Ce marquage est exprimé par des multi-ensembles sur des paires de la forme $\langle p_i, mp_i \rangle$ où p_i est une place de RN et le terme mp_i (de sorte Multiset) est un multi-ensemble de termes algébriques représentant les jetons résidants dans cette place (dans le processus Th).

- Le sous-terme t_{abs} de sorte Trans représente le nom de la transition abstraite dont le

Franchissement (dans le processus père) a donné naissance au processus Th. Notons que le processus racine n'est généré par le franchissement d'aucune transition abstraite. Dans un souci d'uniformité on représente le nom de la transition abstraite donnant naissance au processus racine par la constante nullTrans (de sorte Trans).

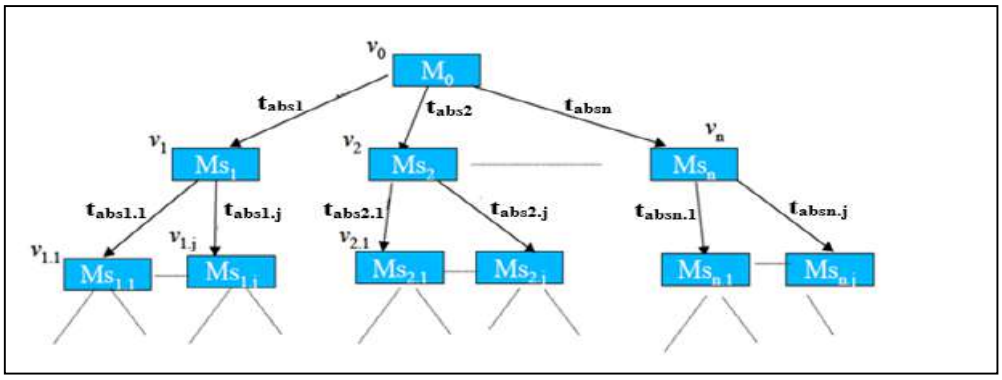


Figure IV.2. *Forme générale du marquage étendu associé à un RECATNet.*

➢ Le sous-terme *ThreadChilds* représente les processus générés par ce processus Th dans la configuration courante de l'arbre des processus Tr. La constante *nullThread* (de sorte Thread) représente le processus vide.

Par conséquent, le marquage étendu Tr (état) d'un RECATNet est décrit par un terme algébrique de la forme *récursive* générale suivante :

Tr = [M0, nullTrans,
[M_1, tabs1, [$M_{1.1}$, $tabs_{1.1}$, . . .] . . . [$M_{1.j}$, $tabs_{1.j}$, . . .]]
[M_2, $tabs_2$, [$M_{2.1}$, $tabs_{2.1}$ . . .]. . . [M2.j , $tabs_{2.j}$, . . .]] . . .
. . .
[Mn, tabsn, [Mn.1, tabsn.1, . . .] . . . [Mn.j, tabsn.j, . . .]].

Où :

- M0 est le marquage initial
- tabsi est la transition abstraite qui a donné naissance d'un processus de marquage Mi à partir de processus racine.
- Tabsi,j est la transition abstraite qu'a donné naissance d'un processus de marquage Mi,j à partir d'un processus de marquage Mi.

Ce terme algébrique peut être formalisé dans la logique de réécriture par la théorie équationnelle suivante :

❖ **Formalisation de sort PLACE-MARKING**

La formalisation suivante montre la spécification de type PLACE-MARKING qui représente le marquage d'une place.

```
fmod PLACE-MARKING is
sorts Place Multiset Place-marking .
op ems : ->Multiset .
------ The constant implementing the empty
marking ems
op _(+)_ : Multiset Multiset ->Multiset
[assoccomm id: ems] .
op<_;_> : Place Multiset -> Place-marking .
endfm.
```

❖ **Formalisation de sort MARKING**

La formalisation suivante montre la spécification de type MARKING qui représente le marquage ordinaire.

```
fmod MARKING is
including PLACE-MARKING .
sort Marking .
subsort Place-marking < Marking .
op em : -> Marking .
------ The constant implementing the empty
Marking em
op _(*)_ : Marking Marking -> Marking
[assoccomm id: em] .
Endfm.
```

❖ **Formalisation de sort THREAD**

La formalisation suivante montre la spécification de type THREAD qui représente le marquage étendu.

```
fmod THREAD is
including MARKING .
sorts Thread Trans TransAbs .
subsort TransAbs <  Trans .
op nullTrans : -> TransAbs .
op nullThread : -> Thread .
op [_,_,_]: Marking TransAbs Thread->Thread.
op _ _ : Thread Thread -> Thread
[assoccomm id: nullThread] .
Endfm
```

4.2. Règles de réécriture associées à un RECATNet

Chaque transition (élémentaire ou abstraite) ou étape de coupure d'un RECATNet est décrit par une règle de réécriture conditionnelle.

Afin de définir les règles de réécriture de chaque transition t, nous prenons les notations suivantes :

- p (resp.p') est une place pré (resp. post) de la transition t
- mp et mp' sont des variables de sort *Multiset*.
- Nbr est une fonction qui retourne le nombre de termes algébriques dans un *Multiset*.
- Cap est une fonction qui définit la capacité maximale d'une place en termes de nombre de termes algébriques.

4.2.1. Règles élémentaires

Une transition élémentaire $t \in T_{elt}$ est modélisée par une règle de réécriture (dite élémentaire) de la forme suivante :

crl[telt] : <p, mp (+) DT(p, telt)> (*)<p', mp'> =><p, mp> (*)<p', mp'(+)CT(p',telt)>
if (IC(p,telt) and TC(telt) and Nbr(mp'(+)CT(p',telt)) < Cap(p')) .

4.2.2. Règles abstraites

Une transition abstraite t est modélisée par une règle de réécriture (dite *abstraite*) de la forme suivante :

crl[t] : [<p, mp (+) DT(p, t)> , tabs, Th] =>[<p, mp> , tabs, Th [<p'', CT(p'',t) >, t, nullThread]]if (IC(p, t) and TC(t)) .

4.2.3. Règles de coupure

Une coupure τ est modélisée par une règle de réécriture (dite *copure*) rt ∈ Rcut de la forme suivante :

crl[τ] : <M (*) <p', mp'> , tabs, Th [M1 (*) <pfinal, mpfinal>, tabs1, mTh]> => [M (*) <p', mp' (+) ICT(p', tabs, i), tabs, Th>] if ($\forall i$ and Nbr(mp'(+) ICT(p',tabs,i) < Cap(p'))

5. ETUDE DE CAS (FIPA-BROKING-PROTOCOL)

5.1. Analyse par graphe d'accessibilité

Le schéma suivant montre le graphe d'accessibilité de notre exemple (FIPA-BROKING-PROTOCOL).

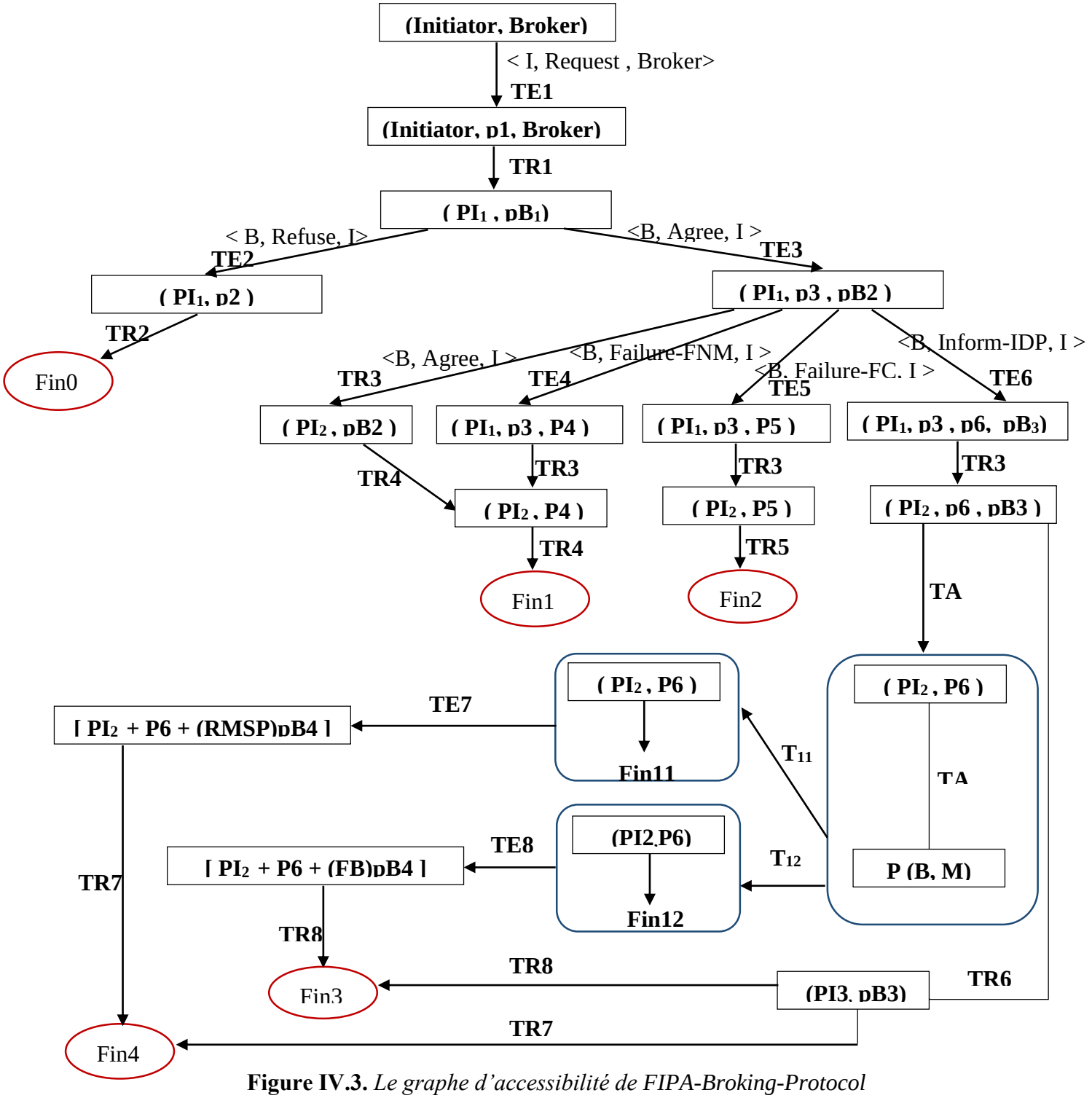


Figure IV.3. *Le graphe d'accessibilité de FIPA-Broking-Protocol*

A partir du graphe d'accessibilité de la figure IV.3, nous pouvons déduire certaines propriétés :

- **L'accessibilité**

C'est une propriété fondamentale pour un formalisme qui sert à montrer qu'un état donné est accessible à partir de l'état initial. Par exemple, l'état final (M= [Fin5]) est accessible à partir de l'état initiale (M0 = [Initiator, Broker]) avec la séquence de franchissement suivante :
[TE1 ; TR1 ; TE3 ; TE6 ; TR3 ; TR6 ; TR8].

- **Le Blocage**

Cette propriété a pour objectif de montrer qu'il n'y a pas de blocage i.e. le système s'arrête de s'évoluer. Selon la figure IV.3, il est clair que notre protocole ne peut atteindre une situation de blocage car toutes ses séquences d'exécution terminent par un état final.

5.1. Analyse par Maude

Maude est un langage qui permet de simuler les RECATNets en utilisant la logique de réécriture. En effet, Maude donne la possibilité de vérifier des propriétés des systèmes spécifiés en RECATNets.
Dans cette section, nous allons effectuer plusieurs tests :

- **Premier test**

Ce test permet de vérifier l'accessibilité de certains états selon des hypothèses bien définies. Durant cette phase de tests, nous allons utiliser la commande 'search' qui a la syntaxe suivante :

Search {in module :} term-src search type term-cible { such that condition} .

Cette commande permet de vérifier l'accessibilité de terme 'term-cible' à partir de terme 'term-src' sous certaines conditions. Les deux termes précédents représentent des états du système à analyser.

- **Hypothèse 1**

Le Broker refuse la requête envoyée par l'Initiator, car il a perdu le contact avec l'agent cible. Pour cela il envoie l'acte FC(Failure Connexion) à l'Initiator. Selon cette hypothèse, la syntaxe de la commande 'search' est la suivante :
search in RECATNET-FIPA : < Sender ; (s1 * Request) > (*) < Broker ; [b1 , Failure-FC , Failure] > =>+ < Fin2 ; (s1 * Failure-FC) > .
L'exécution de cette commande sous le système Maude donne le résultat suivant :

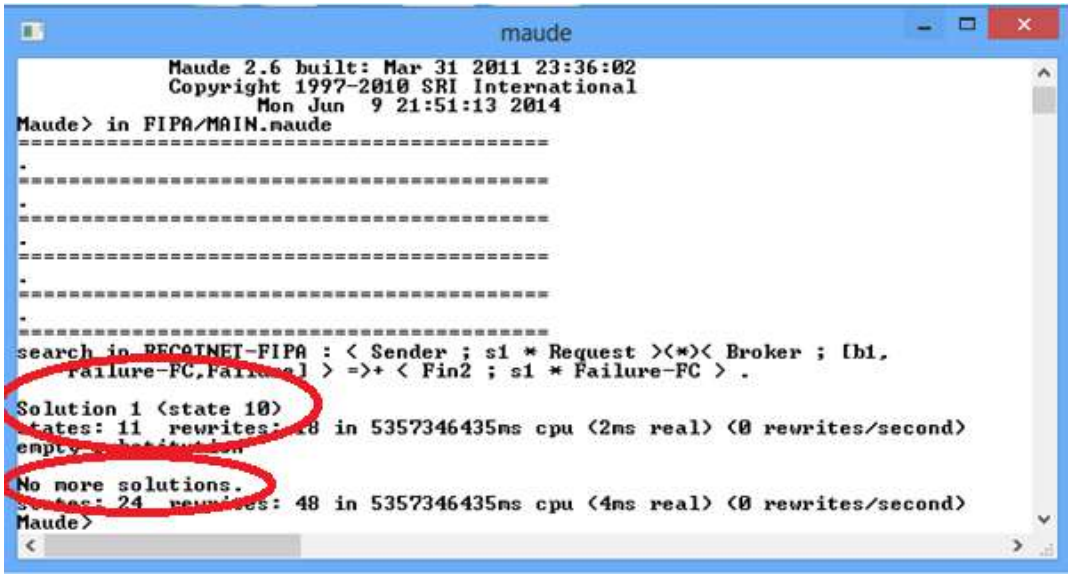


Figure IV.4. *Exécution de la commande 'search' selon l'hypothèse 1 du premier test*

D'après la figure IV.4, il est clair que le terme algébrique term-cible qui décrit l'état [Fin2 ; s1*Failure-FC] est bien accessible à partir du terme algébrique term-src qui décrit l'état initial < Sender ; (s1 * Request) > (*) < Broker ; [b1 , Failure-FC , Failure]. Cela signifié que notre protocole FIPA-Broking marche correctement selon cette hypothèse.

Dans le but de montrer le comportement de FIPA-Broking durant ce test, nous utilisons la commande 'show search graph'. La figure IV.5 montre une partie de comportement du FIPA-Broking par l'exécution de la commande 'show search graph':

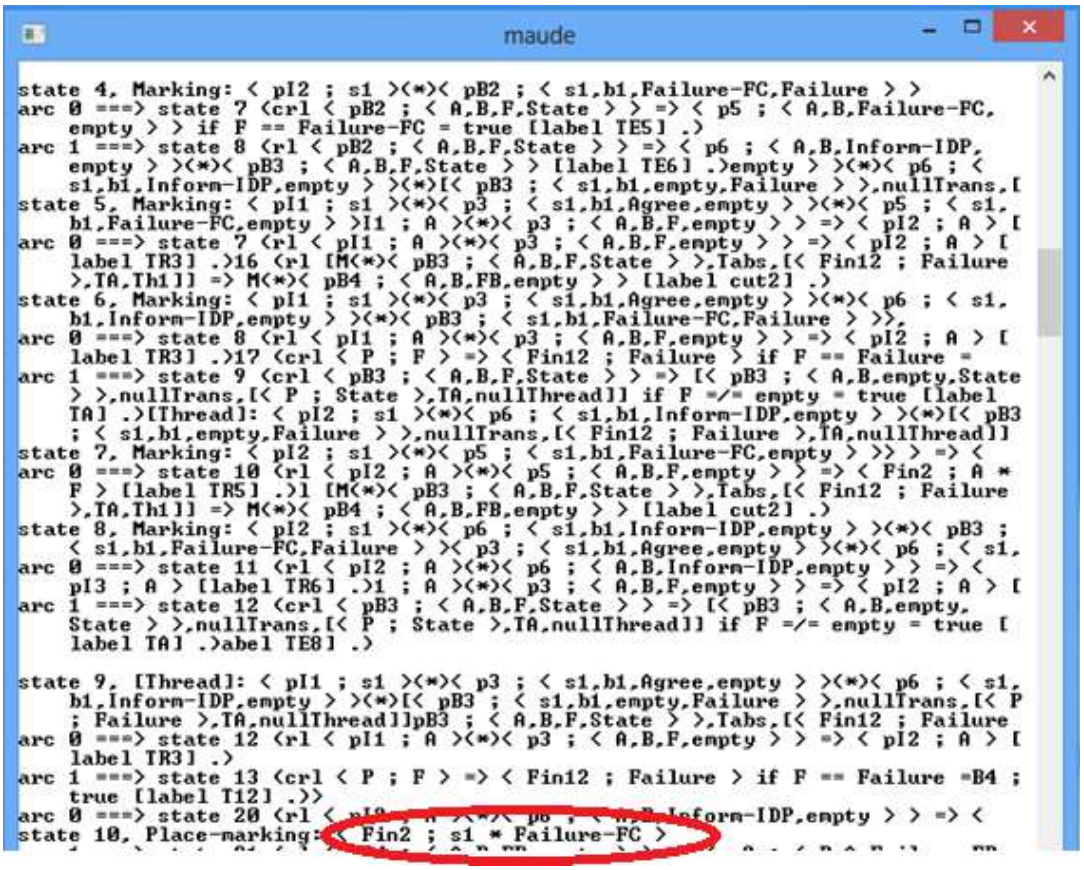


Figure IV.5. *Partie du graphe d'accessibilité de FIPA-Broking générée par la commande 'show search graph' selon l'hypothèse 1 du premier test*

- **Hypothèse 2**

Le Broker refuse la requête envoyée par l'Initiator, car il n'a pas pu trouver un agent cible et il envoie l'acte **FNM** (failure-no-match: no-potentiel agent) à l'agent Initiator.

Selon cette hypothèse, la syntaxe de la commande 'search' est la suivante :

search in RECATNET-FIPA : < Sender ; (s1 * Request) > (*) < Broker ; [b1 , Failure-FNM] > =>+ < Fin2 ; (s1 * Failure-FC) > .

L'exécution de cette commande nous donne le résultat suivant :

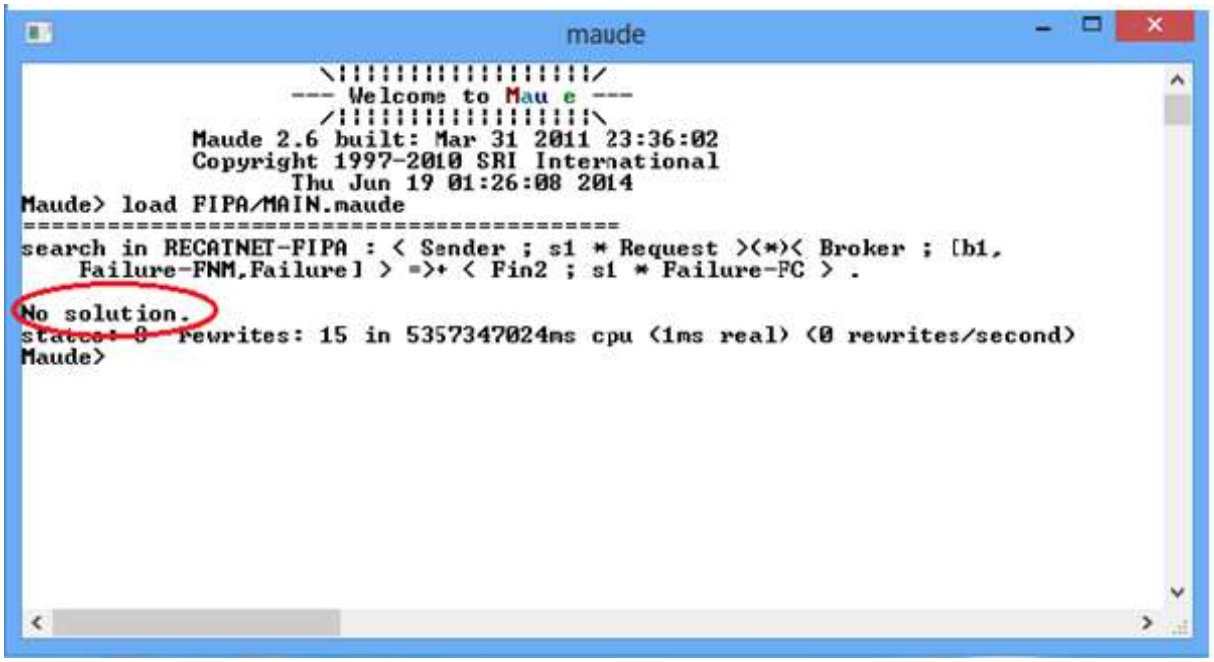


Figure IV.6. *Résultat de la commande 'search' selon l'hypothèse 2 du premier test*

D'après la figure IV.6 et selon l'hypothèse posée précédemment, le système FIPA-Broking ne peut atteindre le marquage final [Fin2 ; s1*Failure-FC]. Dans le but de connaitre le comportement de système FIPA-Broking, nous utilisons la commande ' show search graphe ' qui donne le résultat suivant :

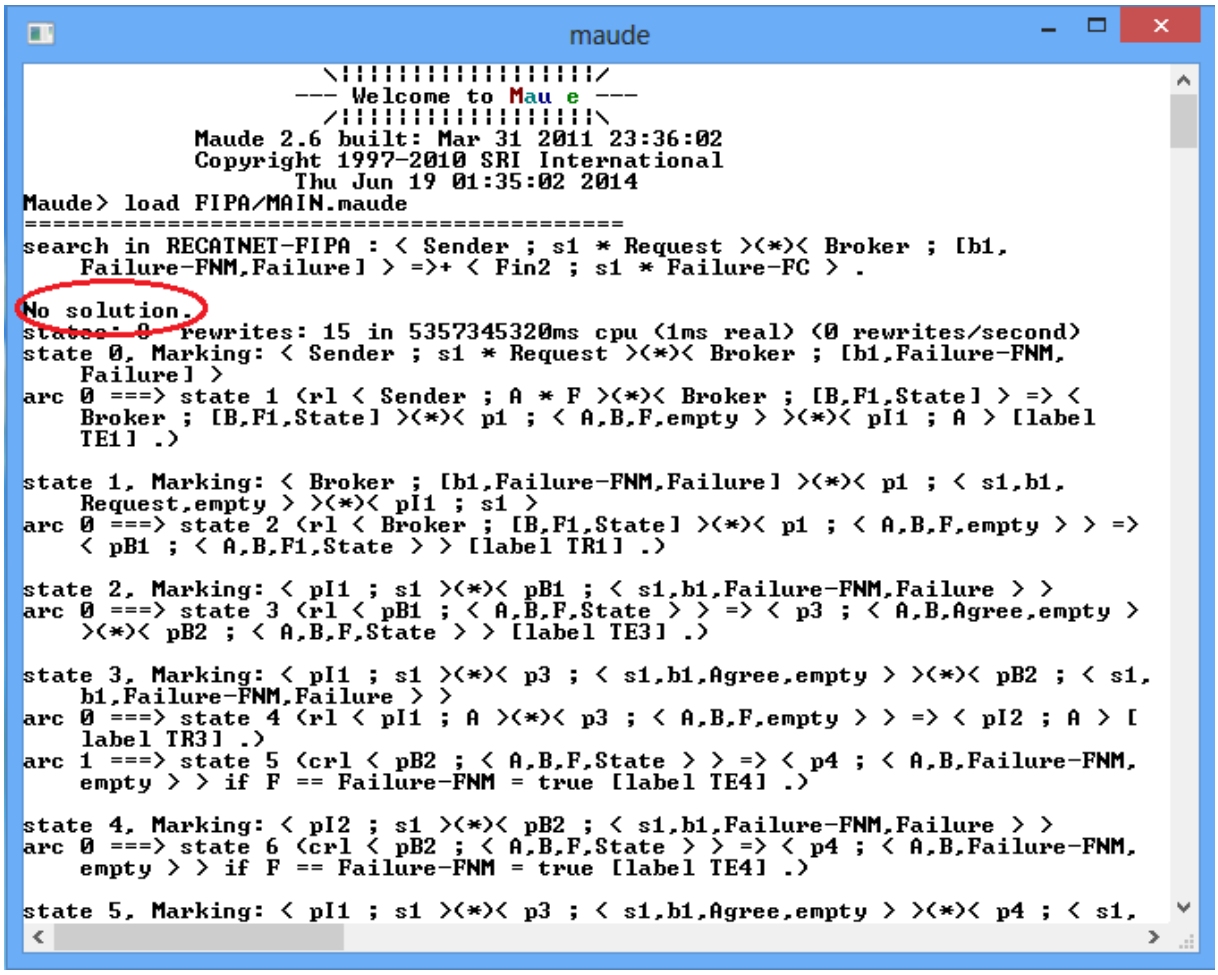


Figure IV.7. *Partie du graphe d'accessibilité de FIPA-Broking générée par la commande 'show search graph' selon l'hypothèse 2 du premier test*

D'après la figure IV.7, et selon l'hypothèse imposée précédemment le système atteint le marquage final [Fin3 ; s1*Failure-FNM] qui indique que le Broker n'as pas pu joindre l'agent cible.

Ce qui nous montre bien que le marquage [Fin3 ; s1*Failure-FNM] n'est pas accessible à partir de l'hypothèse posée.

Deuxième test

Dans cette partie nous allons utiliser le module 'model Checker' fournit par le système Maude. Ce module permet de vérifier les propriétés des systèmes exprimés en logique temporelle [**27**].

Nous allons effectuer deux vérifications selon les hypothèses suivantes :

- **Hypothèse 1**

Le Broker a trouvé un agent cible, pour cela il envoie l'acte informe-IDP à l'agent Initiator, et il reste en attente d'une réponse de la part de l'agent cible. Dans cette hypothèse la réponse est considérée comme Succes qui donne fin à l'interaction par un état final

< Fin3 ; (s1 * Reply-RMSP) >.

La commande associée :

reduce in RECATNET-CHECK : modelCheck (initialState , <> finalState). Où:

- initialState : < Sender ; (s1 * Request) > (*) < Broker ; [b1 , Inform-IDP , Succes] >.
- finalState: < Fin3 ; (s1 * Reply-RMSP) > .

La figure suivante montre le résultat d'exécution de cette commande :

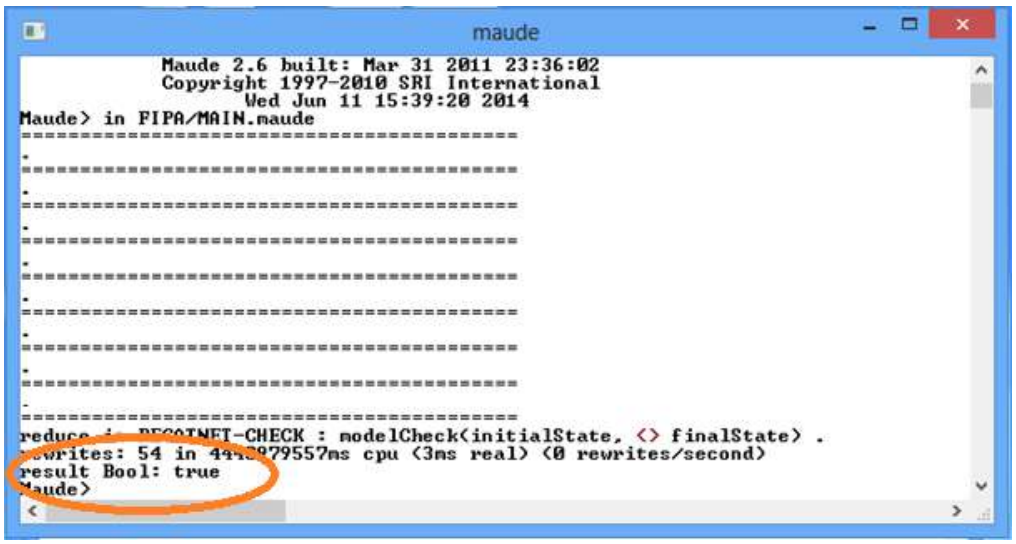


Figure IV.8. *Vérification par model-checher de système Maude.*

La figure IV.8 montre que l'hypothèse 1 est vraie i.e. lorsque l'agent cible envoie sa réponse 'Succes' l'agent Initiator reçoit l'acte Reply-RMSP de la part de l'agent Broker.

- **Hypothèse 2**

Le Broker a trouvé un agent cible, pour cela il envoie l'acte informe-IDP à l'agent Initiator, et il reste en attente d'une réponse de la part de l'agent cible. Dans cette hypothèse nous supposons que la réponse est Failure et l'interaction se termine par l'état final < Fin3 ; (s1 * Reply-RMSP) > .

La commande associée à cette hypothèse est la suivante :

reduce in RECATNET-CHECK : modelCheck(initialState , <> finalState1). Où:

- initialState : < Sender ; (s1 * Request) > (*) < Broker ; [b1 , Inform-IDP , Failure] >
- finalState: < Fin3 ; (s1 * Reply-RMSP) > .

La figure IV.9 montre le résultat d'exécution de la commande :



Figure IV.9. *Contre-exemple généré par le model-checker de système Maude.*

La figure IV.9 montre que l'hypothèse 2 est fausse i.e. lorsque l'agent cible envoie sa réponse 'Failure' l'agent Initiator reçoit l'acte Failure-FB de la part de l'agent Broker et l'interaction doit se terminer par un échec (< Fin4 ; (s1 * Failure-FB) >) voir le schéma.

6. Conclusion

Dans ce chapitre nous avons présenté notre approche d'implémentation de RECATNets sous Maude. Pour cela nous avons exprimé la sémantique de RECATNets en termes de la logique de réécriture. Ensuite, nous avons entamé la partie test pour garantir le bon fonctionnement de système étudié.

Plusieurs tests ont été effectués, selon des hypothèses bien définies en utilisant les différentes commandes fournies par le système Maude. En plus nous avons pu utiliser le model-cheker LTL afin de vérifier certaines propriétés temporelles.

Conclusion Générale

CONCLUSION GENERALE

L'objectif de ce travail était d'utiliser une méthode formelle afin de vérifier/analyser les protocoles de communications dans les systèmes multi-agents.

Pour pouvoir faire une telle vérification/analyse, nous avons d'abord étudié les systèmes multi-agents. L'étude de ce domaine, nous a permis de constater que l'ensemble des concepts sur lesquels repose ce domaine en font une approche comportementale très riche. Puis, nous avons présenté le formalisme des réseaux de Perti.

Le réseau de Perti est une méthode formelle fondée sur une théorie mathématique qui permet la modélisation et l'analyse des systèmes informatiques.

Après, nous avons proposé un ensemble de règles de transformation d'un modèle AUML vers un réseau de Perti de haut niveau appelé RECATNets. Ce dernier est un formalisme connu par sa puissance de modélisation/analyse des systèmes complexes.

Par la suite, nous avons exprimé le modèle de RECATNets en termes de la logique de réécriture. Ceci est dans le but d'utiliser le système Maude dans la vérification de modèle de RECATNets.

Dans le but de compléter ce travail, nous proposons d'automatiser notre approche de transformation en utilisant le concept de Méta-modèle.

Références Bibliographiques



Bibliographie



[1] **Alexandre VIOLLET**, « Un protocole entre agents pour l'alignement d'ontologie », université Joseph Fourier (UJF), 2003/2004. Mémoire de Master2 recherche mathématique et informatique.

[2] **V. Augusto**: *«Réseaux de Petri »* école Saint-Etienne .2012.Support de cours.

[3] **Imane, Melle BELLIFA.** Approche Multi-Agent pour la reconnaissance de Diabète. 03 Juillet 2011. Mémoire de fin d'étude.

[4] **K. Barkaoui, and A. Hicheuer**, Towards analysis of flexible and collaborative workflow using recursive ecatnets. In "CBP 2007", pages 206-217, LNCS 4928 (26-28 2007). Springer Berlin/Heidelberg, Benatallah, B.ter Hofstede, A., Paik. Article.

[5] **P. Borovansky, C.Kirchner, H.Kirchner, and P. Morea.Elan**, from a rewriting logic point of view. Théories. Computer. Sci. 285(2), 155-185(2002).Article

[6] **Achichi, Boubakeur.** Convention sociale dans les SMA Analyse, Modélisation Et Simulation. Octobre 2010.mémoire Master académique.

[7] **Jean-Pierre Briot**, Yves Demazeau, "Introduction aux agents : Principes et architecture des systèmes multi-agents", collection IC2, Hermès, 2001. Support de cours.

[8] **Daniel D. Corkill.** "Collaborating Software: Blackboard and Multi-Agent Systems & the Future", In Proceedings of the International Lisp Conference, New York, October 2003.Article.

[9] **M. Clavel**, F. Duran, S. Ecker, P. Lincoln, N. Marti-Oliet, J. Meseguer, C. Talcott. "The Maude 2.0 System". In Proc. Rewriting Techniques and Applications (RTA), Volume 2706 of LNCS, Spring-Verlag , p. 76-87., 2003.Article.

[10] **M. Clavel**, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, C. Talcott. "Maude Manual (Version2.2)". December 2005.

[11] **M. Diaz** : « *Les Réseaux de Petri–Modèles fondamentaux* » 2002, Hermes Science Publications : Paris. Thème de recherche.

[12] **Nicolas Esposito,** "Les agents mobiles–Une introduction", DASSAULT SYST`EMES –Recherche et nouvelles technologies, France, 2000..

[13] **Jacques Ferber,** "Les Systèmes Multi-agents: vers une intelligence collective", InterEdidion ,1995 Livre,

[14] **T. Finin**, R. Fritzson, D. McKay, and R. McEntire, "KQML as an agent Communication language", In Third international conference on information and knowledge management. ACM Press, November 1994.

[15] **El Fallah Segrouchni, S. Haddad**: A recursive model for distributing planning: In second international conference in multi-gent systems. Kyoto, Japan 1996.

[16] **Geneva, Switzerland,** « Foundation for Intelligent Physical Agents», 2003.Livre

[17] **Sahki, Hanine.** Vers une Architecture d'un Système de Dialogue Multi-agents Basé sur l'Argumentation Application à la négociation dans le domaine de e-commerce. 2008. Thèse de doctorat.

[18] **A. Hicheuer, K. Barkaoui, and.Boudiaf.** Modeling workflows with Recursive ecatnets. Symbolic and Numeric Algorithms for Scientific Computing, International Symposium on 0, 389-398 (2006).Publication

[19] **C. Hewitt**, "Viewing control structures as patterns of passing messages", Artificial Intelligence, 8: 323–364, 1977 Livre.

[20] **S. Hadad** and **D.Poitrenaud**. Recursive petri nets: Theory and application to discrete event systems. *Acta Inf.* 44(7), 463-508 (2007). Article

[21] **James Odell,** H. **Van Dyke Parunak**, Bernhard Bauer, « Extension de UML pour des systèmes basés agents »,2000 .Livre

[22] **Tarek Jarraya,** Réutilisation des protocoles d'interaction et Démarche orientée modèles pour le développement multi-agents, Thèse de Doctorat en informatique, l'Université de Reims, 2006.Livre

[23] **N. R. Jennings and M. Wooldridge,** "Applications of Intelligent Agents", Queen Mary & Westfield College, University of London, 1998.Livre

[24] **Jennings N, Sycara K., Wooldridge M,** "A Roadmap of Agent Research and evelopment"in Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers,Boston,1998.Mémoire de magister en informatique.

[25] **Y. Labrou and T. Finin**, "A Proposal for a new KQML Specification", rapport technique TR CS-97-03, C.S. and E.E. dep., university of Maryland Baltimore County, février 1997.

[26] **Marjorie LE Bars**," Un Simulateur Multi-Agent pour l'Aide à la Décision d'un Collectif: Application à la Gestion d'une Ressource Limitée Agro-environnementale", Thèse de Doctorat, université paris ix-France, 2002

[27] **Z. Manna**, A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems, Specification. *Springer, 1991.Livre.*

[28] **J. Meseguer**, "Conditional Rewriting Logic as a Unified Model of concurrency". Theoretical Computer Science, 1992.Livre.

[29] **J. Meseguer.** "Rewriting logic and Maude: a Wide-Spectrum Semantic Framework for Object-based Distributed Systems". In S. Smith and C.L. Talcott, editors, Formal Methods for Open Object-based Distributed Systems, (FMOODS'2000), p. 89-117. Kluwer, 2000.Livre.

[30] **Mickael Albert,** « Elaboration d'une passerelle AUML vers Jade », Ecole Polytechnique de l'Université de Nantes, 2005.

[31] **A. Newell**, "The Knowledge Level. Artificial Intelligence", page 87-127, 1982.Livre.

[32] **H. Penny**, "Blackboard Systems: The Blackboard Model of Problem–solving and the Evolution of Blackboard Architectures", AI Magazine, **7** (2), pp. 38 - 53, 1986.Livre.

[33] **Stuart Russell & Peter Norvig, P,** "Artificial intelligence" A modern Approach. 2010 Livre.

[34] **R. Valette** : « *Les réseaux de Petri*». LAAS-CNRS Toulouse. Septembre 2002. Article.

[35] **Wooldridge M, Jennings N.R**. "Agents Theories, Architecture and Language: A Survey, In: Intelligent Agents, ECAI'94, M. Wooldridge and N.R. Jennings (Eds.) LNAI 890, Springer Verlag, pp 1-32, 1994. Article.

[36] **Franco Zambonelli, Nicholas R. Jennings, Michael Wooldridge**, "Developing Multi-agent Systems: The Gaia Methodology", University di Modena e Reggio Emilia, 2003.Article

[37] **Zeineb Graja,** « méthode formelle: vers une approche dirigée par les modèles pour la conception d'applications à base d'agents », université de Sfax faculté des sciences économiques, 2009.mémoire de master.

[38] http://www.fipa.org/repository.php.

Annexe

Annexe A

I. Logique de Réécriture

Dans la logique de réécriture, chaque système concurrent, est représenté par une théorie de réécriture Â = (Σ, E, L, R) Sa structure statique est décrite par la signature (Σ, E), tandis que sa structure dynamique est décrite par les règles de réécriture R. Une conséquence intéressante des définitions de la logique de réécriture est qu'une théorie Â = (Σ, E, L, R) devient une spécification exécutable du système concurrent qu'elle formalise.

II. Maude

Maude [29] est un langage de spécification et de programmation basé sur la logique de réécriture. Maude est simple, expressif et performant. Maude offre peu de constructions syntaxiques et une sémantique bien définie

- **Niveau de Spécification de Système**

Ce niveau est basé sur la théorie de réécriture. Il est principalement décrit par les modules systèmes. Pour une bonne description modulaire, trois types de modules sont définis dans Maude. Les modules *fonctionnels* permettent de définir les types de données. Les modules *systèmes* permettent de définir le comportement dynamique d'un système.
Enfin, les modules *orienté-objet* qui peuvent, en fait, être réduits à des modules systèmes offrent explicitement les avantages du paradigme objet. Dans le cadre de notre travail, nous faisons appel uniquement à des modules fonctionnels et systèmes, alors nous écartons la présentation des modules orienté-objet.

- **Modules Fonctionnels.**

Les modules fonctionnels définissent les types de données et les opérations qui leur sont applicables en termes de la théorie des équations.
Voici un exemple du module fonctionnel :

```
fmod NAT is
sort Nat . sort PositiveNat .
subsort PositiveNat<Nat .
op 0 : ->Nat .
op S_ : Nat ->Nat .
op _+_ : Nat Nat -> Nat [comm] . *** N + M = M + N
vars N M : Nat .
var P : PositiveNat .
cmb P : PositiveNat if P =/= 0 . *** axiome d'adhésion indiquant que P est de type PositiveNat
*** si P est différent de 0
eq N + 0 = N . *** N + 0 et N dénotent le même terme (même classe d'équivalence) qui est
[N]
eq (S N) + (S M) = S S (N + M) .
endfm
```

- **Modules Systèmes.**

Les modules systèmes permettent de définir le comportement dynamique d'un système.. Un exemple simple montrant la puissance de ce type de module est donné dans ce qui suit :

```
mod NAT is
extending NAT .
op _? _ : Nat Nat ->Nat .
vars N M : Nat .
rl [R1] : N ? M =>N .
rl [R2] : N ? M =>M .
endm
```

L'opérateur ? modélise le choix aléatoire entre deux entiers.

- **Niveau de Spécification de Propriété**

Ce niveau de spécification définit les propriétés du système à vérifier. Le système est bien sûr décrit à l'aide d'un module système. En évaluant l'ensemble des états accessibles à partir d'un état initial, le Model Checking permet de vérifier une propriété donnée dans un état ou un ensemble d'états. La propriété est exprimée dans une logique temporelle LTL (Linear Temporel Logic) ou BTL (Branching Temporel Logic).

II.1. Exécution de Maude

Nous donnons une vue générale de l'exécution d'une session de Maude sous Windows pour mettre le lecteur dans le contexte. Nous pouvons commencer une session avec Maude en appelant le fichier de commande MS-DOS (même chose pour les autres plates-formes). Nous avons trois fenêtres, mais la principale à la forme qui se trouve dans la figure 1. Le système Maude est maintenant prêt pour accepter des commandes ou des modules. Durant une session l'utilisateur interagit avec le système en saisissant sa requête au 'Maude prompt'. Par exemple, si on veut quitter Maude Maude>quit 'q' peut être utilisé comme abréviation pour la commande 'quit'. On peut aussi saisir des modules et utiliser autres commandes. Ce n'est pas vraiment pratique de saisir un module dans le 'prompt', plutôt l'utilisateur peut écrire un ou plusieurs modules dans un fichier texte et faire entrer le fichier dans le système en faisant appel à la commande 'in' ou la commande 'load'. Supposant que le fichier s'appelle my-nat.maude contenant le module NAT décrit au-dessus, alors nous pouvons l'introduire au système en faisant la commande suivante :

Maude> in my-nat.maude

Après entrer le module NAT nous pouvons par exemple, réduire le terme s szero + s sszero (qui correspond à 2 + 3 dans la notation Peano) dans de tel module. Le lancement et la validation de la commande 'reduce' retourne un résultat comme suit :

Maude> reduce in NAT : s s zero + s ss zero .

reduce in NAT : s s zero + s ss zero .

rewrites: 3 in 0ms cpu (0ms real) (~ rewrites/second)

result Nat: s ssss zero

Ce n'est pas nécessaire de donner le nom du module explicitement dans lequel on réduit le terme. Toutes les commandes dont un module a besoin référent au module en cours par défaut, sinon il faut donner explicitement le nom du module. Le module en cours est en général le dernier introduit ou utilisé, ou bien nous pouvons utiliser la commande 'select' pour sélectionner un module pour être le module en cours.

Maude> reduce s s zero + s sszero .

reduce in NAT : s s zero + s ss zero .

rewrites: 3 in 0ms cpu (0ms real) (~ rewrites/second)

result Nat: s ssss zero

En général, les termes sont plus longs que ce simple exemple. Il est préférable d'écrire la commande dans le fichier contenant le module. La requête doit être introduite après le module, mais à l'extérieur de tout module.

Maintenant, nous introduisons quelques commandes de Maude que nous avons utilisé beaucoup pendant ce travail. Il

s'agit de 'reduce', 'rewrite', 'continue' et 'search'.

Maude possède une librairie standard des modules prédéfinis qui, par défaut, sont chargés par le système

automatiquement au début de chaque session. Chacun de ses modules prédéfinis peut être importé par un autre module défini par l'utilisateur. Ces modules prédéfinis se trouvent dans des fichiers au même répertoire que le fichier exécutable de Maude. Parmi ces fichiers, on trouve prelude.maude et model-checker.maude. En fait, le premier fichier contient plusieurs modules prédéfinis. Nous pouvons trouver comme exemples de modules prédéfinis dans ce fichier :

BOOL, STRING et NAT. Ces modules déclarent les sortes et les opérations pour manipuler respectivement les valeurs booléennes, les chaînes de caractères et les nombres naturels. Le fichier model-checker.maude contient des modules prédéfinis déjà introduits dans la section 3.2. Ces modules contiennent les outils nécessaires pour l'utilisation du Model Checker de Maude.

- **Commande ‘reduce’**

Cette commande permet au terme désigné ‘term’ d’être réduit par le biais des équations et les axiomes d’adhésion dans le module donné ‘module’. ‘reduce’ peut être abrévié à ‘red’. Si la clause ‘in’ est omise, le système prend en considération le module en cours :

reduce {in module :} term .

- **Commande ‘search’**

La commande search performe une recherche dans les calculs de la commande ‘rewrite’. Cette recherche débute du terme ‘subject’ jusqu’à un état final qui correspond à ‘pattern’ et qui satisfait une condition optionnelle :

search {[bound]} {in module :} subject searchtype pattern { such that condition} .

Les valeurs possibles pour ‘searchtype’ sont :

=>1 un pas de calcul

=>+ un ou plusieurs pas de calcul

=>* zéro ou plusieurs pas de calcul

=>! Seulement les états finaux sous forme canonique sont permis

Le nombre maximal des solutions trouvées est borné mais il peut s’étendre à l’infini. Pour afficher le graphe généré par ‘search’ il faut saisir la requête suivante :

show search graph .

- **Module RECATNET-CHECK**

```
mod RECATNET-CHECK is
protecting RECATNET-PREDS .
including MODEL-CHECKER .
including LTL-SIMPLIFIER . *** optional

var M : Marking .
ops initialState1  initialState finalState finalState1 : -> Thread .

eq initialState = < Sender ; ( s1 * Request ) > (*) < Broker ; [ b1 , Inform-IDP , Succes ] > .
eq initialState = < Sender ; ( s1 * Request ) > (*) < Broker ; [ b1 , Inform-IDP , Failure ] > .
endm
```

❖ **Module RECATNET-PREDS**

```
mod RECATNET-PREDS is
protecting RECATNET-FIPA .
including SATISFACTION .
subsort Thread < State .
ops finalState finalState1 : -> Prop .
var T : Thread .
var M : Marking .
eq < Fin3 ; ( s1 * Reply-RMSP ) > (*) M |= finalState = true .
eq < Fin4 ; ( s1 * Failure-FB ) > (*) M |= finalState1 = true .
endm
```

❖ **Module MULTISET-OPERATION**

```
fmod MULTISET-OPERATION is
****including NATURAL .
pr NAT .
including PLACE-MARKING .
sort CTLToken .
subsort CTLToken < Multiset .
vars mts mts1 mts2 : Multiset .
vars t1 t2 : CTLToken .
------ Operations on multi-sets ------------------
op _Inclu_ : Multiset Multiset -> Bool .
op Nbr_ : Multiset -> Nat .
op _Supp_ : Multiset Multiset -> Multiset .
op _Inter_ : Multiset Multiset -> Multiset .
------ Difference ----------------------------------
eq mts1 Supp ems = mts1 .
eq ems Supp mts1 = ems .
eq (t1 (+) mts1) Supp (t1 (+) mts2)= mts1 Supp mts2 .
eq (t1 (+) mts1) Supp t1 = mts1 .
eq (t2 (+) mts1) Supp t1 = t2 (+) (mts1 Supp t1) [owise] .
------ Cardinality ---------------------------------
eq Nbr(ems) = 0 .
eq Nbr(t1) = s(0) .
eq Nbr(t1 (+) mts1) = s(Nbr(mts1)) .
------ Inclusion -----------------------------------
ceq mts1 Inclu ems = true if mts1 == ems .
ceq mts1 Inclu ems = false if mts1 =/= ems .
eq ems Inclu mts1 = true .
ceq t1 Inclu (t2 (+) mts1) = true if t1 == t2 .
ceq t1 Inclu (t2 (+) mts1) = t1 Inclu mts1 if t1 =/= t2 .
```

```
eq (t1 (+) mts1) Inclu (t1 (+) mts2) = mts1 Inclu mts2 .
eq (t1 (+) mts1) Inclu (t2 (+) mts2) = false [owise] .
------ Intersection ----------------------------------
eq mts1 Inter ems = ems .
eq (t1 (+) mts1) Inter (t1 (+) mts2) = t1 (+) (mts1 Inter mts2) .
eq (t1 (+) mts1) Inter (t2 (+) mts2) = mts1 Inter mts2 [owise] .
endfm
```

Annexe B

I. La logique LTL

Selon la logique LTL, l'évolution d'un système se réalise à l'aide d'une succession d'actions et chaque action fait évoluer le système d'un état vers un seul et unique état i.e. Il y'a un seul chemin pour évaluer une formule logique.

- **Définition (Syntaxe)**

Les formules de LTL sont données par la grammaire suivante :

LTL ∋ φ, ϕ ::= P | ¬φ | φ∨ϕ | **X** φ | **G** φ | **F** φ |φ **U** ϕ Où :

- P ϵ AP (ensemble des propositions atomiques)

- **(X ou O)** est l'opérateur prochain, **(G ou □)** l'opérateur toujours, **(F ou ◊)** l'opérateur finalement et **(U)** l'opérateur jusqu'à. Ces quatre opérateurs sont appelés des opérateurs temporels linéaires (i.e. exprimant des propriétés le long d'une exécution).

- **Definition (Sémantique)**

Une exécution $\sigma = s_0, s_1, s_2, \ldots$ satisfait une formule LTL φ qu'on note par $\sigma \models \varphi$, si les conditions suivantes sont satisfaites :

- $\sigma \models P$ ssi $P \in Lab(\sigma(0))$, //i.e. $\sigma(0)=s_0$ vérifie la propriété P,
- $\sigma \models \neg\varphi$ ssi $\neg(\sigma \models \varphi)$,
- $\sigma \models \varphi \vee \phi$ ssi $(\sigma \models \varphi)$ ou $(\sigma \models \phi)$,
- $\sigma \models \mathbf{X}\ \varphi$ ssi $\sigma^1 \models \varphi$,
- $\sigma \models \mathbf{G}\ \varphi$ ssi $\forall i, i \geq 0$ on a $\sigma^i \models \varphi$,
- $\sigma \models \mathbf{F}\ \varphi$ ssi $\exists i, i \geq 0$ tel que $\sigma^i \models \varphi$,
- $\sigma \models \varphi\ \mathbf{U}\ \phi$ ssi $\exists i, i \geq 0$ tel que $(\forall j<i$, on a $\sigma^j \models \varphi)$ et $(\sigma^i \models \phi)$.

Note : Les opérateurs **F** et **G** peuvent être exprimés en fonction des opérateurs **X** et **U** i.e.

- **F** φ ≡ True **U** φ,
- **G** φ ≡ ¬ **F** ¬φ,

II. Implémentation de Module LTL sous le système Maude

```
fmod LTL is
  protecting BOOL .
  sort Formula .

  *** primitive LTL operators
  ops True False : -> Formula [ctor format (g o)] .
  op ~_ : Formula -> Formula [ctor prec 53 format (r o d)] .
  op _/\_ : Formula Formula -> Formula [comm ctor gather (E e)
prec 55 format (d r o d)] .
  op _\/_ : Formula Formula -> Formula [comm ctor gather (E e)
prec 59 format (d r o d)] .
  op O_ : Formula -> Formula [ctor prec 53 format (r o d)] .
  op _U_ : Formula Formula -> Formula [ctor prec 63 format (d r o
d)] .
  op _R_ : Formula Formula -> Formula [ctor prec 63 format (d r o
d)] .
```

```
*** defined LTL operators

  op _->_ : Formula Formula -> Formula [gather (e E) prec 65
format (d r o d)] .
  op _<->_ : Formula Formula -> Formula [prec 65 format (d r o d)]
.
  op <>_ : Formula -> Formula [prec 53 format (r o d)] .
  op []_ : Formula -> Formula [prec 53 format (r d o d)] .
  op _W_ : Formula Formula -> Formula [prec 63 format (d r o d)] .
  op _|->_ : Formula Formula -> Formula [prec 63 format (d r o d)]
. *** leads-to
  op _=>_ : Formula Formula -> Formula [gather (e E) prec 65
format (d r o d)] .
  op _<=>_ : Formula Formula -> Formula [prec 65 format (d r o d)]
.

  vars f g : Formula .

  eq f -> g = ~ f \/ g .
  eq f <-> g = (f -> g) /\ (g -> f) .
  eq <> f = True U f .
  eq [] f = False R f .
  eq f W g = (f U g) \/ [] f .
  eq f |-> g = [](f -> (<> g)) .
  eq f => g = [] (f -> g) .
  eq f <=> g = [] (f <-> g) .
```

III. Model Checker

Le Model Checker de Maude vérifie si cette formule est valide (selon la nature de la formule et la procédure du Model Checker adoptée par le système Maude) dans cet état ou l'ensemble de tous les états accessibles depuis l'état initial. Si la formule n'est pas valide, un contre-exemple est affiché. Le contre-exemple concerne l'état dans lequel la formule n'est pas valide:

```
fmod MODEL-CHECKER is
Including SATISFACTION.
…
Op counterexample : TransitionListTransitionList -> ModelCheckResult [ctor] .
opmodelCheck : State Formula ~>ModelCheckResult .
…
Endfm
```