République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Cheikh Larbi Tbessi –Tébessa-
Faculté des Sciences et de la Technologie
Département Informatique
Ecole **D**octorale en **S**ciences et **T**echnologie de l'**I**nformation et de la **C**ommunication (INI)

N° D'ORDRE : …….
SERIE : ……..…

MEMOIRE DE MAGISTER
EN INFORMATIQUE

Option: Système d'Information et Connaissance

Formalisation des Besoins Fonctionnels des Systèmes Multi-Agents à l'aide de Maude

Présenté par : Mr HAMIDANE Fathi

Dirigé par :
- **Mme H.Belleili**
- **Mr. F.Mokhati**

Soutenu le ……/2009

Devant le Jury composé de:

Président:	Dr KAZAR Okba	**Maître de conférences**	Université de Biskra
Rapporteur:	Dr BELLEILI Habiba	**Maître de conférences**	Université de Annaba
Co-Rapporteur:	Dr MOKHATI Farid	**Maitre de conférences-B**	Université de Oum el Bouaghi
Examinateurs:			
	Dr SERIDI Hassina	**Maître de conférences**	Université de Annaba
	Dr MESLATI Djamel	**Maître de conférences**	Université de Annaba


بسم الله الرحمن الرحيم

À mes parents,
à Mr F.Mokhati et
Mme H.Belleili.

Remerciements

Je rends la grâce à Mon Dieu de m'avoir soutenu tout au long de cette thèse. Et de m'avoir dirigé vers les personnes ci-dessous pour m'aider à la mener à bonne fin.

Mes remerciements vont en direction de :

- Mon directeur de mémoire, Mme Habiba Belleili, Maitre de Conférences à l'Université d'Annaba. Je vous remercie Mme pour votre patience, vos précieux conseils, votre encouragement prodigué, et sur la bonne volonté.
- Mon co-directeur, Mr. Farid Mokhati, Maitre de Conférences-B à l'Université de Oum-Elbouaghi. Je le remercie pour son enthousiasme, son encouragement prodigué, ses conseils et les connaissances fruits de ses propres efforts, sa bonne volonté, sa patience et sa disponibilité permanente.

Je les remercie beaucoup pour leur soutien indéfectible.

Comme je tiens à remercier :

- Mr. Le président du jury Okba KEZAR Maitre de Conférences à l'université de Biskra, pour m'avoir fait l'honneur de présider le jury de cette thèse.
- Mme. Hassina SERIDI Maitre de Conférences à l'université de Annaba pour l'acceptation de juger ce travail.
- Mr. Djamel MESLATI pour avoir fait partie du jury de ce travail.
- Finalement, je remercie mes parents pour leurs soutiens tout au long de ce travail. Ainsi tous mes amis et tous qui ont contribué directement ou indirectement à l réussite de ce travail.

Résumé

Ce mémoire présente une approche systématique permettant de translater les besoins fonctionnels d'un système multi-agents décrits par les diagrammes de cas d'utilisation UML étendus et, les diagrammes de séquences AUML dans une spécification formelle Maude. Notre approche propose dans un premier temps d'étendre les cas d'utilisation UML en utilisant les stéréotypes UML pour tenir compte de spécificités de SMAs. Dans un second temps, nous associons à chaque cas d'utilisation un ou plusieurs diagrammes de séquences AUML réalisant les différents scénarios possibles de la fonctionnalité décrite par le cas d'utilisation proprement-dit. Une fois élaborés les différents diagrammes subissent une validation afin d'assurer la cohérence inter et intra modèles. Le langage Maude, basé sur la logique de réécriture, offre des bases formelles et solides pour la spécification et la programmation des systèmes concurrents. Les principales motivations de ce travail sont : (1) la formalisation des besoins fonctionnels d'un système multi-agents à l'aide de Maude, et (2) l'intégration de la validation formelle de la cohérence des modèles, dès la phase d'élicitation des besoins, dans un processus de développement des SMAs.

Mots Clés :

Besoins fonctionnels, Spécification formelle, Diagramme de cas d'utilisation, Agent UML, Logique de Réécriture, Maude.

Abstract

This work presents a systematic approach allowing the translation of the functional requirements of multi-agents system described by extended UML use case diagrams and, AUML sequence diagrams into a formal specification Maude. Our approach proposes in first time, to extend UML use case by using UML stereotypes for taking into account MAS' specificities. In second time, we associate to each use case, one or several AUML sequence diagrams realizing the different possible scenarios relative to such a use case. Once elaborated, the different diagrams undergo a validation to assure inter-and intra model coherence. The formal and object-oriented language Maude, based on rewriting logic, supports formal specification and programming of concurrent systems.The main motivations of this work are: (1) formalizing the functional requirements of multi-agents system by using Maude, and (2) integrating the formal validation of the coherence of the models, since the requirements elicitation phase, in a MAS development process.

Key words :

Functional requirements, Formal specification, Use case diagram, Agent UML, Rewriting logic, Maude.

الملخص

هذا العمل يطرح نهج منظم يسمح بتوجمة الاحتياجات الوظيفية لنظام متعدد الأعوان الممثلة بالرسوم البيانية لحالة الاستخدام الممتدة للغة النموذج الموحدة و, الرسوم البيانية للتسلسل لعون لغة النموذج الموحدة إلى مواصفات رسمية مكتوبة بلغة مود. نهجنا يسمح : أولا ، بتمديد الرسوم البيانية لحالة الاستخدام للغة النموذج الموحدة من خلال استخدام القوالب للغة النموذج الموحدة من أجل أن نأخذ في الاعتبار خصوصيات الأنظمة متعددة الأعوان. ثانيا، نضم لكل حالة استخدام رسم بياني تسلسلي واحد أو اكثر لعون لغة النموذج الموحدة التي تمثل مختلف السيناريوهات الممكنة لو ظيفة النظام الموصوفة عن طريق حالات الاستخدام المذكورة في السابق. عند إتمام الرسومات البيانية تتم عملية المصادقة لضمان الترابط المنطقي بين و داخل النموذج. لغة مود ترتكز عل منطق إعادة الكتابة التي تقدم قواعد رسمية و متينة لتحديد و برمجة الأنظمة المتنافسة. من الدوافع الرئيسية لهدا العمل هي : أولا إضفاء الطابع الرسمي للاحتياجات الوظيفية للأنظمة متعددة الأعوان بواسطة النظام مود , ثانيا : دمج المصادقة الرسمية للترابط المنطقي لمختلف النماذج من مرحلة إحصاء الاحتياجات في عملية تطوير الأنظمة متعددة الأعوان.

الكلمات الدالة :

الاحتياجات الوظيفية, مواصفات رسمية, رسم بياني لحالة استخدام, عون لغة النموذج الموحدة, منطق إعادة الكتابة, مود.

Sommaire Général

CHAPITRE III : Méthodologies de Développement des Systèmes Multi-Agents : Une Étude Comparative

Liste des Figures

Liste des Tables

INTRODUCTION GENERALE

I. Cadre et motivations

Les systèmes multi-agents (SMA) constituent un sous domaine de l'intelligence artificielle distribuée (IAD). Ces systèmes sont composés d'un ensemble d'entités appelés 'agents'. Ces derniers possèdent plusieurs caractéristiques, nous citons entre autres, la pro-activité, la réactivité, l'autonomie, et la sociabilité [Woo02]. Ces différentes caractéristiques font des SMA, des systèmes complexes, difficiles à maîtriser.

Plusieurs travaux de recherches ont été réalisés dans ce domaine et à tous les niveaux (spécification, conception, test, …etc). Bien que ces travaux aient apportés plusieurs éléments de réponse très importants à plusieurs problèmes, leurs aspects méthodologiques ne sont pas encore maîtrisés. A ce stade, plusieurs méthodologies (GAIA[Woo00], Prometheus [Pad02], DACS [Bus04], etc.) de développement des SMA ont été proposées dans la littérature afin de faciliter et d'aider les développeurs tout au long du processus du développement de leurs applications multi-agents.

Ces méthodologies ont certes apportés des réponses importantes au processus de développement des SMA, mais elles n'ont pas encore atteint un niveau de maturité élevé. Parmi les lacunes de ses méthodologies, nous citons l'absence de formalisation des besoins fonctionnels des SMA. La plupart de ces méthodologies utilisent dans leurs phases d'analyse, des spécifications informelles ou à la limite semi-formelle. La qualité du modèle d'analyse est d'une importance extrême pour le reste des phases du processus de développement. Sa validation formelle permet d'éviter une multitude de problèmes pouvant affecter la qualité du développement ainsi que sont coût [Dan07].

II. L'objectif et contribution

Dans l'objectif de profiter des avantages des approches formelles et celles semi-formelles, et de l'intégration de la validation formelle de la cohérence des modèles, dès la phase d'élicitation des besoins, dans un processus de développement des SMAs, nous proposons dans ce manuscrit une nouvelle approche de formalisation des besoins fonctionnels des SMA à l'aide de langage Maude.

Dans le cadre de notre approche, nous avons opté pour Maude [Cla96] pour les avantages qu'il procure. Basé sur une logique saine et complète dite la logique de réécriture [Mes92], Maude est un langage multi-paradigmes, il combine la programmation fonctionnelle et la programmation orientée objet. Par ailleurs, Maude est très puissant en termes de spécification, validation et vérification des systèmes concurrents, ce qui en fait un bon candidat pour la spécification et la validation des systèmes multi-agents.

L'approche proposée est structurée en trois étapes principales. La première étape consiste à la description des besoins fonctionnels d'un système multi-agents en utilisant les diagrammes de cas d'utilisation UML étendus, en utilisant les stéréotypes UML. Après avoir établi le diagramme de cas d'utilisation, nous passons, à la réalisation de différents cas d'utilisation en se basant sur les diagrammes de séquence AUML pour modéliser les différents scénarios de chaque cas d'utilisation. Dans la deuxième étape, les diagrammes considérés doivent subir une validation inter-modèles afin de vérifier la consistance du système. La troisième étape consiste à la génération d'une description Maude à partir des diagrammes utilisés.

III. Organisation du mémoire

Le reste de ce manuscrit est organisé en quatre chapitres :

1. Le premier chapitre présente les notions de base des systèmes multi-agents ainsi que les méthodologies d'analyse et de conception des SMAs.

2. Dans le deuxième chapitre nous présentons la logique de réécriture et le langage MAUDE.

3. Une étude comparative entre les différentes méthodologies présentées dans le premier chapitre est présentée dans le troisième chapitre.

4. Le quatrième chapitre présente l'approche que nous proposons pour la description formelle des besoins fonctionnels des systèmes multi-agents.

CHAPITRE I :

Les Systèmes Multi-Agents

Chapitre I :

Les Systèmes Multi-Agents (SMA)

I. Introduction

Les Systèmes Multi-Agents (SMA) sont des systèmes informatiques distribués, issus du domaine de l'Intelligence Artificielle Distribués (IAD).Ils représentent une nouvelle façon d'analyser, de concevoir et d'implanter des systèmes informatique complexes. Ils sont composés d'entités informatiques souvent intelligentes qui interagissent entre elles. L'interaction entre agents est généralement organisée sous forme de protocoles d'interaction et ce dans l'objectif de garantir une bonne coopération et/ou une bonne coordination d'actions [Cla01].

La technologie agent est devenue une solution importante pour la plupart des procédés industriels qui sont souvent complexes. En effet, les spécificités du paradigme agent ainsi que les méthodologies et outils qui sont développés à base d'agent ont facilité sa pénétration dans l'industrie.

Malgré les différentes théories d'agent développées, des langages, des architectures et la réussite des applications basées-agent, la plupart de travaux réalisés dans la littérature sont développés sans tenir compte de méthodologies de développement des SMA. L'utilisant de méthodologies orientées-agent décharge les développeurs de plusieurs tâches fastidieuses et les orientent au bon développement de leurs applications et ce tout au long de leur cycle de vie [Igl99].

Nous présentons dans ce chapitre les concepts de bases des systèmes multi-agents, ainsi que les méthodologies de développement des SMA.

II. Concepts et définitions

Dans cette partie nous présentons quelques des notions de base relatives au paradigme agent.

II.1 Agent

Il n'existe pas à l'heure actuelle, dans la littérature scientifique, de consensus sur la définition d'un agent tant les disciplines dans lesquelles il est fait référence sont nombreuses. Cependant, la définition proposée dans [Jen98] est largement reprise au sein de la communauté des systèmes multi-agents. Cette définition est la suivante : *« Un agent est un système informatique, situé dans un environnement, et qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu.»*. Plusieurs chercheurs ont proposés d'autre définition au niveau du concept agent, nous citons entre autres:

a. D'après [Woo00a] un agent est un système informatique encapsulé situé dans un environnement dans lequel il est capable d'effectuer une action flexible et autonome, compatible aux objectifs de la conception. Les agents sont :

- des entités clairement identifiables de résolution de problèmes avec des bornes et des interfaces bien définies ;
- situés dans un environnement particulier ; ils reçoivent des entrées liées aux états de cet environnement par des capteurs et agissent sur cet environnement par des émetteurs ;
- destinés à atteindre un objectif spécifique ;
- autonomes et responsables de leur comportement ;
- capables d'adopter un comportement flexible pour résoudre des problèmes selon les objectifs de la conception; ils sont réactifs (capables de s'adapter aux changements d'état de leur environnement) et proactifs (capables de prendre des initiatives) ;
- capables dans un univers multi-agents, de communiquer, coopérer, se coordonner, négocier les uns avec les autres.

b. On appelle agent une entité réelle ou abstraite qui est capable d'agir sur elle-même et sur son environnement, qui dispose d'une représentation partielle de cet environnement, qui, dans un univers multi-agents, peut communiquer avec d'autres agents et dont le comportement est la conséquence de ses observations, de sa connaissance et des interactions avec les autres agents [Fer95].

c. Un agent est une entité qui perçoit son environnement et agisse sur celui-ci [Rus97].

d. Un agent est une entité qui fonctionne continuellement et de manière autonome dans un environnement où d'autres processus se déroulent et d'autres agents existent [Sho93].

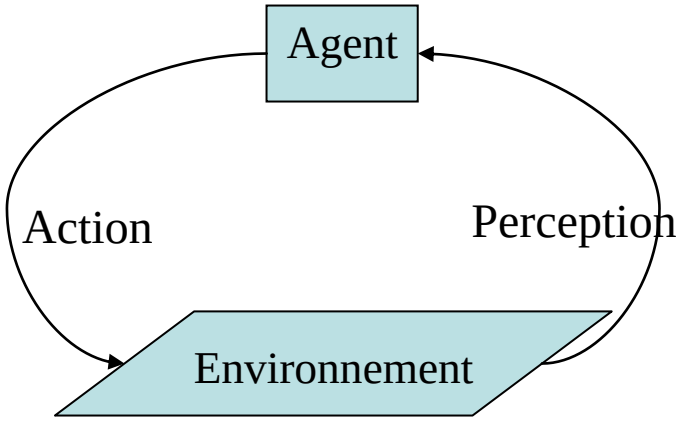

Figure 1: Description d'un Agent

1. Les caractéristiques des agents

D'après [Woo00b, Jen98] un agent est une entité qui possède les caractéristiques suivantes :

a. L'Autonomie : l'agent est capable d'agir sans l'intervention d'un tiers (humain ou agent) et contrôle ses propres actions ainsi que son état interne.

b. La Flexibilité : regroupe trois choses **:**

- **Réactivité :** Il est capable de percevoir des changements dans son environnement et d'y répondre en temps opportun afin de satisfaire leurs objectifs ;
- **Pro-activité :** L'agent doit exhiber un comportement proactif et opportuniste, tout en étant capable de prendre l'initiative au "bon" moment ;

- **Sociabilité :** l'agent doit être capable d'interagir avec les autres agents (logiciels et humains) quand la situation l'exige afin de compléter ses tâches ou aider ces agents à accomplir les leurs

c. Situé : l'agent est capable d'agir sur son environnement à partir des entrées sensorielles qu'il reçoit de ce même environnement. Exemples: systèmes de contrôle de processus, systèmes embarqués, etc ;

d. La mobilité : capacité de se déplacer d'un système vers autre a travers le réseau afin de se rapprocher des ressources qu'il utilise.

e. La sincérité : un agent ne doit pas communiquer de fausses informations**.**

f. L'intelligence : un agent intelligent doit faire preuve de :

- **Rationalité** *:* capacité d'un agent a sélectionné les meilleures actions qui lui permettent d'atteindre un de ses buts.
- **Intentionnalité** *:* la volonté d'un agent d'atteindre un but ou d'effectuer une action.
- ***Adaptabilité** :* un agent adaptatif est un agent qui possède un haut niveau de flexibilité.

2. Déterminant d'un Agent [Lab93]

C'est l'ensemble nécessaire et suffisant de ses caractéristiques :

- **Structurelles :** déterminant l'ensemble de ses composants.
- **Environnementales :** liées a la représentation que se fait l'agent de son environnement et de lui-même.
- **Comportementales :** contraignent l'ensemble de ses comportements, en accord avec les caractéristiques environnementales.

3. Architecture d'Agent

On peut dire que l'architecture d'un agent est une description de son organisation interne : les données et les connaissances de l'agent, les opérations qui peuvent être effectuées sur ses composants et le flux de contrôle des opérations [Flo02].

Le choix d'une architecture ou d'une autre est, bien sûr, lié à la structure conceptuelle de l'agent.

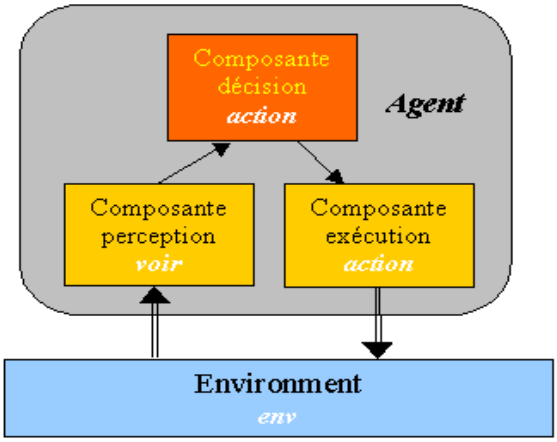

Figure 2: Structure générale d'un agent.

- ***voir* : E -> P** est la fonction qui décrive la capacité d'observation de l'environnement, où E est l'ensemble d'états de l'environnement et P est l'ensemble des perceptions de l'agent ;
- ***action* : P -> A** est la fonction qui représente le processus de décision de l'agent : elle détermine quelle action A (A étant l'ensemble des actions disponibles pour l'agent) l'agent choisit en fonction de la perception P qu'il a sur son environnement ;
- ***env* : E x A ->P(E)** est la fonction qui décrit l'évolution de l'environnement ; pour un état E et une action A de l'agent, l'environnement peut modifier son état dans un des états d'un sous-ensemble $E_{e,a}$ où P(E) est l'ensemble qui contient le sous ensemble $E_{e,a}$.

A. Architecture BDI (Belief Desire Intention)

L'architecture BDI a été développée par les chercheurs [Bra87, Bra88, Geo87, Rao91a, Rao91b, Rao92, Sin94]. L'agent qui a une architecture BDI, est un agent généralement représenté par un "état mental" ayant les attitudes mentales suivantes:

- **La croyance :** les croyances d'un agent sont les informations que l'agent possède sur l'environnement et sur d'autres agents qui existent dans le même environnement (Ce que l'agent connaît de son environnement).

- **Le Désir :** les désirs d'un agent représentent les états de l'environnement, et parfois de lui-même, que l'agent aimerait voir réalisés (les états possibles envers lesquels l'agent peut vouloir s'engager).
- **L'intention :** les intentions d'un agent sont les désirs que l'agent a décidé d'accomplir ou les actions qu'il a décidé de faire pour accomplir ses désirs (Les états envers lesquels l'agent s'est engagé, et envers lesquels il a engagé des ressources).

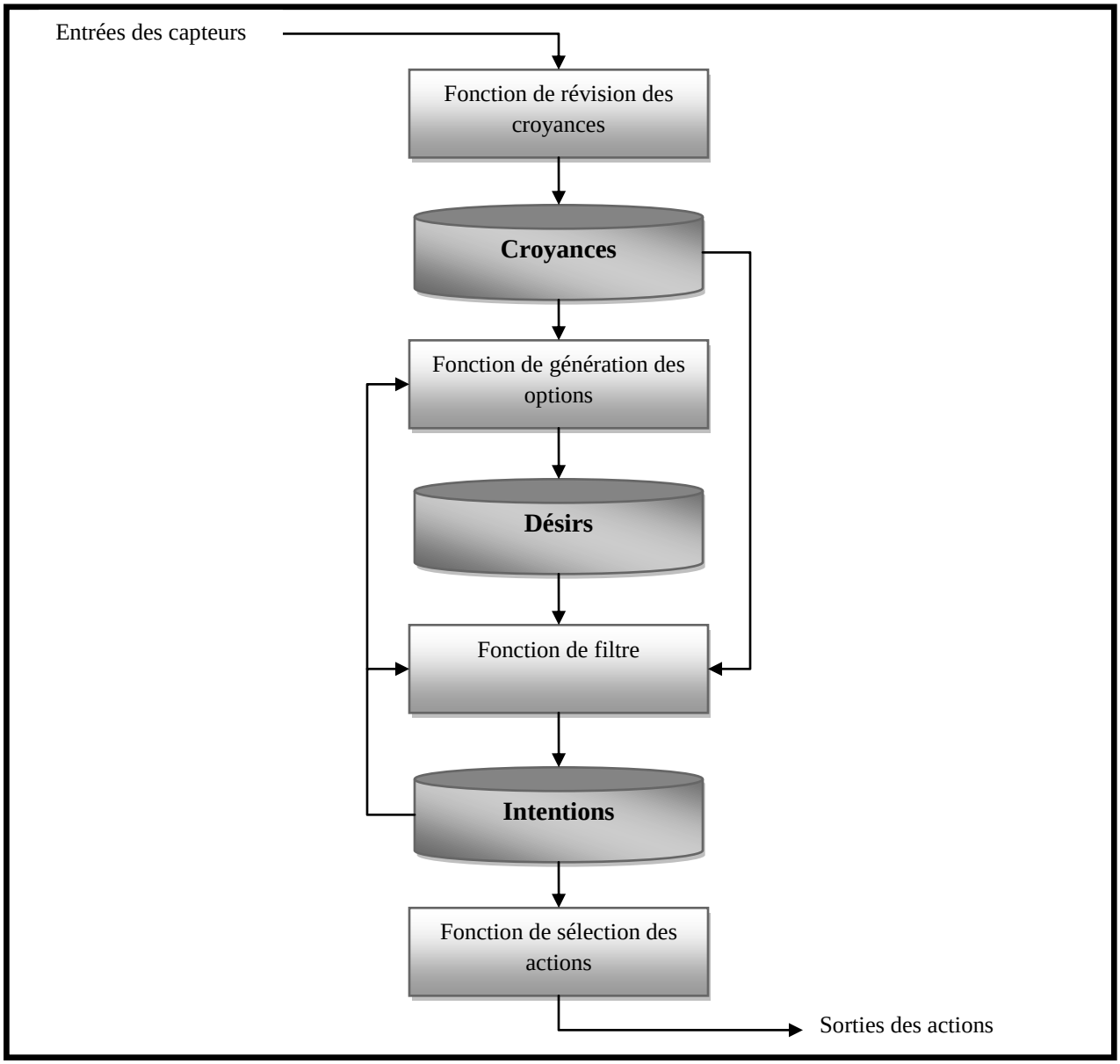

Figure 3: Architecture BDI d'un agent

B. Architecture Hybride

Dans ce cas, un agent est composé de plusieurs couches, arrangées selon une hiérarchie, la plupart des architectures considèrent que trois couches suffisent amplement. Ainsi, au plus bas niveau de l'architecture, on retrouve habituellement une couche purement réactive, qui prend ses décisions en se basant sur des données brutes en provenance des senseurs. La couche intermédiaire fait abstraction des données brutes et travaille plutôt avec une vision qui se situe au niveau des connaissances de l'environnement. Finalement, la couche supérieure se charge des aspects sociaux de l'environnement, c'est à dire du raisonnement tenant compte des autres agents [Cha02].

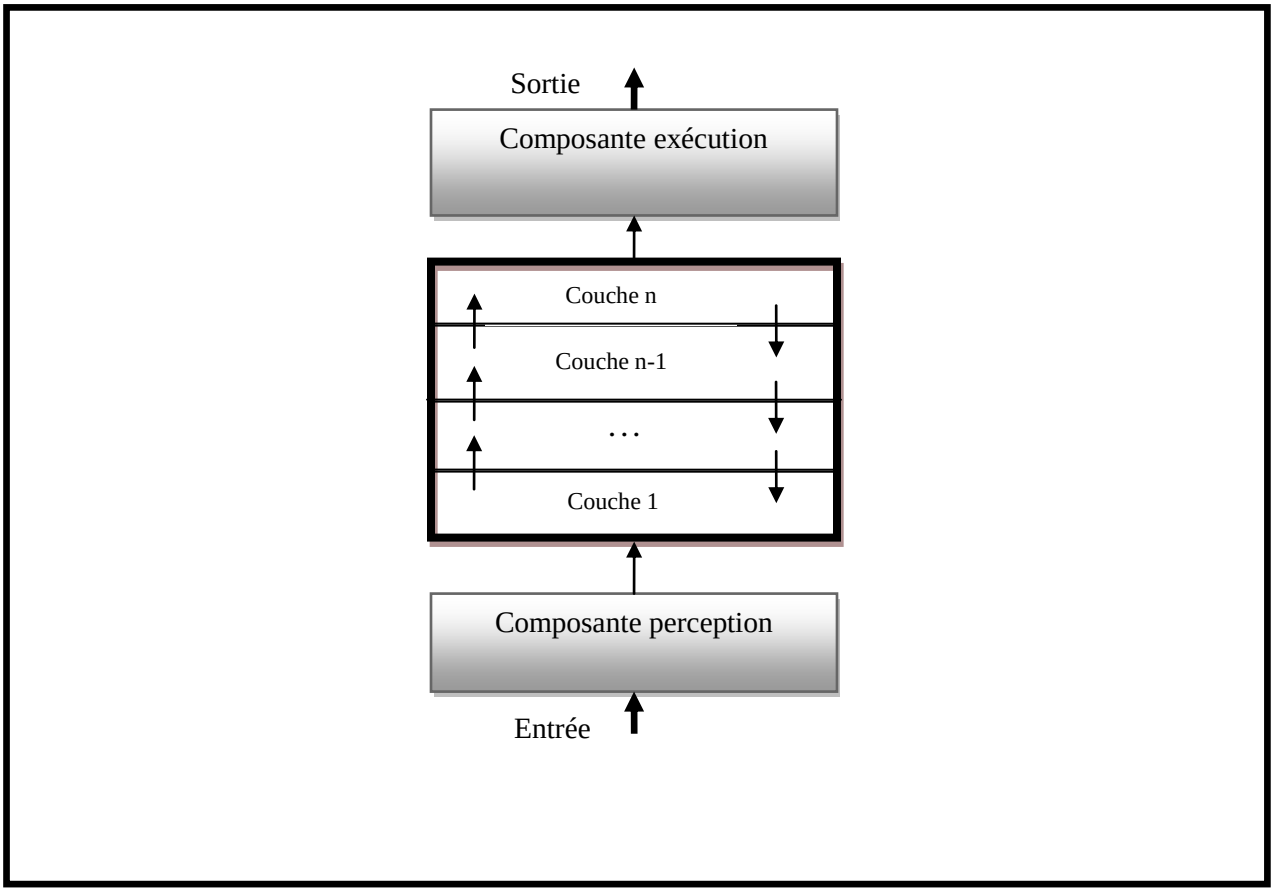

Figure 4: Architecture hybride en couches verticales

4. Type d'Agent

Selon Ferber [Fer95] les agents sont classés selon leur connaissance qui exprime la complexité des fonctionnalités d'agent en deux grandes familles :

- **Agents Cognitifs**

  Ils sont munis de mécanismes de raisonnement évolués et capables de manipuler une représentation symbolique explicite, ce sont des agents qui :

  - Possèdent une représentation explicite de l'environnement.
  - Peuvent tenir compte de leurs passés.
  - Sont des agents complexes.
  - Se trouvent en petit nombre dans la société d'agents.
  - Sont de forte granularité.
  - Communiquent par un langage (par exemple ACL de la FIPA).

- **Agent Réactifs**

  Uniquement capable de percevoir et agir sur l'environnement, Ils n'ont pas une représentation symbolique de l'environnement ou des connaissances et ils ne possèdent pas de croyances, se sont des agents qui :

  - N'ont pas de représentation explicite de l'environnement.
  - N'ont pas de mémoire de son histoire.
  - Fonctionnent suivant le mécanisme stimulus/action.
  - Se trouvent en grand nombre dans un SMA.
  - Se caractérisent par une faible granularité.
  - Communiquent principalement par trace ou signaux (modifications locales de l'environnement perceptibles par les agents cibles).

Remarque : Une troisième classe d'agents a également émergée appelée les agents hybrides intégrant les aspects cognitifs et ceux réactifs.

II.2. Les systèmes Multi-Agents (SMA)

Plusieurs définitions ont été proposées pour le concept SMA parmi ces définitions nous citons :

a. D'après [Cha02] « Un *système multi-agent* est un système distribué composé d'un ensemble d'agents. Contrairement aux systèmes d'IA, qui simulent dans une certaine mesure les capacités du raisonnement humain, les SMA sont conçus et

implantés idéalement comme un ensemble d'agents interagissant, le plus souvent, selon des modes de *coopération*, de *concurrence* ou de *coexistence* ».

b. Selon Ferber [Fer95] : on appelle un système multi-agent (ou SMA) un système composé des éléments suivants :

- *Un environnement E,* c'est-à-dire un espace disposant généralement d'une métrique ;
- *Un ensemble d'objet O*. Ces objets sont situés, c'est-à-dire que, pour tout objet, il est possible à un moment donné, d'associer une position 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 sont des objets particuliers, lesquels représentent les entités actives du système ;
- *Un ensemble de relation 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 du monde à cette tentative de modification, que l'on appellera les lois de l'univers".

c. Un SMA est un système dans lesquels des agents artificiels opèrent collectivement et de façon décentralisée pour accomplir une tâche [Fer95b].

d. Un système multi-agents est un système distribué composé d'un ensemble organisé d'agents [Cha01].

e. L'approche Voyelles (***AEIO***) : basée sur la décomposition des SMA en quatre parties [Dem95] :

- ***Agents :*** qui concernent les modèles (ou les architectures) utilisés pour la partie active de l'agent.
- ***E*nvironnements :** milieux dans lesquels sont prolongés les agents.
- ***I*nteractions :** concernent les infrastructures, les langages et les protocoles d'interaction entre agents.

- ***Organisations :*** qui structurent les agents en groupes, hiérarchie, relations, etc.

L'approche voyelles est guidée par trois principes :

1. ***le principe déclaratif* :** d'un point de vue déclaratif un SMA est composé d'agent, environnement, interaction et organisation :

   SMA = Agent + Environnement + Interaction + Organisation

2. ***Le principe fonctionnel :***

   Fonction (SMA) = Fonction (agents) + Fonction collectives

3. ***Le principe de récursion :***

   (SMA* = SMA)

Un SMA peut être [Joe05] :

- ***Ouvert*** : les agents y entrent et en sortent librement (ex: un café).
- ***Homogène*** : tous les agents sont construits sur le même modèle (ex: une colonie de fourmis).
- ***Hétérogène*** : des agents de modèles différents, de granularité différentes (ex: l'organisation d'une entreprise).

En plus, il existe d'autres caractéristiques des SMA qu'on peut les résumés dans [Fer95b]**:**

- il n'y a aucun contrôle global du système multi-agents.

- chaque agent a des informations ou des capacités de résolution de problèmes limitées, ainsi chaque agent à un point de vue partiel.

- les données sont décentralisées.

- le calcul est asynchrone.

Les systèmes multi-agents se distinguent d'une collection d'agents indépendants par le fait que les agents interagissent en vue de réaliser conjointement une tâche ou d'atteindre conjointement un but particulier [Cha01]. En accord avec Ferber [Fer95], la notion d'interaction est au centre de la problématique des systèmes multi agents. On entend par l'interaction dans le contexte des systèmes multi-agents l'ensemble d'actions réalisables par un agent pour agir sur le monde qui l'environne [Tra01]. Notons que l'interaction n'est pas limitée au simple échange des

messages, elle montre une conversation basée sur un échange conventionné de messages [Bar95], dont des schémas typiques utilisés pour la structuration des conversations (*Protocoles d'Interaction*). En fait, Les différentes formes d'interactions sont [Tra01] :

a) ***La communication*** : la communication est à la base des interactions et de l'organisation sociale d'un SMA, car elle permet l'échange des informations et la coordination des activités entre les différents agents. La communication entre agents peut être directe (un agent échange les messages avec ses accointances), ou indirecte (les agents interagissent via une base de données partagée appelée tableau noir).

b) ***La coopération entre agents :*** La coopération est l'une des plus importantes rubriques des SMA où les agents doivent coopérer pour atteindre un objectif commun. D'après J. Ferber [Fer95] la Coopération = action d'opérer conjointement avec quelqu'un, en précisant que ce « quelqu'un » ne se restreint pas a une personne humaine. Glize, Gleizes et Camps [Gli98] ont démontré que tout système coopératif est fonctionnellement adéquat. «Un système est fonctionnellement adéquat si son activité est «correcte» dans l'environnement dans lequel il est immergé. Il s'intègre, ainsi, durablement dans le monde ou en façonne un nouveau. L'activité correcte n'est décidable que par un observateur extérieur jugeant les interactions et connaissant la fonction que le système doit réaliser dans son environnement».

c) ***La coordination d'activités :*** afin d'assurer la cohérence d'un système multi-agents et de surmonter les situations conflictuelles engendrées par l'environnement, les agents doivent coordonner leurs activités. La coordination d'actions est l'ensemble des activités supplémentaires qu'il est nécessaire d'accomplir dans un environnement comprenant plusieurs agents [Bon88].

d) ***La négociation :*** dans une situation conflictuelle, les agents peuvent être amenés à négocier leurs actions afin d'arriver à une solution [Bri01]. Durfee [Dur89] définissent la négociation comme étant le processus qui consiste à améliorer les accords (en réduisant les inconsistances et l'incertitude) sur des points de vue communs ou des plans d'action grâce à l'échange structuré d'informations pertinentes.

III. Les Méthodologies d'analyse et de conception des SMA

Dans l'objectif de guider et de faciliter le développement des systèmes multi-agents, plusieurs méthodologies ont été émergées dans la littérature. Ces méthodologies sont, en fait, inspirées soit de celles orientées objets soit de celles issues de l'ingénierie de connaissances [Igl99]. Elles étendent les méthodologies existantes afin d'assurer une prise en compte explicite des concepts qui entoure le paradigme agent pour la construction d'agents et de système multiagents [Igl99].

Parallèlement, la maturité du paradigme agent a favorisé le développement des méthodologies directement centrées sur les agents. Celles-ci offrent une prise en compte explicite des rôles, des comportements autonomes, des interactions et de l'organisation sociale [Woo00]. La figure suivante présente une classification des méthodologies développées dans le champ de recherche du AOSE(Agent-Oriented software Engineering)[Alo04].

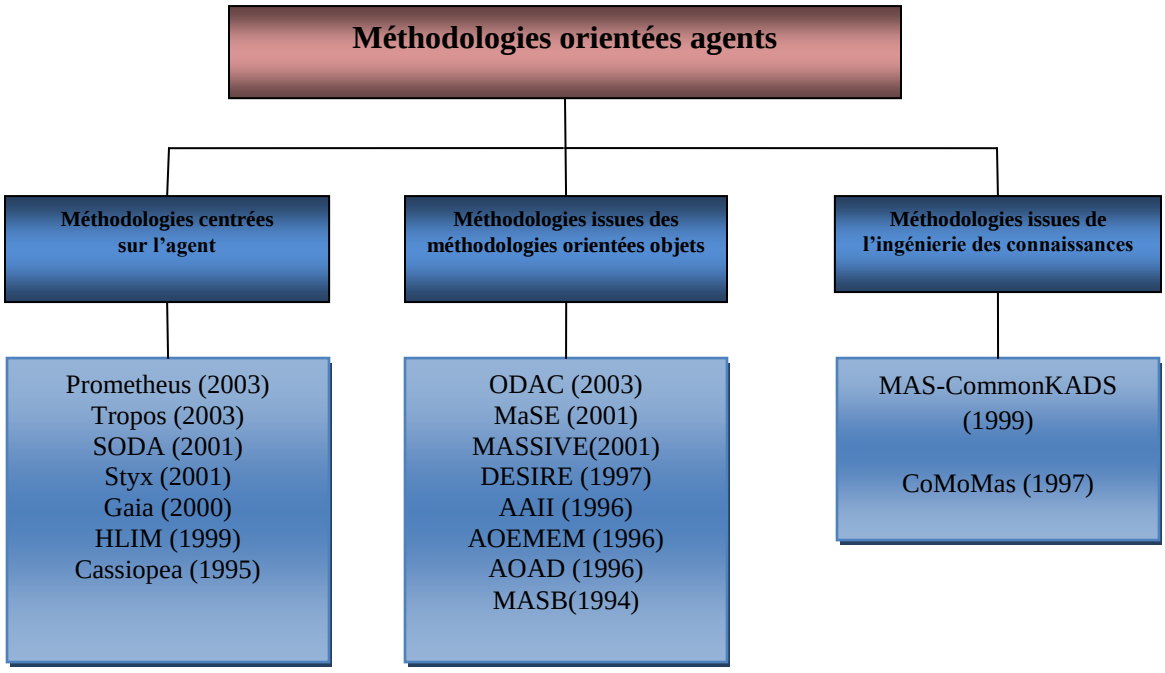

Figure 5: Les méthodologies orientées agents.

III.1. Méthodologies issues des méthodologies orientées objets

Dans cette partie nous présentons la méthodologie MaSE [Del01] qui possède un processus de développement très clair et facile à comprendre. En outre, cette méthodologie est très connue et très utilisée dans le développement des SMA.

1. La Méthodologie MaSE (Multiagent Systems Engineering) [Del01]

Cette méthodologie propose d'assister les phases de développement et de déploiement de systèmes multi-agents. Cette méthodologie offre un guide de conception de la phase de spécification jusqu'à son implémentation. Elle est composée de deux phases principales: analyse (*Identifier les buts*, *Appliquer les cas d'utilisation* et *Perfectionner les rôles*) et conception (*Créer les classes d'agent*, *Construire les conversations*, *Assembler les classes d'agent* et *Concevoir le système*). Les étapes associées à cette méthodologie sont les suivantes :

- ***Identifier les buts***: identifie les buts du système afin de les structurer et de les représenter sous la forme d'une hiérarchie.
- ***Appliquer les cas d'utilisation***: identifie les rôles et leurs interactions selon deux sous-étapes en utilisant les cas d'utilisation et les diagrammes de séquence. Les cas d'utilisation définissent le comportement général du système, ses fonctionnalités, son environnement (utilisateurs et acteurs) et les différents rôles du système. Les diagrammes de séquence représentent les messages échangés entre les rôles.
- ***Perfectionner les rôles*** : identifie la décomposition fonctionnelle du système selon deux sous-étapes : Rôles et Tâches Concurrentes. Chaque rôle correspond au moins à un but dont il est responsable de l'accomplir, auquel s'ajoute un ensemble de tâches correspondantes. Lors de la création des modèles de rôles, les interactions entre les rôles sont définies en connectant les tâches entre elles. Le modèle de Tâches Concurrentes permet de représenter, à l'aide d'automates à états finis, les tâches réalisées par les rôles pour l'accomplissement de leurs buts respectifs.
- ***Créer les classes d'agent***: identifie le système multi-agents par la description des architectures d'agents et de leurs liens conversationnels. Une classe d'agent précise les rôles qui lui sont assignés.

- ***Construire les conversations*** : permet de définir les protocoles de coordination entre deux agents. Il s'agit de détailler, à l'aide d'automates à états finis, les comportements des agents durant la conversation.
- ***Assembler les classes d'agent***: spécifier l'architecture interne des agents.
- ***Concevoir le système***: spécifier la distribution des agents selon l'architecture physique du système.

La figure suivante peut clarifier le processus de développement des SMA suivi par la méthodologie **MaSE :**

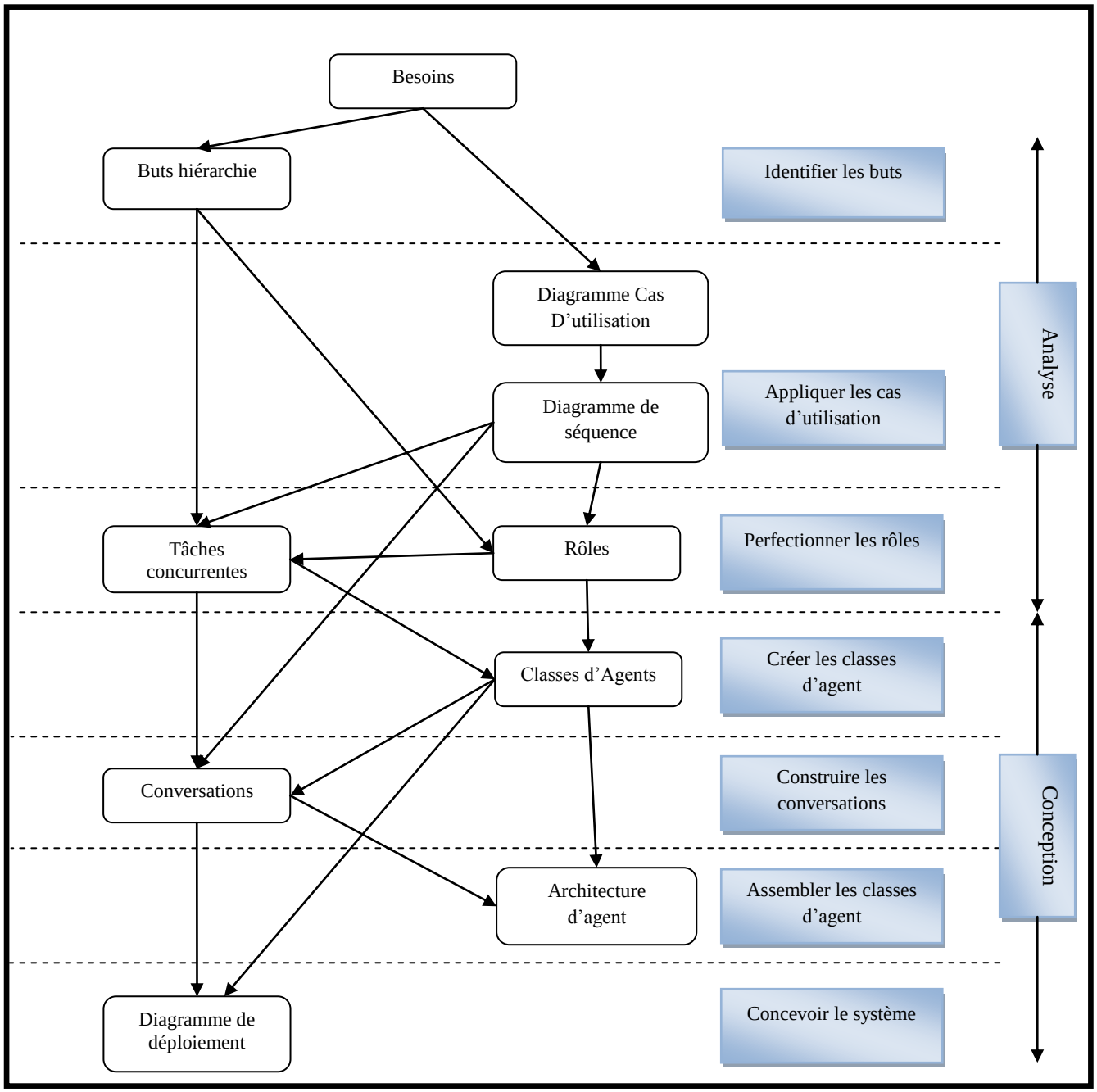

Figure 6: La Méthodologie MaSE.

2. Avantage de l'approche

Plusieurs raisons qu'on peut citer pour justifier l'extension des méthodologies orienté objet :

Premièrement, il y a des similitudes entre le paradigme orienté Objet et le paradigme orienté-agent [Bur96, Kin96]. Depuis les temps tôt, ils sont établis des rapports entre l'intelligence artificielle distribuée (IAD) et de la programmation Basée Objet Concurrente (OBCP) [Ala88, Gas92]. Comme indiqué par Shoham [Sho93], les agents peuvent être considérés en tant qu'objet actifs, objet avec un état mental. Les deux paradigmes utilisent l'envoi des messages pour communiquer entre eux. La différence principale [Sho93] est le type des messages dans le paradigme orienté-agent et la définition de son état qui est basé sur ses croyances, désirs, intentions, engagements, etc.

Beaucoup de méthodologies orientées objet sont utilisées dans l'industrie avec un succès tel que la technique de modélisation d'objet (OMT) [Rum91], la technologie de la programmation orientée objet (OOSE) [Jac92], la conception orientée objet [Boo91], le RDD (Responsibility Driving Design) [Wir90] et UML (Unified Modelling Language) [Rsc97]. Cette expérience peut être une clé pour faciliter l'intégration de la technologie agent parce que d'une part, les ingénieur en logiciel peuvent être peu disposées à utiliser et apprendre une nouvelle méthodologie complète, et d'une part, les gestionnaires préférerent suivre les méthodologies qui ont été testées avec un succès [Igl99].

Et finalement, certaines techniques pour l'identification d'objets peuvent également être utilisées pour identifier des agents, en l'occurrence, *les cas d'utilisation* [Jac92].

3. Inconvénient de l'approche

Malgré les similitudes entre les objets et les agents, évidemment, les agents ne sont pas un simple objet. Ainsi, les méthodologies orientées objet n'adressent pas ces différents aspects [Sho93, Bur96, Eli96].

Premièrement, bien que les objets et les agents utilisent l'envoi de messages pour communiquer les uns avec les autres, un envoi de message pour l'objet n'est qu'un appel de méthode, tandis que les agents distinguent différents types de messages et échangent ces messages le plus souvent dans des protocoles complexes. En outre, les agents analysent ces messages et peuvent décider s'ils exécutent ou non des actions, ils sont munis de mécanisme de raisonnement évolués et capables de manipuler une représentation explicite [Igl99].

Une autre différence consiste que les agents peuvent être caractérisés par leur état mental, et les méthodologies orientées objet ne définissent pas des techniques pour modeler comment les agents effectuent leurs inférences, leur processus de planification, etc [Igl99]. En outre, les agents sont autonomes et responsables de leurs actions alors que les objets n'en sont toujours pas. [Sab01]

Et finalement, les agents sont caractérisés par leur aspect social (interagir avec des agents si nécessaire afin de résoudre ses propres problèmes et qu'il doive être capable *d'aider* les autres dans leurs activités). Des procédures pour modeler ces relations sociales entre les agents doivent être définies [Igl99].

III.2. Méthodologies issues de l'ingénierie des connaissances

A ce niveau deux méthodologies ont été proposées qui sont basées sur l'ingénierie de connaissance à savoir, CoMoMAS et MAS-CommonKADS.

1. **La Méthodologie CoMoMAS** (Contribution to Knowledge Modelling in a Multi-Agent Framework)

Glaser [Gla96] propose une extension de la méthodologie CommonKADS [Kin96] pour la modélisation des SMAs à l'aide des modèles suivants :

- ***Modèle d'agent*** **:** c'est le modèle principal de la méthodologie qui définit l'architecture et la connaissance d'agent.
- ***Modèle d'expertise*** **:** décrit les compétences cognitives et réactives de l'agent.
- ***Modèle de tache*** **:** décrit la décomposition des tâches, et ses détails si les tâches sont résolues par un utilisateur ou un agent.
- ***Modèle de coopération*** **:** décrit la coopération entre les agents, l'utilisation des méthodes de résolution de conflit et la connaissance de coopération (communication primitives, protocole et les interactions).
- ***Modèle de système*** **:** définit les aspects organisationnels de la société d'agents ainsi que leurs aspects architecturaux.
- ***Modèle de conception*** **:** rassemble les modèles précédents dans un ordre opérationnel avec les besoins non fonctionnels (les besoins de qualité).

2. **La méthodologie MAS-CommonKADS** (Multiagent System Knowledge Analysis and Development)

MAS-CommonKADS [Fis97] étend la méthodologie CommonKADS [Sch94], pour la modélisation des SMAs, en ajoutant des techniques des méthodologies orientées objet(OO) telles que OMT(Object Modelling Technique) [Rum91], OOSE(Object Oriented Software Engineering) [Jac92], RDD(Responsibility Driving Design) [Wir90] à l'ingénierie de protocoles decrivant les protocole d'agents telles que SDL(Specification and Description Language)[Itu94] et MSC(Message sequence charts) [Ekk96]. La méthodologie définit les Modèles suivant :

- ***Modèle d'agent*** **:** détermine les caractéristiques de l'agent: les capacités de raisonnement, les habiletés (percevoir/s'agir), les services, les groupes et la hiérarchie des agents.
- ***Modèle de tâche*** **:** décrit les tâches pouvant être réalisées par les agents (les buts) ainsi que la décomposition des tâches et les méthodes de résolution de problèmes.
- ***Modèle d'expertise*** **:** détermine les connaissances dont les agents ont besoin pour obtenir leurs objectifs.
- ***Modèle d'organisation*** **:** détermine l'organisation dans laquelle le système multi-agents est considéré comme une organisation sociale de la société d'agents.
- ***Modèle de coordination*** **:** détermine les conversations entre les agents : les interactions, les protocoles et les capacités nécessaires.
- ***Modèle de communication*** **:** décrit les interactions entre les utilisateurs et les agents pour le développement d'interfaces adaptées.
- ***Modèle de conception*** **:** collecte les modèles ci-dessus. Il contient trois sous modèles : Conception de réseau qui conçoit les fondations du réseau d'agents. Conception d'agent qui conçoit l'architecture interne de l'agent. Conception de plate-forme qui sélectionne la plate-forme de développement d'agents pour chaque architecture d'agents.

Le cycle de vie de développement de logiciel dans MAS-CommonKADS suit la description des phases ci-dessous :

- **Conceptualisation** : décrit le problème en identifiant toutes les exigences, les besoins, les buts, les tâches...etc. Cette phase utilise les ***cas d'utilisation*** (scenarios), une technique orienté objet, pour aider à comprendre l'exigence informelle et pour tester le système, les interactions sont formalisées à l'aide de MSC (Message Sequence Charts). Le résultat de cette phase c'est d'obtenir une première description du problème.

- **Analyse** : la phase d'analyse détermine les besoins fonctionnels du système multi-agents par le développement d'un ensemble de modèles :

- **Modélisation d'agent** : crée les instances originales pour identifier et décrire les agents en utilisant le modèle d'agent en étapes: l'identification des agents dans le système c-à-d l'identification des acteurs qui apparaissent dans les cas d'utilisation, et l'identification les relations entre les agents proposés en se basant sur les cas d'utilisation.
- **Modélisation de tâche**: découvrit et décrit toutes les tâches possibles dans le système en utilisant le modèle de tâche. Il décompose des tâches en utilisant l'approche de haut à bas et en construisant un arbre des tâches. Chaque tâche est décrite par son nom, ses entrées, ses sorties, sa structure, ses contrôles, les capacités nécessaires...etc.
- **Modélisation de coordination:** le modèle de coordination possède deux étapes à savoir : (1) c'est la définition des voies de communication et la construction d'un prototype. (2) c'est l'analyse des interactions et la détermination des interactions complexes (avec des protocoles de coordination).
- **Modélisation de connaissance**: modélise la connaissance du domaine. Elle contient la connaissance nécessaire pour obtenir l'objectif de l'agent et la connaissance de l'environnement et des autres agents. Il contient trois types de connaissances : La connaissance de domaine concernant le problème, la connaissance d'inférence représentant les étapes inférées pour résoudre une tâche, la connaissance de tâche représentant l'ordre de la structuré inférée.

- **Modélisation d'organisation** : développe le modèle d'organisation en se basant sur le modèle d'agent. Ce modèle présente les relations statiques ou structurées entre les agents.

- Conception : à partir de l'ensemble des agents qui ont été déterminés dans la phase d'analyse, on conçoit le modèle de conception selon les étapes suivante :

- **Conception de réseau d'agent :** détermine les fondations nécessaires pour le système multi-agents. Ces fondations contiennent trois aspects: la facilité de réseau concernant les services du réseau, niveau de sécurité, le protocole de l'application. La facilité de connaissance concernant la représentation des connaissances (ontologie). La facilité de coordination concernant les mécanismes pour la gestion des ressources communes.
- **Conception d'agent** : détermine l'architecture qui convient le plus à chaque agent. Chaque agent est composé de modules: de communication humaine, de communication agent et de réaction de service externe…etc.
- **Conception de platforme** : sélectionner le logiciel (environnement de développement multiagent).

- Développement et test : tâche de codage et test des agents.

- Opération : mettre le système en opération et en maintenance.

3. Avantages de l'approche

Les méthodologies d'ingénierie de connaissances peuvent fournir une bonne base pour la modélisation des SMAs puisqu'elles traitent le développement des systèmes basés sur la connaissance. Puisque les agents ont des caractéristiques cognitives, ces méthodologies peuvent fournir les techniques pour modeler cette connaissance d'agent [Igl99].

La définition de la connaissance d'un agent peut être considérée comme un processus d'acquisition de la connaissance, et ce processus est adressé seulement à ces méthodologies [Igl99].

L'extension des méthodologies de l''ingénierie des connaissances actuelles peut tirer profit de l'expérience acquise de ces méthodologies. En outre, les outils existants et les librairies

d'ontologie développée et les librairies des méthodes de résolution des problèmes peuvent être réutilisés [Igl99].

Finalement ces méthodologies ont été appliquées à plusieurs projets avec succès [Igl99].

4. Inconvénients de l'approche

La plupart des problèmes liés aux méthodologies d'ingénierie de connaissance pour la conception des SMAs sont: acquisition de connaissance, modélisation et la réutilisation, en plus ces méthodologies conçoivent un système basé sur la connaissance en tant que centralisé, Ainsi, ils ne s'adressent pas aux aspects distribués ou sociaux des agents [Igl99].

III.3. Méthodologies basées agents

Les méthodologies centrées sur les agents (Hlim [Eal99], Gaia [Woo00], Tropos[Giu01], Prometheus [Pad02] et DACS [Bus04]) sont caractérisées par la prise en compte des concepts du paradigme agent. Les propriétés telles que l'autonomie, la réactivité, la pro-activité et la sociabilité sont explicitement considérées lors des phases d'abstraction du système étudié. Ainsi, la structure de l'organisation multi-agents est spécifiée dès la phase d'analyse du système. Il s'agit pour ces méthodologies d'intégrer, au niveau de la phase d'analyse et de conception, des notions qui ne sont pas explicitement définies dans les approches basées sur les méthodologies orientées objets [Lab06].

1. La méthodologie HLIM (High-Level and Intermediate Models)

La méthodologie HLIM [Eal99] fait ressortir, à travers deux phases, les aspects des agents tels que : leurs objectifs, leurs plans, leurs croyances et les rapports entre ces agents. Comme les techniques orientées objet capturent les objets dans le système, leurs attributs, leurs structures et leurs liens, HLIM capture les agents, leurs attributs, leurs structures et leurs liens. La méthodologie HLIM est supportée par les UCMs [Buh95] (Uses Case Maps)[1] et qui est basée sur le langage Extented Markup Langage (XML).

Le processus suivi par cette méthodologie est accompli en deux étapes. La première étape a pour but la production d'un Modèle nommé HLM (High level Model). Ce dernier capture la structure et le comportement du système, il permet d'identifier les agents du système et leur

[1] Les UCMs constituent le formalisme utilisé pour la représentation et la modélisation des résultats issus de l'analyse.

niveau de comportement. La deuxième étape sert à obtenir une compréhension claire des comportements des entités qui participent à leur exhibition et des rapports entre ces entités. Cette phase se réalise à travers quatre modèles à savoir :

- **Modèle Interne d'Agent** : qui décrit les agents dans le système en termes de leur structure et leur comportement internes. Il capture les aspects de l'agent tels que les buts, les plans et les croyances.

- **Modèle relationnel :** décrit les relations inter-agents. Ces relations peuvent être de dépendance et de juridiction et sont décrites respectivement par les diagrammes de dépendance et les diagrammes de juridiction. Un diagramme de dépendance clarifie les agents qui fournissent des services entres eux. Un diagramme de juridiction décrit l'organisation des agents en termes de leur statut d'autorité.
- **Modèle Conversationnel** : identifie les messages qui sont échangés par les agents lors de leur coopération et de leur négociation.
- **Model de contrat** : définit les prévisions sur la manière dont les agents peuvent satisfaire les relations de dépendance ainsi que les attentes des agents lorsqu'ils jouent les rôles définis par le diagramme de juridiction.

La figure 7 donne une vue globale des différents modèles la méthodologie HLIM.

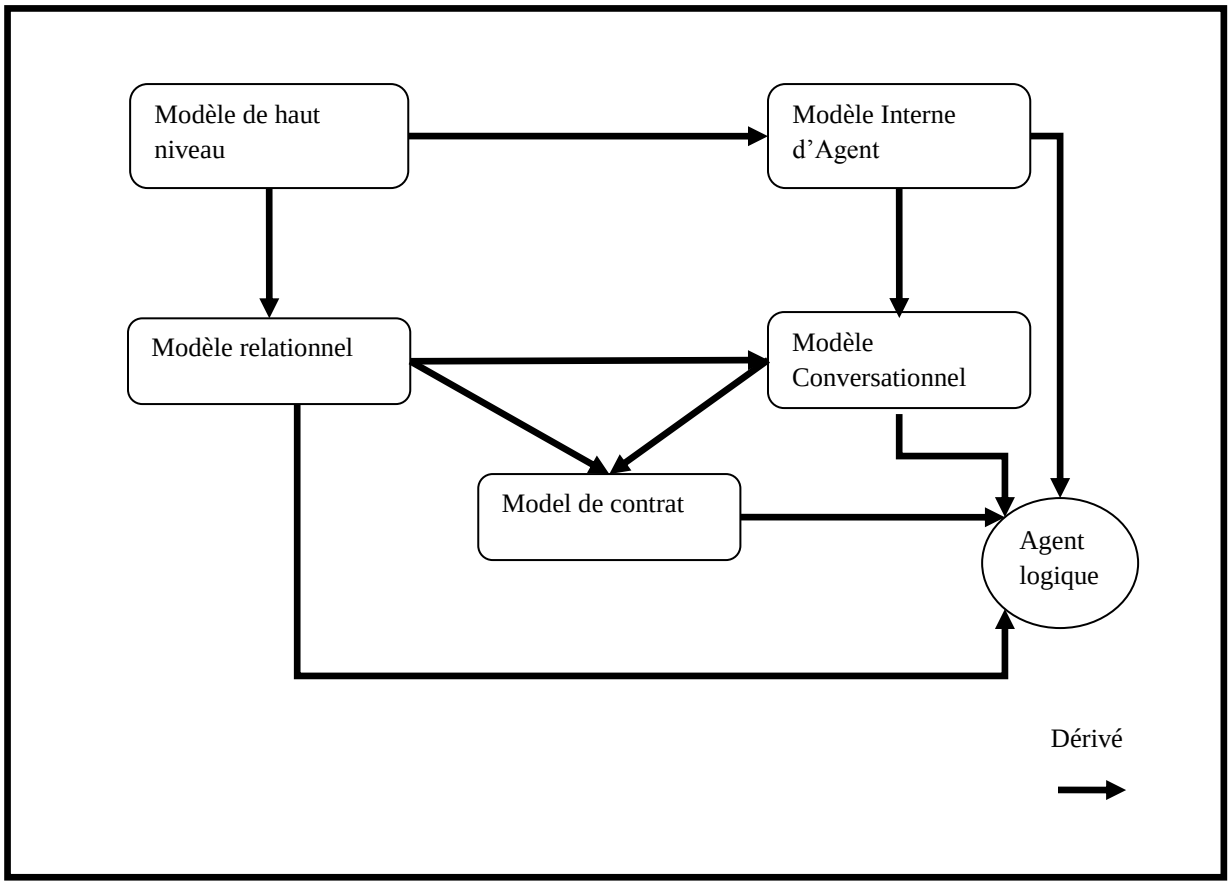

Figure 7: Les Modèles de HLIM.

2. La méthodologie GAIA

Selon Wooldridge [Woo00] GAIA est le nom donné aux hypothèses formulées par l'écologiste *James Lovelock* selon lesquelles les organismes vivants du monde peuvent être compris comme des composantes d'une seule entité, lesquelles composantes régularisent l'environnement mondial. Les auteurs de GAIA considèrent un SMA comme une société ou une organisation artificielle ou encore un ensemble institutionnalisé de rôles. La méthodologie Gaia exploite l'abstraction organisationnelle pour fournir une méthode d'analyse et de conception d'un système ouvert et complexe. Elle s'appuie pour cela sur une description de niveau micro (structure de l'agent) et de niveau macro (société d'agents). Les agents sont identifiés à partir des notions de rôles et de responsabilités. Chaque rôle est composé de quatre attributs : responsabilités, permissions, activités et protocoles. Les responsabilités définissent ce que l'agent est supposé faire. Les permissions définissent les ressources qu'un rôle peut utiliser. Les activités représentent les tâches qu'un rôle peut réaliser sans interaction avec d'autres rôles. Les protocoles définissent comment les agents

interagissent afin de remplir leurs rôles. Ces concepts sont présentés en deux phases : analyse et conception.

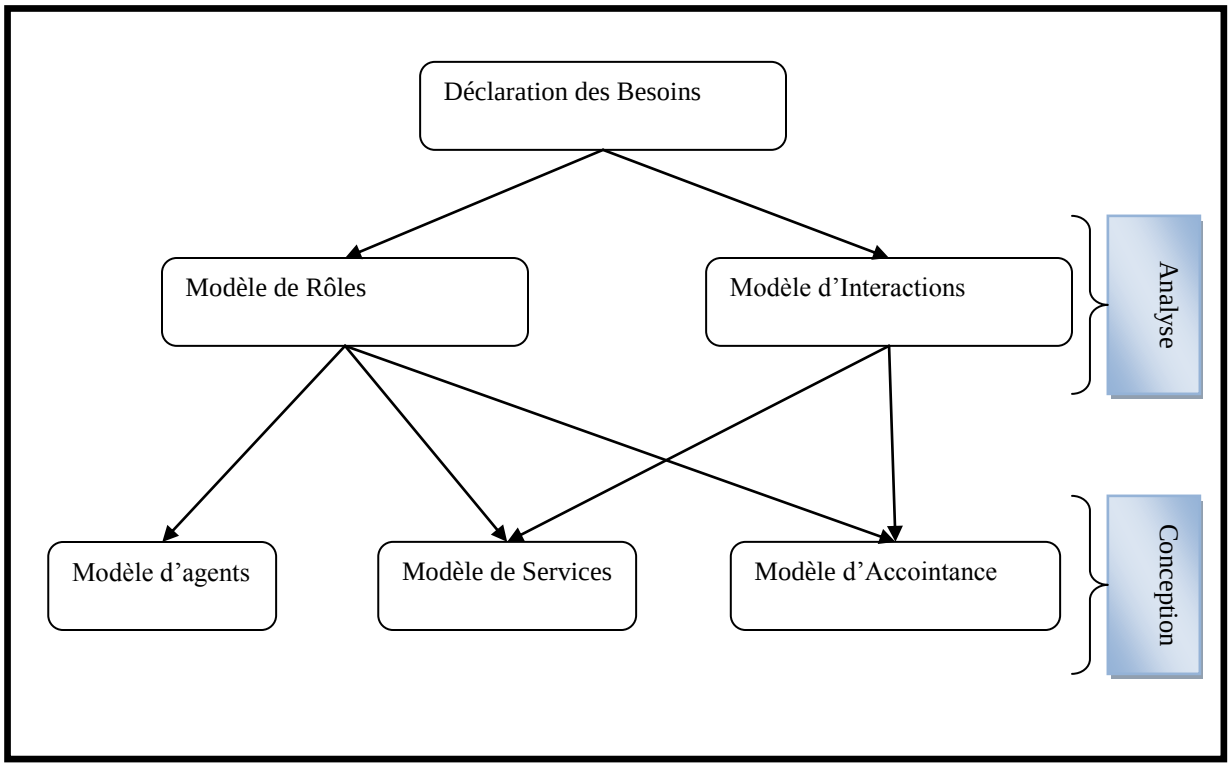

Figure 8: Processus de développement dans Gaia

La figure 8 montre que la méthodologie GAIA possède deux principales phases à savoir :

- **La phase Analyse :** Dans cette phase, les aspects niveau-macro sont adressés par le modèle d'interaction, et les aspects de niveau-micro sont adressés par le modèle de rôle. Elle concentre sur des aspects d'organisation d'agent (tels que des rôles, des responsabilités, des types d'agent, des protocoles) et la conception du domaine d'organisation (telles que services, interactions, connaissances, activités). Dans cette phase, les tâches principales sont de définir une collection de rôles des agents et de définir des relations entre ces agents. Pour envelopper la phase analyse, deux modèles on été définis : modèle de rôle et modèle d'interaction.
    - ***Modèle de Rôle* :** identifie les différents rôles que joueront les différents agents présents dans le système. Les rôles sont définis en fonction des attributs présentés précédemment. La table 1 est appropriée pour faire une description sur les rôles jouerons les agents du système.

Agent /Rôle Schéma: Nom du rôle	
Description: short description of the role	
Protocoles et activités	Protocoles de communication et les activités dans lesquels les rôles jouent.
Autorisations	Les droits associés au rôle.
Responsabilités: Sécurité	Responsabilités en matière de sécurité

Table 1: Schéma du modèle de rôle

- ***Modèle d'Interaction*:** identifie les protocoles de communication entre les rôles.

- **La phase de conception :** on distingue trois modèles définis dans cette phase à savoir : le modèle d'agent, le modèle de service et le modèle d'accointance.
  - **le modèle d'agent :** identifie les agents du système en les associant aux différents rôles.
  - **le modèle de service :** identifie les services associés avec chaque rôle et spécifie les propriétés de ces services.
  - **Modèle d'accointance:** identifie la structure de l'organisation multi-agents en définissant les liens de communication entre les agents.

Une extension de la méthodologie Gaia est proposée dans [Zam03] afin d'intégrer deux niveaux d'abstractions organisationnels : règles organisationnelles et structures organisationnelles. Les règles organisationnelles expriment, lors de la phase d'analyse, les règles sociales sous la forme de contraintes pour l'exécution des protocoles et des activités liées aux rôles. Les structures organisationnelles servent à définir, au niveau de la phase de conception, la topologie du système agents capable de mettre en application les règles organisationnelles définies lors de la phase d'analyse.

La méthodologie Gaia ne propose pas de critères explicites pour l'identification des rôles, aussi bien au niveau individuel qu'au niveau social. De plus, elle ne pas précise pas la manière d'associer les processus de décisions aux rôles et aux agents. [Lab06]

3. La méthodologie TROPOS

Tropos [Giu01] est une méthodologie de développement orienté agents qui couvre entièrement le cycle de développement logiciel : de l'ingénierie des besoins a l'implémentation. Il assure la construction incrémentale d'un modèle du système et de son environnement. Deux caractéristiques clés de TROPOS sont :

- La notion d'agent de type BDI ainsi que les états mentaux sont utilisés dans toutes les phases de développement.
- Une importance spéciale est mise sur l'ingénierie des premiers besoins qui élabore les acteurs de l'application ainsi que leurs intentions.

C'est une méthodologie en cinq phases : *ingénierie de premièrs besoins, ingénierie des besoins avancés, conception architecturale, conception détaillée et implémentation.*

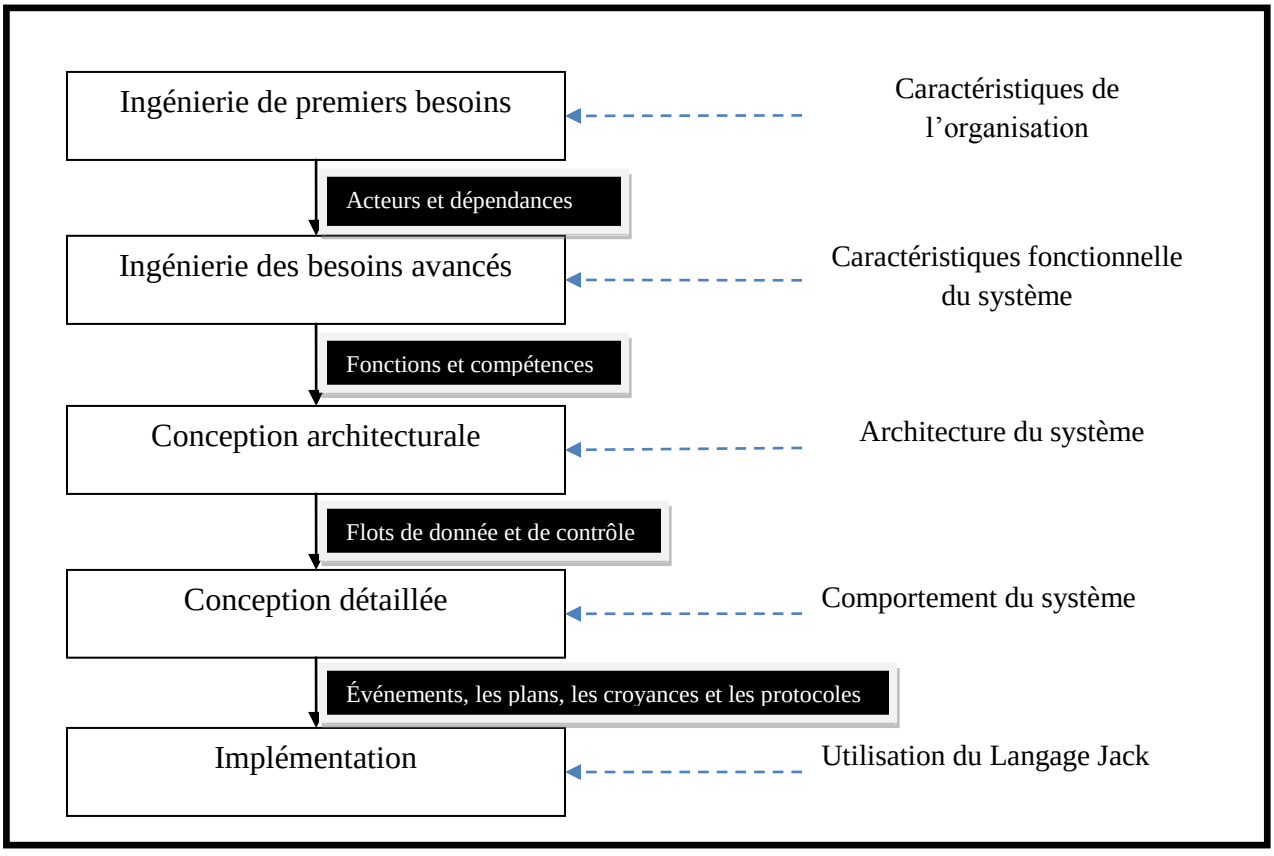

Figure 9: Processus de développement des SMA dans la méthodologie TROPOS

- **Ingénierie des premiers besoins :** c'est la première partie de la phase d'analyse des besoins, elle consiste à comprendre le problème à travers une étude organisationnel du contexte. La sortie de cette phase est un modèle d'organisation qui inclut les acteurs appropriés et leurs dépendances respectives (diagramme d'acteurs et diagramme de buts). Les acteurs sont caractérisés par leurs buts. Le diagramme d'acteurs et de buts permet la création de modèles dans cette phase d'ingénierie des premiers besoins.
- **Ingénierie des besoins avancés :** L'analyse des besoins avancés est consacrée à la définition des besoins fonctionnels et non fonctionnels du système à réaliser. Cette phase prend le système à concevoir comme un acteur. Ce dernier possède un certain nombre de dépendances sociales avec d'autres acteurs dans leur environnement.
- **Conception architecturale :** Définit l'architecture globale du système selon des acteurs (sous-systèmes) interconnectés par leurs dépendances (flots de contrôle et de données). L'introduction des nouveaux acteurs est possible à cette phase. Ce processus se fait en trois étapes :
  - Inclusion de nouveaux acteurs et délégation des sous-buts aux sous-acteurs,
  - Inclusion de nouveaux acteurs selon un choix de style d'architecture spécifique,
  - Inclusion de nouveaux acteurs qui peuvent contribuer positivement à la satisfaction d'exigences.
- **Conception détaillée :** Définit les détails de l'agent.
  - C'est dans cette phase que les buts, les croyances, les capacités des agents ainsi que les communications entre agents sont spécifiés en détail.
  - Cette activité est très reliée au langage de programmation orienté agents choisi.
  - Les diagrammes d'activités d'UML sont utilisés pour représenter les capacités et les plans.
  - Un sous-ensemble des diagrammes d'AUML (Agent UML) est utilisé pour la spécification des protocoles d'interaction entre agents.

  Le diagramme de capacité : modélise une capacité du point de vue d'un agent à l'aide d'un diagramme d'activité d'UML.
  - Un évènement externe établit l'état initial du diagramme.
  - Les nœuds du diagramme représentent les plans.
  - Les transitions modélisent les événements.
  - Les objets représentent les croyances.

- **Implémentation :** La phase d'implémentation se fait à l'aide du langage de programmation orienté agents nommé JACK. L'architecture des agents de JACK est de type BDI, l'agent à des buts qu'il tente de satisfaire à l'aide de ses plans et de ses croyances.
  - Agent **:** définit le comportement de l'agent en incluant ses capacités, les types de messages et d'événements auxquels il répond et les plans qu'il utilise.
  - Capacité : peut inclure des plans, des événements, des croyances ainsi que d'autres capacités tout comme un agent mais en permettant une certaine réutilisation.
  - Croyance : est définit dans une base de données relationnelles contenant toutes les croyances d'un agent.
  - Événement : définit une condition permettant une action de l'agent autant pour les événements internes qu'externes de la conception détaillée.
  - Plan : définit une séquence d'instructions permettant l'atteinte d'un but.

4. La méthodologie PROMETHEUS

La méthodologie PROMETHEUS [Pad02] a été développée pour le besoin de trouver une manière d'aider les étudiants et des développeurs en industrie pour la conception des systèmes d'agent BDI. Elle est appropriée pour la conception des systèmes clos contenant des agents contrôlés et fiables, par contre, elle n'est pas appropriée pour la conception de systèmes ouverts [Mar03]. Prometheus est complète dans le sens où elle couvre toutes les activités nécessaires au développement des systèmes basés sur des agents intelligents.

La méthodologie PROMETHEUS se compose de trois phases, à savoir : *la spécification de système, la conception architecturale et la conception détaillée* [Pad02].

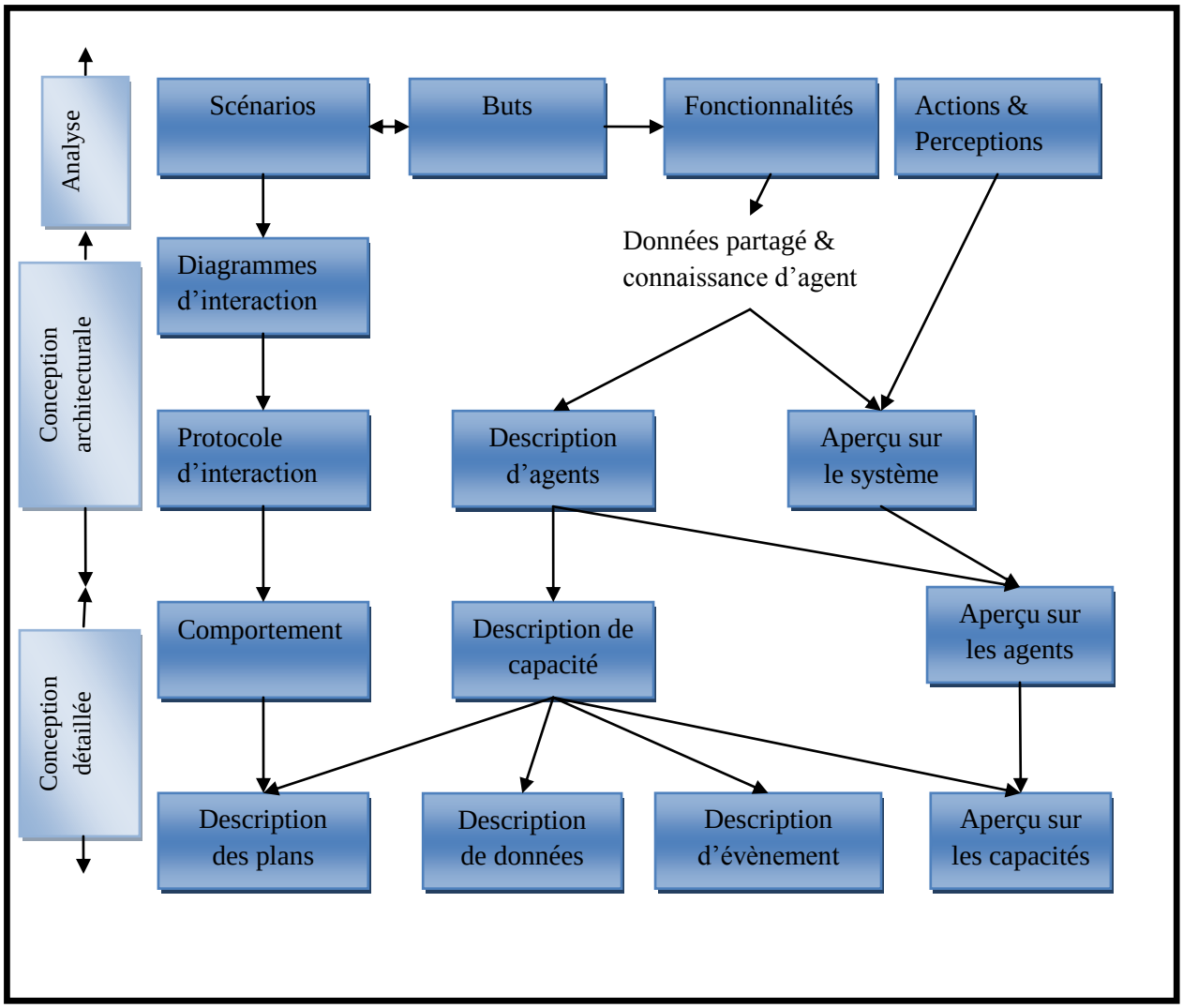

Figure 10: Les phases de la méthodologie Prometheus

- **La spécification de système :** Cette phase sert à définir les actions, les perceptions et les fonctionnalités du système. Les actions et les perceptions définissent l'interface entre les agents et leur environnement. Les fonctionnalités décrivent en un sens plus large ce que devrait faire le système. Des scénarios de cas d'utilisation sont créés pour fournir une vue plus globale de l'interconnexion entre actions, perceptions, et fonctionnalités.
- **La conception architecturale :** consiste à définir les agents du système et leurs fonctionnalités. Au cours de cette sous phase, on définit également les événements auxquels les agents réagissent, les messages qu'ils peuvent recevoir ou émettre. Les

protocoles d'interaction sont donc spécifiés sur la base des diagrammes d'interaction. Enfin, les données partagées sont identifiées à cette étape.

- **La conception détaillée :** Elle se préoccupe de la structure interne des agents et de la façon dont ils exécutent leur tâche.

Chacune de ces phases inclut des modèles qui se concentrent sur la dynamique du système, les modèles (graphiques) qui se concentrent sur la structure du système ou de ses composants, et les formes textuelles de descripteur qui fournissent les détails pour les différentes entités.

5. La méthodologie DACS (*Designing Agent-based production Control System*)

La méthodologie DACS [Bus04], est orientée vers le développement de systèmes multi-agents pour le contrôle de systèmes de production. Elle s'appuie pour cela sur l'identification des agents et sur la conception des interactions qui les unissent. En outre, elle est orientée vers les concepteurs de systèmes industriels qui ne disposent pas nécessairement de connaissances spécifiques dans le domaine des systèmes multi-agents. Cette méthodologie est composée des trois étapes suivantes :

- ***Analyse des processus décisionnels:*** identifie les décisions qui interviennent pour le contrôle d'un système de production. Il s'agit, à l'issue de cette étape, d'obtenir un modèle de décisions au sein duquel sont reportées les tâches décisionnelles, ainsi que les dépendances entre décisions qui requièrent des interactions.
- ***Identification des agents***: identifie l'architecture du système multi-agents. Il s'agit de grouper les tâches décisionnelles en fonction de différents critères, et ensuite affecter un agent par groupe de décisions identifié. Au cours de cette étape sont définis : les agents, les décisions dont ils assurent la responsabilité, ainsi que les besoins d'interactions par agent.
- **Sélection des protocoles d'interaction**: identifie, à partir d'une librairie existante, les protocoles adaptés aux situations d'interactions. Ces protocoles sont par la suite adaptés aux situations courantes.

L'application de la méthodologie DACS permet d'identifier, sur la base de critères décisionnels, les agents ainsi que leurs responsabilités. Les interactions entre les décisions vont définir l'architecture du système multi-agents. Cette méthodologie a été appliquée au

sein de la société *DaimlerChrysler* pour le contrôle d'une unité d'assemblage de tableaux de bord. La méthodologie DACS n'adresse aucune recommandation sur le choix d'architecture d'agent. De plus, cette méthodologie ne repose pas sur une modélisation de domaine (système de production) pour la conception du système multi-agents. Enfin, les considérations d'opérationnalisation du modèle multi-agents sont écartées du cadre méthodologique.

IV. Conclusion

Nous avons présenté dans ce chapitre les notions de base des systèmes multi-agents ainsi que plusieurs méthodologies orientées-agent existantes dans la littérature. Ces méthodologies sont soient issues des méthodologies orientées objet soient issues des méthodologies de l'ingénierie des connaissances ou centrées directement sur le paradigme agent. Nous avons également discuté les avantages et les inconvénients de chacune d'approches. Nous nous intéressons dans ce manuscrit par les méthodologies qui utilisent les diagrammes de cas d'utilisation dans l'étape de l'analyse de besoin. Une étude comparative est présentée dans le chapitre 3.

CHAPITRE II :

La Logique de Réécriture et le Langage Maude

Chapitre II :

La Logique de Réécriture et le Langage Maude

I. Introduction

La spécification de logiciels est une activité d'importance extrême dans le développement de logiciels de qualité. Cette activité peut être faite suivant des approches informelles, semi-formelles ou formelles. Chacune de ces approches possèdent des avantages et des inconvénients. Bien que les deux premières approches soient plus utilisées dans la littérature par rapport à la dernière, l'automatisation des activités de validation et de vérification nécessite une spécification plus rigoureuse en termes de spécification formelle.

A l'heure actuelle, plusieurs langages de spécification formelle (Lotos[Iso89], Z[Abr80], Object-Z[Duk95], Maude [Cla96]) existent dans la littérature, basés sur des logiques relativement différentes. Nous avons opté dans ce mémoire, pour le langage Maude. Basé sur une logique saine et complète dite la logique de réécriture, le langage formel et orienté objet Maude supporte la spécification et la programmation de systèmes concurrents. Quoique chaque langage soit bien approprié à une certaine catégorie d'application, Maude offre un pouvoir d'expression assez riche pour la spécification formelle et la programmation des systèmes concurrents. Il intègre à la fois la puissance de description (spécification et programmation), la possibilité de simulation pour validation et en particulier la vérification formelle basée sur les techniques du model-checking [Mok07].

Quant à la logique de réécriture, elle est considérée comme un cadre unificateur de plusieurs modèles formels qui expriment la concurrence [Mes90, Mes00]. Parmi ces modèles nous citons: les systèmes de transitions étiquetés [Mar93], les réseaux de pétri [Mar95], la machine chimique abstraite [Mar95].

II. La logique de réécriture

La logique de réécriture, dotée d'une sémantique saine et complète, a été développée par **José Meseguer** et son groupe au laboratoire d'informatique SRI International [Mes92] comme une conséquence des travaux sur les logiques générales. Cette logique unifie tous les modèles formels qui expriment la concurrence [Mes90, Mes00]. Elle permet de décrire les systèmes concurrents qui ont des états et qui évoluent en terme de transition, cette logique est représenté par une théorie de réécriture T= (Σ, E, L, R), tel que :

- La structure statique du système est décrite par la signature (Σ , E) qui représente les états d'un système.
- La structure dynamique est décrite par les règles de réécriture de la forme: rl [l]: t->t' qui représentent les transitions. Dans la logique de réécriture, les formules logiques sont dites règles de réécriture [Mes90].

Une règle de réécriture peut aussi être conditionnelle de la forme crl [l]: t->t' if C . Ce qui indique que le terme ***t*** devient (se transforme en) ***t'*** si une certaine condition *C* est vérifiée. Le terme ***t*** représente un état partiel d'un état global du système décrit.

Les travaux de recherche en logique de réécriture, bien qu'ils sont toujours très jeune, ils ont montré de bons signes de vitalité, y compris deux ateliers internationaux [Mes96, Kir98] et plus d'une centaine de papiers de recherches, et trois langages d'implémentation : ***ELAN*** [Kir95, Bor96] en **France**, ***CAFEOBJ*** [Fut94, Fut98] au **Japon**, et ***MAUDE*** en **USA**. MAUDE offre un cadre formel puissant pour la spécification et la vérification du comportement des systèmes concurrents [Att92].

II.1. La Théorie de Réécriture

Une théorie de réécriture est représentée par le quadruplet T= (Σ, E, L, R) tel que :

1. Σ est une signature algébrique,
2. E est un ensemble d'équations,
3. L est un ensemble d'étiquettes (Label) et
4. R est un ensemble de règles de réécriture : ensemble de règles de réécriture qui peuvent être conditionnelles et/ou inconditionnelles. Elles sont de la forme :

- règle de réécriture inconditionnelle : **R : [t] → [t'].**
- règle de réécriture conditionnelle : **R : [t] → [t'] if Cond.**

II.2. Règles de Déduction

Etant donné une théorie de réécriture, l'ensemble des formules appliquées par conséquence logiques de cette théorie est défini par les règles de déduction de la logique de réécriture. Dans un système concurrent, la séquence des transitions exécutées à partir d'un état initial donné, constitue ce qu'on appelle le calcul qui correspond à une preuve ou à une déduction dans la logique de réécriture.

Pour une théorie de réécriture R, on dit que [t] → [t'] est prouvable dans R (ou R implique une formule [t] → [t']) et en écrit R ⊢ [t] → [t'] si et seulement si [t] → [t'] est obtenue par une application finie des règles de déduction suivantes:

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

$$\frac{}{[t] \rightarrow [t']}$$

- **Congruence :** pour chaque $f \in \Sigma_n$, $n \in \boldsymbol{N}$

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

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

  $r : [t(x_1, \ldots, x_n)] \rightarrow [t'(x_1, \ldots, x_n)]$ if $[u_1(\bar{x})] \rightarrow [v_1(\bar{x})] \wedge \ldots \wedge [u_k(\bar{x})] \rightarrow [v_k(\bar{x})]$ dans R,

  $[w_1] \rightarrow [w'_1] \ \ldots \ [w_n] \rightarrow [w'_n]$

$$[u_1(\bar{w}/\bar{x})] \rightarrow [v_1(\bar{w}/\bar{x})] \ \ldots \ [u_k(\bar{w}/\bar{x})] \rightarrow [v_k(\bar{w}/\bar{x})]$$

$$\overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$$

$$[t(\bar{w}/\bar{x}))] \rightarrow [t'(\bar{w}'/\bar{x}))]$$

Cette règle montre que par une substitution $(\bar{w}/\bar{x}))$ de x_i par w_i, $1 \leq i \leq n$, on peut déduire que $[u_j(\bar{w}/\bar{x}))] \rightarrow [v_j(\bar{w}/\bar{x}))]$, $1 \leq j \leq k$, en plus si on peut déduire [wi] → [w'i], pour $1 \leq i \leq n$, il en résulte $[t(\bar{w}/\bar{x}))] \rightarrow [t'(\bar{w}'/\bar{x}))]$.

- **Transitivité :**

$$\frac{[t1] \rightarrow [t2] \quad [t2] \rightarrow [t3]}{[t1] \rightarrow [t3]}$$

- **Symétrie :**

$$\frac{[t1] \rightarrow [t2]}{[t2] \rightarrow [t1]}$$

II.3. La réécriture concurrente

Conséquence importante de la définition de la logique de réécriture est que la réécriture concurrente, au lieu d'émerger comme une notion opérationnelle correspond exactement à la déduction dans cette logique.

Les règles de réécriture définies dans une théorie de réécriture sont indépendants les unes des autres, c-à-d, il n'y a aucun ordre d'exécution entre elles et à chaque étape de déduction, touts les règles dont la partie gauche correspond à un sous terme de l'expression actuelle seront identifiées et pourront être appliquées en concurrence. Selon le nombre de règles identifiées par la règle de remplacement, nous aurons la taxonomie suivante d'une séquence de déduction [t] → [t'] :

1. si le nombre des règles de réécriture identifiées par la règle de remplacement de la logique de réécriture est nul, le pas est appelé « 0-step concurrent T-rewrite » car il n'y a pas de réécriture réelle mais un remplacement d'un terme par un autre équivalent.

2. si la règle de remplacement est utilisée au moins une fois, ce qui signifie qu'au moins une règle de réécriture est appliquée, mais la règle de transitivité n'est pas utilisée, la réécriture [t] → [t'] est appelée « one-step concurrent T-rewrite » car elle est obtenue en une seule étape de déduction.

3. dans le cas ou toutes les règles de déduction sont appliquées, la réécriture [t] → [t'] est obtenue après plusieurs étapes ([t] → [t1] → … → [tn] → [t']) et sera appelée « concurrent T-rewrite ».

Remarque : *Nous appelons une théorie de réécriture T séquentielle, si toutes les réécritures sont nécessairement séquentielles. Une théorie de réécriture séquentielle T est en plus déterministe, si pour chaque [t] il existe au plus une réécriture* « one-step » (*nécessairement séquentielle)* [t] → [t'] dans ce cas il n'y a pas de choix entre plusieurs règles de réécriture.

III. MAUDE

MAUDE est un langage formel de programmation déclarative et outil de spécification formelle basé sur la logique de réécriture **[**Mes98**].** Il étend la programmation déclarative au–delà de ses présentes réalisations statiques à une programmation beaucoup plus dynamique pour couvrir des applications concurrentes **[**Cla09**]**.

✓ Le choix de Maude est motivé essentiellement par [Pet05]:

- Le paradigme orienté objet qu'il supporte et qui n'est pas soutenu par la plupart des méthodes et outils formels.

- Sa capacité à modéliser les systèmes concurrents.

- Le rendement élevé et la robustesse aussi bien qu'une syntaxe intuitive.

- Ses spécifications qui sont exécutables, ce qui permet aux utilisateurs de valider par simulation leurs systèmes.

Par ailleurs, le langage Maude offre plusieurs avantages, nous citons entre autres [Cla09] :

- ✓ **Simple :** les programmes ont une sémantique très claire pour être aussi simple et compréhensible que possible. Ils utilisent des équations et des règles qui ont une sémantique de réécriture très simple.
- ✓ **Expressif :** il est possible d'exprimer naturellement un grand champ d'applications s'étendant de simples systèmes déterministes à des systèmes concurrents de grande complexité.
- ✓ **Performant :** le langage est utilisé non seulement pour des spécifications exécutables mais aussi pour une programmation réelle.

Le langage MAUDE est utilisé dans une large catégorie d`applications telles que la bioinformatique, la définition du langage de programmation, l'analyse et la spécification matériel et logiciels, la programmation Internet, les agents distribués…etc. **[**Cla09**]**. Il existe deux versions du système Maude : *Core Maude* et *Full Maude*.

Core Maude

On appelle *Core Maude* l'interpréteur de Maude implémenté en C++. Il offre toutes les fonctionnalités de base de Maude. Dans *Core Maude* on utilise seulement les modules fonctionnels et les modules systèmes non paramétrés. Il offre une bibliothèque de modules prédéfinis permettant ainsi aux utilisateurs de se décharger de développer plusieurs types abstraits de données. Les techniques de model-checking, la réflexivité, le métalangage ainsi que la mise au point sont supportés par *Core Maude*.

Full Maude

Il représente une extension de Maude, écrit en Maude. Les modules peuvent être paramétrés, et peuvent aussi être instanciés par des traces appelés *views*. Les paramètres sont des théories représentant les exigences sémantiques pour une instanciation correcte. Les théories peuvent être elles-mêmes paramétrés. *Full Maude* utilise les modules orientés-objet (qui peuvent aussi être paramétrés) s'appuyant sur les notions d'objet, message, classes et héritage.

III.1. Syntaxe de base du langage MAUDE

1. Les Identificateurs

Les identificateurs sont les éléments de base, ils sont employés pour nommer les modules, les sortes et pour former des noms d'opérateurs par exemple NAT, Nat, et Agent sont des identificateurs. En général un identificateur dans MAUDE est une chaîne de caractère d'ASCII tel que :

- Ne contient pas d'espace.
- Les caractère spéciaux '{', '}', '(', ') ', '[', '] ' et ',' séparent une séquence de caractères en différents identificateurs, par exemple ' ab{c,d}eh' compte sept identificateurs nommés 'ab' , '{' , 'c' , ',' , 'd' , '}' et 'eh' .
- Le caractère ' ‘ ' indique que le blanc ou les caractères spéciaux ne séparent pas la séquence de caractère, il est placé devant tout caractère spécial ou bien entre deux chaîne vides.

2. Sorts

La première chose qu'on doit déclarer dans une spécification ce sont les types (généralement appelés *sortes* dans les spécifications algébriques) de données. Les sortes sont organisées via une relation de sous typage *subsort***.**

Une sorte est déclarée en utilisant le mot clé *sort***,** suivi par un identificateur (le nom de la sorte) suivi par un espace (un blanc) et d'un point, comme suit :

sort (sort) .

Plusieurs sortes peuvent être déclarées en utilisant le mot clé **sorts,** comme suit :

sorts (sort1) (sort2) . . . (sortn) .

Exemple : sort Zéro .

sort Nznat .

sort Nat .

Ou bien :

sorts Zéro Nznat Nat .

Les identificateurs utilisées pour les sortes ne peuvent pas contenir aucun des caractère ':', '.', ',', '[', ou ']' .

La relation de subsort sur des sortes est au même titre que la relation de sous-ensemble sur les ensembles dans le modèle prévu de ces sortes. Les inclusions de subsort sont déclarées en utilisant le mot-clé *subsort* :

subsort (Sort1) < **(Sort2) .**

On dit que la sorte (Sort1) est une subsort de(Sort2). Par exemple les déclarations :

subsort Zéro < Nat .

subsort NzNat < Nat .

Indique que les sorts **Zéro** (contenant seulement la constante 0) et **NzNat** (les nombres naturels différents de zéro) sont des subsorts de **Nat** (les nombres naturels). Plusieurs relations de subsort peuvent être déclarées en utilisant le mot clé **subsorts** , comme suit :

subsorts (Sort1) . . . (Sorti1) < . . . < (Sortn1) . . . (Sortnin) .

Les déclarations ci-dessus peuvent être données dans une déclaration simple comme suit :

subsorts Zéro NzNat < Nat .

3. Déclaration d'opérateurs

Dans un module Maude, un opérateur est déclaré avec le mot-clé **op** suivi de son nom, suivi des deux points, suivis de la liste de sortes pour ses arguments, suivi du symbole'->' suivi de la sorte de son résultat, optionnellement suivie d'une déclaration d'attributs, suivi d'un espace (blanc) et d'un point. Ainsi la déclaration générale prend la forme suivante :

op (OpName) : (Sort0) . . . (Sortk) -> (Sorti) [(Attributs)] .

Voici quelques déclarations d'opérateurs :

op zero : -> Zero .

op s_ : Nat -> NzNat .

op sd : Nat Nat -> Nat .

ops _+_ _*_ : Nat Nat -> Nat .

Si la liste d'argument est vide, l'opérateur s'appelle une constante. Ainsi **zéro** est une constante.

Le nom de l'opérateur est une chaîne de caractères qui peut se composer de plusieurs identificateurs. Le tirait de soulignement (_) joue un rôle spécial dans ces chaînes de caractères. Si aucun caractère de soulignement n'est présent dans la chaîne de caractères de l'opérateur, l'opérateur est déclaré sous la forme de *préfixe*. Si les caractères de soulignement apparaissent dans la chaîne, alors leur nombre doit coïncider avec le nombre de sortes déclarées comme arguments de l'opérateur. L'opérateur est alors sous la forme de *mixfix.* Dans l'exemple ci-dessus les opérateurs **s_ , _ + _ , _ * _** sont sous la forme de *mixfix*.

4. Surcharge d'opérateur

Les opérateurs dans MAUDE peuvent être surchargés, c-à-d que nous pouvons avoir plusieurs déclarations d'opérateurs pour le même opérateur avec des arguments et des résultats différents. L'exemple suivant présente une surcharge de l'opérateur _ + _ :

op _+_ : NzNat Nat -> NzNat .

sort Nat3 .

ops 0 1 2 : -> Nat3 .

op _+_ : Nat3 Nat3 -> Nat3 .

5. Déclaration des Variables

Une variable est contrainte à s'étendre au-delà d'une sorte particulière. Des variables peuvent être déclarées dans MAUDE avec une syntaxe se composant d'un identificateur (le nom de la variable), de deux points (:) et d'un autre identificateur (exprimant la sorte). Par exemple, *N : Nat* signifie que la variable *N* est de sorte *Nat*.

Une variable peut également être déclarée dans un module en utilisant le mot clé **var** suivi d'un identificateur (le nom de la variable) suivi de deux points avec un espace (blanc) avant et après, suivi d'un identificateur (sa sorte), suivi d'un espace (blanc) et d'un point :

Exemple : var N : Nat .

NB : lorsque on a plusieurs variables qui ont la même sorte on peut les déclarées avec le mot-clé **vars**

Exemple : vars : N M : Nat .

6. Les Termes

Un terme est une constante, une variable ou l'application d'un opérateur à une liste d'arguments.

7. Les commentaires

Les commentaires dans MAUDE sont écrits avec trois étoiles (***)

Exemple : **var N : Nat . *** N est une variable de sorte Naturel**

8. Les Modules

Dans Maude les unités de base de spécification et de programmation sont appelés les modules. Maude regroupe trois types de modules à savoir :

- Modules fonctionnels,
- Modules systèmes et
- Modules objets.

9. Module Fonctionnel

a. Définition :

Les modules fonctionnels définissent des types de données et les opérations qui en manipulent. De point de vue programmation, un module fonctionnel est un programme à caractère équationnel avec une syntaxe qui définit un certain nombre de sortes, leurs

éléments et les fonctions sur ces sortes. De point de vue de spécification, un module fonctionnel est une théorie équationnelle avec une sémantique algébrique.

Chaque module fonctionnel a un nom, qui est un identificateur dans MAUDE. Dans le cas d'un nom composé, les différentes parties sont reliées par un trait d'union.

La déclaration d'un module fonctionnel est comme suit :

fmod (Module Name) is (Declarations And Statements) endfm

b. Les équations inconditionnelles :

Les équations sont déclarées avec le mot-clé eq suivi d'un terme (la partie gauche), le signe d'égalité (=), puis un terme (la partie droite) et optionnellement suivi d'une liste d'attributs suivie par un espace et un point. Ainsi La déclaration est comme suit :

eq (Term-1) = (Term-2) [Statement Attributes] .

Les termes *t* et le *t'* dans une équation t = t' doivent tous les deux avoir la même sorte. Pour que l'équation soit exécutable, toute variable dans le terme *t'* doive également apparaître dans *t*.

Exemple :

eq N + zéro = N .

c. Les équations conditionnelles :

Les équations conditionnelles se composent de différentes équations t = t', une condition peut être une équation simple, ou une conjonction d'équations en utilisant la conjonction de connectivite binaire /\

La forme générale de l'équation conditionnelle est comme suit :

ceq (Term-1) = (Term-2)

if (EqCondition-1) /\ . . . /\ (EqCondition-k) [(StatementAttributes)] .

Exemple : Nous prenons l'exemple qui donne la valeur absolue d'un nombre entier.

op abso : Int -> Int .

var x : Int .

ceq abso(x) = x

if (x >= 0) .

ceq abso(x) = - x if (x < 0) .

d. Les Attributs d'opérateur :

Les déclarations d'opérateurs peuvent inclure les attributs qui fournissent des informations additionnelles au sujet de l'opérateur: sémantique, syntaxique, pragmatique, …etc. Les attributs sont déclarés dans une seule paire de crochets '[', ']' , après la sorte du résultat et avant le point de la fin.

Voici quelque type d'attribut pouvant être rencontrés dans MAUDE :

- assoc (associativité)
- comm (commutativité)
- idem (idempotence)
- iter (opérateur réitéré)
- ctor (constructeur).

Voici un exemple d'un module fonctionnel :

fmod Foot-Ball is

sort Team .

op _vs_ : Team Team -> Team [comm] .

ops Barcelone RealMadrid Juventus : -> Team [ctor] .

eq Barcelone vs RealMadrid = Barcelone .

eq Juventus vs RealMadrid = Juventus .

eq Barcelone vs Juventus = Juventus .

endfm

La validation de la spécification Maude représentée par un module fonctionnel se fait à l'aide de la commande ***red*** suivi de la première partie d'équation.

Exemple : red Barcelone vs RealMadrid .

Le résultat après la réécriture est Barcelone.

10. Module Systèmes

a. Définition :

D'un point de vue spécification, un module système est une théorie de réécriture avec un modèle sémantique initial. D'un point de vue programmation, est un programme concurrent de modèle déclaratif avec une syntaxe définissable par l'utilisateur [Cla09].

Un module système dans Maude indique une théorie de réécriture. Une théorie de réécriture a des sortes et des opérateurs et peut avoir des équations et règles, qui peuvent être inconditionnelles ou conditionnelles. Un module système est déclaré dans Maude en utilisant les mots-clés :

mod (Module Name) is (Declarations And Statements) endm

Le nom du module système est un identificateur, comme pour les modules fonctionnels, il est préférable de l'utiliser en majuscule, dans le cas d'un nom composé, les différentes parties sont reliées par un trait d'union (**-**).

Par exemple :

mod AUTOMATE-ETAT-FINI is

. . . * << partie déclaration et expression. >>**

endm

La partie *déclaration et expression* peut contenir différentes parties :

1) Importation des modules.
2) Déclaration des sorts et des subsorts.
3) Déclaration des opérateurs.
4) Déclaration des variables.
5) Les équations.
6) Les règles.

Les parties (1 à 5) sont exactement les mêmes dans les modules fonctionnels, il reste à savoir comment sont déclarées les règles conditionnelles et inconditionnelles.

b. Les Règles inconditionnelles :

Mathématiquement, une règle de réécriture a la forme ***t*** **=>** ***t'*** telle que ***t*** et ***t'*** sont des termes de la même sorte qui peuvent contenir des variables. Les variables, les sortes et les opérations utilisées dans la partie droite d'une règle doivent apparaître dans sa partie gauche. Intuitivement, une règle décrit une transition dans un système. Une règle inconditionnelle est présentée dans MAUDE avec la syntaxe suivante:

rl [(Label)] : (Term-1) => (Term-2) [(Statement Attributes)] .

c. Les Règles conditionnelles :

Une règle conditionnelle est présentée dans MAUDE avec la syntaxe suivante:

crl [(Label)] : (Term-1) => (Term-2)

if (Condition-1) /\ ... /\ (Condition-k) [(Statement Attributes)] .

Voici un exemple d'un module système qui choisit le premier nombre entier comme résultat parmi deux nombres :

mod CHOICE-INT is

including INT .

subsort Int < Configuration .

op _?_ : Int Int -> Int .

vars I J : Int .

rl [choose_first] : I ? J => I .

endm

La validation de la spécification Maude représentée par un module système se fait à l'aide de la commande ***rew*** suivie d'une configuration initiale.

11. Les Modules Orientés Objet

Dans la version Full Maude les systèmes concurrents orientés objet peuvent être spécifiés par des modules orientés objet. Ces derniers ont la structure suivante ***omod … endom*** en utilisant une syntaxe plus appropriée pour la description des systèmes orientés objet que celle utilisée dans les modules systèmes [Cla09].

Comme dans les modules systèmes les objets sont représentés comme suit :

< O : C | **attr-1, attr-2, … , attr-n** >

Mais dans les modules O.O la spécification de l'objet sera représentée comme suit :

< O : C | **a1 : v1, a2 : v2, … , an : vn** >

a. Les classe :

Les classes sont définies avec le mot-clé ***class,*** suivi par le nom de la classe, suivi d'une barre `| ', suivi d'une liste de déclarations d'attributs séparés par des virgules. Chaque déclaration d'attribut est de la forme ***a : S,*** où ***a*** représente l'identifiant de l'attribut et ***S*** représente le type(***Sort***). Donc la déclaration de la classe ayant la forme suivante :

class C | a1 : Sort-i, ... , an : Sort-n .

Exemples :

class Account | bal : Int .

class Person | name : String, age : Nat, account : Oid .

b. Message :

La syntaxe de la déclaration des messages est similaire à la déclaration des opérateurs mais en utilisant *msg* (pour déclarer un seul *message*) et *msgs* (pour déclarer plusieurs messages) au lieu de *op* et *ops*, et ont comme résultat la sorte ***Msg***.

Exemples :

msg from_to_transfer_ : Oid Oid Nat -> Msg .

msgs credit debit : Oid Nat -> Msg .

c. Héritage :

Maude tient compte du concept de l'héritage entre classes. Une classe *C1* peut hérité les propriétés d'une autre classe *C2* en utilisant le mot clé *subclass* comme suit: *subclass C1 < C2* . Donc tous les attributs, les messages et les règles de réécritures relatives à la classe mère en l'addition des nouveaux attributs déclarés dans la classe dérivée caractérisent la structure et le comportement des objets de cette dernière.

Exemples :

class SavingAccount | rate : Float .

subclass SavingAccount < Account .

Remarque : dans Full-Maude, les modules sont mis entre parenthèses

Exemple d'un module O.O :

(omod BANK-ACCOUNT is

protecting INT .

class Account | bal : Int .

msgs credit debit : Oid Int -> Msg .

var A : Oid .

vars M N : Int .

rl [credit] : < A : Account | bal : N > credit(A, M) => < A : Account | bal : N + M > .

crl [debit] : debit(A, M) < A : Account | bal : N > => < A : Account | bal : N – M >

if N >= M .

endom)

Remarque : l'exécution de la spécification écrite à l'aide d'un module O.O possède la structure suivante : (rew Configuration .)

12. Importation des modules

Comme dans la plupart des langages de programmation, un module peut importer un ou plusieurs autres modules. Le but de l'importation des modules c'est la minimisation de la taille de la spécification, et pour augmenter la réutilisation des composants [Cla09]. On a trois modes d'importation de modules : *protecting, extending et including*.

- ✓ **protecting MODULE NAME :** L'importation d'un module *M'* dans *M* en mode protecting signifie intuitivement qu'aucun ajout ou modification ne seront apportés au module M' (signifie essentiellement que les déclarations dans le module importé ne sont pas modifiables).
- ✓ **extending MODULE NAME :** Quelques données du module importé peuvent être étendues avec de nouveaux éléments non définis auparavant.
- ✓ **including MODULE NAME :** signifie qu'on peut changer le sens dans lequel les déclarations ont été employées.

IV. Conclusion

Nous avons présenté dans ce chapitre la logique de réécriture et le langage Maude. La logique de réécriture représente un cadre unificateur de tous les modèles formels qui expriment la concurrence, et le langage Maude représente l'implémentation d'une telle logique. Nous avons opté dans le cadre de ce mémoire pour le langage Maude pour les avantages qu'il procure. Maude est, en fait, utilisé dans le reste du manuscrit pour la description des besoins fonctionnels des systèmes multi-agents, et est aussi pour la validation par simulation de telle description.

CHAPITRE III :

Les Méthodologies de Développement des Systèmes Multi-Agents : une Étude Comparative

Chapitre III :

Méthodologies de Développement des Systèmes Multi-Agents : Une Étude Comparative

I. Introduction

Bien qu'il y ait actuellement un regain pour les techniques et les méthodologies de modélisation et de conception des systèmes multi-agents (SMAs), le développement de ces derniers engendre d'énormes problèmes et reste donc un domaine ouvert. Pour que la technologie orientée-agent puisse connaître un véritable succès, il faut une méthodologie rigoureuse dans les différentes phases du processus de développement de systèmes informatiques multi-agents. Les méthodologies de développement des SMAs peuvent être comparées puisqu'elles emploient les mêmes concepts principaux : état mental, tâches, interactions et modélisation de groupe [Woo98].

Nous nous intéressons dans ce chapitre à la comparaison des méthodologies de développement des SMA utilisant les diagrammes de cas d'utilisation pour l'analyse des besoins.

II. Étude Comparative des Méthodologies

- Méthodologies choisies pour établir une étude comparative des méthodologies, nous avons choisi trois méthodologies. Notre choix s'est basé sur les méthodologies qui utilisent les cas d'utilisation d'UML pour formaliser les besoins fonctionnels d'un SMA[2].

[2] Les trois méthodologies qui utilisent les cas d'utilisation pour formaliser les besoins fonctionnels d'un SMA sont : MAS-CommonKADS, MaSE et Prometheus (Voir chapitre I)

- Quant aux critères de comparaison, ces derniers couvrent un ensemble de valeurs. La figure suivante montre les critères sur lesquels nous nous sommes basés pour faire l'étude comparative.

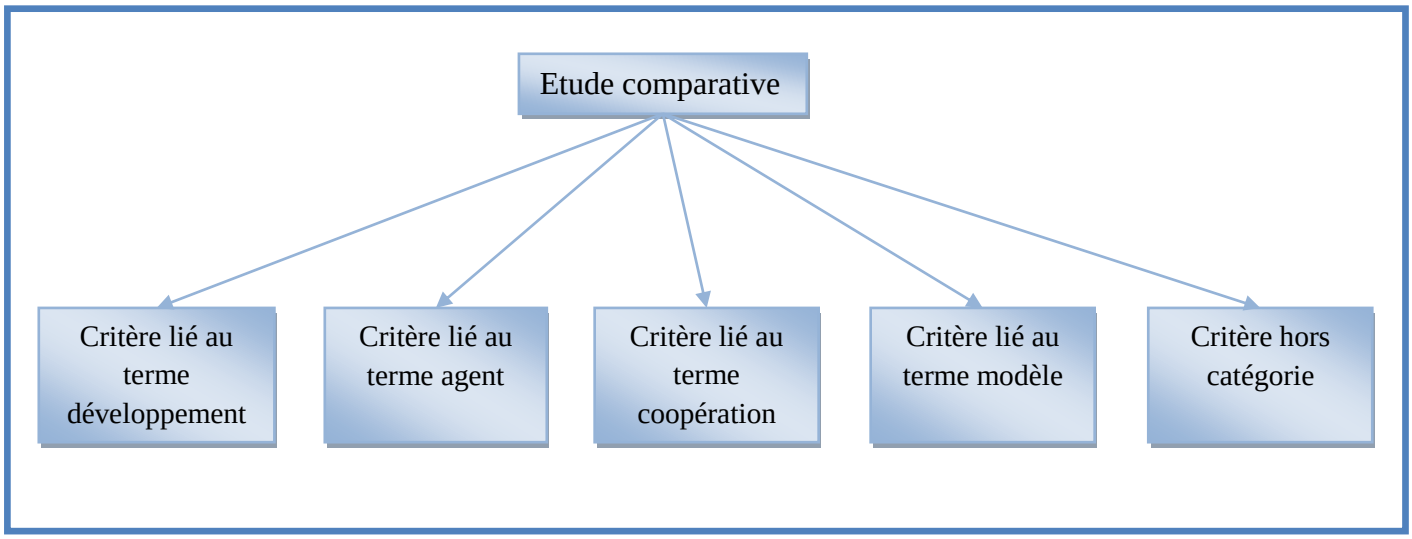

Figure 11 : Structure de critères de comparaison

II.1. Catégorie Développement

Cette catégorie regroupe la plupart des critères liés au terme développement. Il nous permet d'évaluer le processus de développement de chaque méthodologie. Cette catégorie enveloppe quatre critères à savoir :

1. Le processus de développement

C'est un critère important qui représente les étapes classiques du processus de développement du Génie Logiciel, ces étapes représentent un ensemble d'activités et des résultats associés qui produisent des logiciels, chaque activité utilisent et produisent des documents. Ces critères regroupent six activités à savoir : [Ada99]

- ensemble d'activités dont la finalité réside dans l'expression du besoin ou du problème (son but c'est d'éviter de produire un logiciel qui ne répond pas au besoin d'utilisateur). Étude du domaine d'application (phase exploratoire).
- **modélisation** : processus permettant de représenter les données issues de l'analyse. Élaboration de modèles.

- **spécification :** description de ce que doit faire le logiciel en évitant des décisions prématurées de réalisation.

- **conception :** ensemble d'activités dont la finalité est d'élaborer une solution à partir d'un problème déjà modélisé. Description de l'architecture du logiciel; spécification de ses divers composants; description pour chaque composant de la manière dont ses fonctions sont réalisées.
- **validation :** compatibilité entre l'interprétation du modèle par l'utilisateur (client) et le point de vue de celui-ci. A-t-on décrit le bon produit?
- **vérification :** processus qui permet de tester le produit. A-t-on un produit correct?

2. Modèles de développement

Ce sont les modèles du Génie Logiciel sur lesquels se base le cycle de développement d'une méthodologie [Ada99].

- **le modèle cascade :** c'est une approche de développement séquentielle dans laquelle il n'y a pas chevauchement des étapes de développement au cours d'un projet.

- **le modèle en V :** c'est une variante du modèle en cascade, qui montre la symétrie et la complémentarité qui existe entre les phases de production et les différentes phases de test (pré-test et test).

- **le modèle en spirale :** c'est une approche de développement qui consiste à ajouter à chaque étape du modèle cascade une phase de prototypage et une revue technique (validation/révision) avec les utilisateurs. Cette tâche de validation consiste à vérifier la conformité du modèle aux besoins de l'utilisateur.

- **le modèle incrémental (ou évolutif) :** c'est une approche de développement qui consiste à faire une extension progressive des fonctionnalités du système à partir de l'étape de la conception détaillée. Le développement par incrément permet le parallélisme [Sab01].

3. Approche de développement [Ada99]

Ce critère permet d'identifier l'approche suivie par la méthodologie de développement de SMA. Celle-ci peut être : [Sab01]

- **descendante (ou top-down) :** approche qui supporte la construction par module et qui est fondée sur la décomposition fonctionnelle du problème du général au particulier, et le raffinement graduel de celui-ci.
- **ascendante (ou bottom-up) :** c'est une approche par assemblage de composants ou de composition. On s'intéresse aux différents composants, puis on remonte progressivement par composition.
- **évolutive :** on démarre par les sous-ensembles de composants, ensuite on peut soit descendre au niveau bas, soit remonté au niveau supérieur.

4. Disponibilité des outils

Ce critère indique s'il existe des outils qui supportent la méthodologie (existence de bibliothèques d'agents, de composantes d'agents, d'organisations, d'interactions, d'environnement, de supports techniques ou méthodologiques). L'existence des outils qui supporte une méthodologie la rend plus opérationnelle [Sab01].

II.2. Catégorie Agent

Cette catégorie regroupe un certain nombre de critère concernant le terme agent à savoir leurs natures, types, attributs. Ces derniers constituent un facteur important pour leur comportement social et coopératif.

1. Nature d'agent [Afi98]

Ce critère indique si les agents présents dans les modèles de la méthodologie sont :

- **homogènes :** de même nature. Tous les agents sont construits sur le même modèle (exemple : une colonie de fourmis).
- **hétérogènes :** de nature différente. Les agents sont construits avec des modèles différents, de granularité différente.

2. Type d'agent

Ce critère indique les types d'agents que la méthodologie permet de représenter [Rob99].

- **agents intelligents (ou BDI) :** qui sont vus comme des entités qui imitent les processus mentaux ou simulent des comportements rationnels (exemples : les robots footballeurs [Col96]).
- **agents interfaces ou personnels :** ce sont des entités qui assistent les utilisateurs dans l'exécution d'une tâche (exemple : le système de gestion du trafic aérien dans [Kin96]). Ils fournissent une assistance proactive à l'utilisateur pour une application précise.
- **agents mobiles :** qui sont des entités capables d'errer dans des environnements en réseau pour accomplir leurs objectifs (exemple : un virus).
- **agents d'information :** qui sont des agents spécialisés pour filtrer et pour organiser, de façon cohérente les données dispersées et sans lien (exemple : un moniteur de la page Web qui prend en charge les demandes d'informations, effectue les attributions de ressources et contrôle l'enchaînement des tâches [Sco99]).

3. Agents autonomes

Qui sont capables d'accomplir des actions non supervisées (responsables de leurs comportements). Il est capable d'agir sans l'intervention d'un tiers [Rob99].

4. Caractéristique d'agent

Ce critère représente un ensemble de valeurs qui décrit les caractéristiques d'un agent utilisé par la méthodologie [Rob99].

- **adaptabilité :** habileté à apprendre et à s'améliorer avec l'expérience, acquisition dynamique des connaissances.
- **autonomie :** objectif non dirigé, comportement proactif et d'auto démarrage (selfstarting behaviour). Degré d'autonomie sociale : indépendant, semi-autonome, contrôlé.

- **comportement coopératif :** habileté à travailler avec d'autres agents pour atteindre un objectif commun. Degré de coopération : coopératif, compétitif, antagoniste.
- **capacité déductive :** habileté à agir sur des spécifications de tâches abstraites. Un agent peut formuler des hypothèses sur ces tâches et établir des conclusions.
- **habileté de communication :** habileté à communiquer avec d'autres agents avec un langage qui ressemble beaucoup plus aux actes de discours humains que des symboles de programmes (les actes de discours indiquent les actions intentionnelles effectuées au cours d'une communication).
- **mobilité :** habileté à émigrer dans une direction désirée, d'une plate-forme à une autre.
- **personnalité :** habileté à manifester des attitudes d'un caractère humain (un agent peut être égoïste, jaloux, etc.).
- **réactivité :** habileté à réagir selon les états de son environnement.

II.3. Catégorie Coopération

La coopération est l'une des plus importantes rubriques des SMAs où les agents doivent coopérer pour atteindre un objectif commun. Plusieurs auteurs (par exemple [Gol94], [Sek95]) ont étudié l'influence du comportement social des agents sur la performance globale du système. Ils ont montré que la coopération entre agents améliore les résultats. Hogg et Huberman [Hog93] ont montré que lorsque des agents coopèrent dans le cadre d'une résolution distribuée de problème, ils le résolvent plus rapidement que n'importe quel agent travaillant isolément.

1. Type de contrôle

Ce critère indique le type de coordination utilisé dans les modèles d'interaction de la méthodologie [Afi98].

- **centralisé :** le contrôle est assuré par un seul agent (exemple : superviseur-employés).

- **hiérarchique :** c'est une direction dans laquelle il y a une série ascendante de pouvoirs ou de décisions (la coordination dans les entreprises par exemple).
- **distribué :** c'est une direction répartie; le processus de décision est conjoint (le marché par exemple).

Dans tous les cas, les agents doivent être capables d'échanger entre eux des résultats intermédiaires et de faire des transferts de ressources. La coordination est une question centrale pour les SMA. Sans la coordination, un groupe d'agents peut dégénérer rapidement en une collection chaotique d'individus [Cha99].

2. **Mode de communication** [Sab01]

Ce critère indique la prise en compte de la communication de données, du mode de communication qui peut être :

- **direct :** envoi de message.
- **indirect :** utilisation de tableau noir. Les agents accèdent à une base de données partagée appelée tableau noir dans laquelle les informations sont portées. Dans le système MINDS [Huh87], chaque usager travaille avec un agent qui utilise un tableau pour retrouver des documents dans sa base de données locale ou communique par message avec d'autres agents du même type.
- **synchrone :** par téléphone par exemple.
- **asynchrone :** la messagerie par exemple.

3. **Langage de communication** [Afi98]

Ce critère indique le langage utilisé par les agents :

- **les signes :** par exemple le langage de communication des fourmis est basé sur les signes.
- **les actes de discours :** KQML par exemple.

- **autres :** il se peut qu'il existe un langage de communication qui soit différent des deux premiers.

II.4. Catégorie Modèle

Cette catégorie couvre un ensemble de critères liés au terme modèle, qui nous permet d'évaluer les modèles utilisés dans chaque méthodologie. Sept critères ont été associés à cette catégorie à savoir : [Qyu05]

1. Concurrence

Est-ce-que la méthodologie peut supporter le traitement concurrent au niveau d'agent (ex., accomplir les buts concurrents ou la participation aux interactions concurrentes)?

2. Comportement coopérative

est-ce-que les modèles supportent et représentent le comportement coopératif d'agents (c.-à-d., la capacité de collaborer avec d'autres agents pour atteindre les buts communs)?

3. Réutilisabilité

Ce critère indique si la méthodologie fournit ou permet de fournir une librairie de modèles réutilisables. La fourniture de modèles réutilisables par une méthodologie la rend *générique*, ce qui évitera des pertes de temps.

4. Nombre de modèle

Nombre de modèles associés à chaque méthodologie**.**

5. Cohérence de modèle

Pas de redondance. Tout ce qui se rapporte à un même concept se range sous une même entité; ou bien tout ce qui se trouve sous une même entité se rapporte à un même concept [Sab01].

6. Complétude des modèles

L'ensemble des modèles produits doit rendre compte de la totalité de l'univers du discours. Est-ce que l'ensemble des modèles couvrent tous les aspects d'un SMA [Sab01].

7. Complexité de modèle

Inter-relation entre les modèles (couplage); le couplage est bien sûr nécessaire mais il doit cependant être limité au strict minimum (un couplage trop élevé rendra plus difficile la validation et la vérification des modèles); nombre d'éléments à maîtriser pour construire les modèles [Sab01].

II.5. Critères hors Catégorie

Regroupe un ensemble de critères qu'on peut ajouter aux critères définis précédemment afin de nous donner un plus pour évaluer les méthodologies de conception des SMAs.

1. Les taxonomies de spécification

Ce critère défini le type de la spécification utilisée dans la phase analyse de besoin. Il prend trois valeurs à savoir : *informel* (ce sont des spécifications qui sont écrites uniquement en langage naturel), *semi-formel* (spécification ayant l'aspect graphique accompagner du texte) et *formel* (les spécifications sont écrites dans un langage qui a une syntaxe et une sémantique bien déterminées ex : Maude).

2. Système ouvert ou clôt

Ce critère possède deux valeurs à savoir : *ouvert* (signifie que la méthodologie supporte la notion des systèmes ouverts c.-à-d l'ajout et la suppression dynamique des agents) ou *fermer* (système clôt c.-à-d le nombre d'agent est connue a priori).

3. Supporter la notion d'ontologie

Est-ce-que la méthodologie fournit un support pour l'utilisation et la spécification d'ontologie dans les SMAs (système d'agent basé ontologie) ?

4. Approche supportée par les méthodologies

Ce critère fournis un seul sous-critère qui permet d'identifier l'approche utilisée dans le développement du SMA.

- Approche de développement utilisée soit : ***approche Orienté Objet*** (OO) ou approche orientée ***ingénierie de connaissances.***

5. Taille du SMA

Quelle est la taille du SMA que la méthodologie doit supporter. Ce critère possède trois valeurs : *petit, moyen, large*.

6. Domaine d'application

est-ce-que la méthodologie est applicable à n'importe quel domaine (i.e., domaine indépendant) ou a un domaine bien spécifié (i.e., domaine dépendant).

7. Classification des méthodologies

Alonso à définit trois classes des méthodologies de conception des SMA à savoir : (1) les méthodologies qui sont issues de l'OO, (2) qui sont issues de l'ingénierie de connaissances et (3) qui sont basé agent. Donc ce critère peut être soit : OO (Orienté Objet), IC (Ingénierie de Connaissances) ou BA (Basé Agent) [Alo04].

III. Comparaison des méthodologies

Dans cette section nous appliquons l'ensemble de critère cité précédemment aux trois méthodologies. Il s'agit de :

1. Analysis and Design of Multiagent Systems Using MAS-CommonKADS [Fis97].
2. Multiagent Systems Engineering (MaSE)[Del01].
3. La méthodologie Prometheus [Pad02].

III.1. Catégorie développement

La comparaison des trois méthodologies tout en se basant sur les critères relatifs a la catégorie développement est illustrée par la table 2.

Légende

Oui : signifie que la méthodologie a pris en compte cette valeur.

Non : signifie que la méthodologie n'a pas pris en compte cette valeur.

Blanc : signifie qu'on ne peut rien conclure sur la base des documents que nous avons lus.

P (possible) : signifie qu'au regard des éléments décelés dans les documents, on peut déduire que la méthodologie pourrait prendre en compte cette valeur.

		Méthodologies qui utilisent les cas d'utilisation		
Critères	Valeurs	MAS-CommonKADS	MaSE	Prometheus
Etape de processus	Analyse	oui	oui	oui
	Modélisation	oui	oui	oui
	Spécification	oui	oui	oui
	Conception	oui	oui	oui
	Validation		non	
	Vérification	oui	possible	
Modèles De développement	cascade	non	non	
	Incrémentale	non	oui	
	spirale	oui		
	V	non	non	
Approche De développement	Descendante	oui	oui	
	ascendante	oui	oui	oui
	évolutive	non	oui	
Disponibilité Des Supports Logciels	Analyse	OOSE, MSC, RDD, SDL, HMSC, CML, OMT, CommonKADS	AgML	OOSE
	Modélisation	OOSE, MSC, RDD, SDL, HMSC, CML, OMT, CommonKADS	AgML	OOSE
	Spécification	OOSE, MSC, RDD, SDL, HMSC, CML, OMT, CommonKADS	AgML	OOSE
	Conception		AgML, AgDL, UML	OOSE, AUML
	Validation			
	Vérification		AgDL	

Table 2 : Comparaison selon les critères liés au terme processus de développement

III.2. Catégorie agent

La table 3 donne la nature, les caractéristiques et la nature des agents du système au quel la méthodologie s'applique :

		Méthodologies qui utilisent les cas d'utilisation		
Critères	**Valeurs**	MAS-CommonKADS	MaSE	Prometheus
Nature d'agent	Homogène	non	non	non
	Hétérogène	oui	p	p
Type d'agent	agents intelligents (ou BDI)	oui	p	oui
	agents interfaces ou personnels	oui	p	
	Agent Mobile	oui	p	
	Agent d'information	oui	oui	
Caractéristique d'agent	adaptabilité	p	p	p
	autonomie	oui	p	p
	comportement coopératif	oui	p	oui
	capacité déductive	oui	p	oui
	habileté de communication	oui	p	oui
	mobilité			
	Réactivité	oui	p	oui
	Personnalité	oui	p	

Table 3 : Comparaison selon les critères liés au terme agent

III.3. Catégorie coopération

La table 4 indique les notions de coopération utilisées dans le système que la méthodologie peut représenter.

		Méthodologies qui utilisent les cas d'utilisation		
Critère	**Valeurs De Critère**	MAS-CommonKADS	MaSE	Prometheus
Modes de communication	Direct	oui	oui	p
	indirect		non	p
	synchrone	oui	p	p
	asynchrone	oui	p	p
Langage de communication	Signes	oui	p	
	actes de discours	oui	p	oui
	Autres	non	non	non
Type de contrôle	centralisé	p		
	hiérarchique	oui		
	distribué	oui	p	p

Table 4 : Comparaison selon les critères liés au terme coopération

III.4. Catégorie modèle

Chaque méthodologie utilise des modèles pour la modélisation des SMA et chaque modèle à sa propre notation. Cette catégorie permet de faire une évaluation de ces méthodologies selon un certain nombre de critères qui sont liés au terme modèle. La table 5 représente cette évaluation.

	Méthodologies qui utilisent les cas d'utilisation		
Méthodologies / **Critères**	MAS-CommonKADS	MaSE	Prometheus
Concurrence	Non	Oui	Non
Réutilisabilité	P	Oui	
Nombre de modèle	7	4	
Cohérence de modèle	P	P	P
Complétude des modèles	P	Non	P
Complexité de modèle	15	6	

Table 5 : Comparaison selon les critères liés au terme modèle

III.5. Critères hors Catégorie

La table 6 montre la comparaison entre les trois méthodologies selon les critères hors catégorie définis précédemment.

Légende

Info : signifie que la méthodologie utilise des spécifications informelles.

S-Form : signifie que la méthodologie utilise des spécifications semi-formelles.

Form : signifie que la méthodologie utilise des spécifications formelles.

OO : signifie que la méthodologie utilise l'approche Orienté Objet.

IC : signifie que la méthodologie utilise l'approche Ingénierie de connaissance.

BA: signifie que la méthodologie est basé agent (prend en considération le paradigme agent).

Dép: domaine d'application de la méthodologie est dépendant.

Indé : domaine d'application de la méthodologie indépendant.

NS : la taille des agents dans le système est non spécifiée.

	Méthodologies qui utilisent les cas d'utilisation		
Méthodologies / **Critères**	MAS-CommonKADS	MaSE	Prometheus
taxonomies de spécification	S-Form	S-Form	S-Form
Système ouvert	Non	Non	Non
Supporter la notion ontologie	Oui	Non	Non
Approches utilisées	IC & OO	OO	OO
Taille du SMA	NS	<= 10 classes d'agent	N'importe quelle taille
Domaine d'application	Indé	Indé	Indé
Classification des méthodologies	IC	OO	BA

Table 6 : Comparaison selon les critères hors catégories

IV. Conclusion

Dans ce chapitre nous avons présenté un ensemble de critères qui à nous permis de faire une comparaison et évaluation des trois méthodologies qui utilisent les cas d'utilisation comme une technique pour la formalisation des besoins fonctionnelles des SMA. Le choix d'une méthodologie de conception SMA est liée au domaine d'application de la méthodologie (par-exemple : les méthodologies qui sont liées au domaine de la simulation). Certaines méthodologies sont adaptées aux applications bien déterminées (**Cassiopée** est une méthodologie dédié à l'application robot-footballeur).

CHAPITRE IV :

La Génération d'une Spécification Maude à partir

d'un Diagramme de Cas d'Utilisation d'UML

Étendu

Chapitre IV :

La Génération d'une Spécification Maude à partir d'un Diagramme de Cas d'Utilisation d'UML Étendu

I. Introduction

Nous présentons dans ce chapitre une approche générique permettant de translater les besoins fonctionnels d'un SMA décrits à l'aide de diagrammes de cas d'utilisation UML étendus et de diagramme de séquences AUML (Agent UML) [Bau01], dans une spécification formelle Maude. Le langage Maude, basé sur la logique de réécriture, offre des bases formelles et solides pour la spécification et la programmation des systèmes concurrents. Les principales motivations de ce travail sont : (1) la formalisation des besoins fonctionnels d'un système multi-agents à l'aide de Maude, et (2) l'intégration de la validation formelle de la cohérence des modèles, dès la phase d'élicitation des besoins, dans un processus de développement des SMAs.

L'approche proposée est structurée en trois étapes principales. La première étape consiste à la description des besoins fonctionnels d'un système multi-agents en utilisant les diagrammes de cas d'utilisation UML étendus. Nous apportons quelques extensions aux diagrammes de cas d'utilisation UML pour tenir compte de spécificités des SMAs. Les diagrammes de séquences AUML sont également utilisés dans cette étape pour la réalisation de différents scénarios du cas d'utilisation. Dans la deuxième étape, les diagrammes considérés doivent subir une validation inter-modèles afin de vérifier la consistance du système. La troisième étape consiste à la génération automatique d'une description Maude à partir des diagrammes considérés.

Avant de présenter notre approche proposée nous voulons donner un aperçu sur les taxonomies de spécification de logiciels.

II. Les taxonomies des approches de spécification

On distingue trois classes d'approches à savoir :

II.1 Les approches informelles

Les spécifications sont écrites uniquement en langage naturel, elles présentent des lacunes, nous citons entre [Mar96] :

- Elles posent en générale des problèmes de non cohérence, d'ambigüité, de non complétude et de difficulté d'organisation.
- Risque de redondance des informations surtout dans le cas des systèmes complexes.

II.2 Les approches semi-formelles

Les spécifications dans ces approches sont représentées sous forme graphique. Le formalisme graphique introduit un aspect formel mais il est en générale accompagné de textes informels, c'est pour cela qu'on désigne ces techniques comme semi-formelles [Mar96].

II.3 Les approches formelles [Pet05]

Bien qu'elles soient assez difficiles à comprendre, notamment pour ceux qui ne sont pas bien familiarisés avec l'approche ou le langage utilisé [Pet05], les approches formelles ont plusieurs avantages, elles permettent :

- De prouver que les programmes sont corrects puisqu'elles sont basées sur une logique mathématique.
- La validation des composants de la spécification [Hin95].
- La vérification du passage d'une spécification à une autre.
- Une compréhension approfondie du système [Mar96].

III. Diagrammes utilisés

III.1. Diagramme de cas d'utilisation

Les utilisateurs sont souvent des non informaticiens qui ont besoin de moyens simples qui leur permettent d'exprimer leurs besoins. C'est précisément le rôle du diagramme de cas d'utilisation d'UML. Ce dernier capture le comportement d'un système, d'un sous-système, d'une classe ou d'un composant tel qu'un utilisateur extérieur le voit. Il décompose la fonctionnalité du système en unités cohérentes, les cas d'utilisation, ayant un sens pour les acteurs. Les cas d'utilisation permettent d'exprimer le besoin des utilisateurs d'un système, ils sont donc une vision orientée utilisateur de ce besoin au contraire d'une vision informatique.

Les cas d'utilisation d'UML possèdent trois types de relation à savoir : relation d'inclusion, relation d'extension et la généralisation.

1. L'aspect sémantique de la relation d'inclusion

La relation d'inclusion entre deux cas d'utilisation signifie que le comportement du deuxième cas est inclus dans le comportement du premier. Donc l'utilisation de cette relation est prévue lorsqu'il y a des parties communes du comportement de deux ou de plusieurs cas d'utilisation. L'inclusion nous permet de décomposer un cas d'utilisation complexe en sous cas plus simples. Cette relation est symbolisée par le stéréotype «include» [Lau07, OMG07].

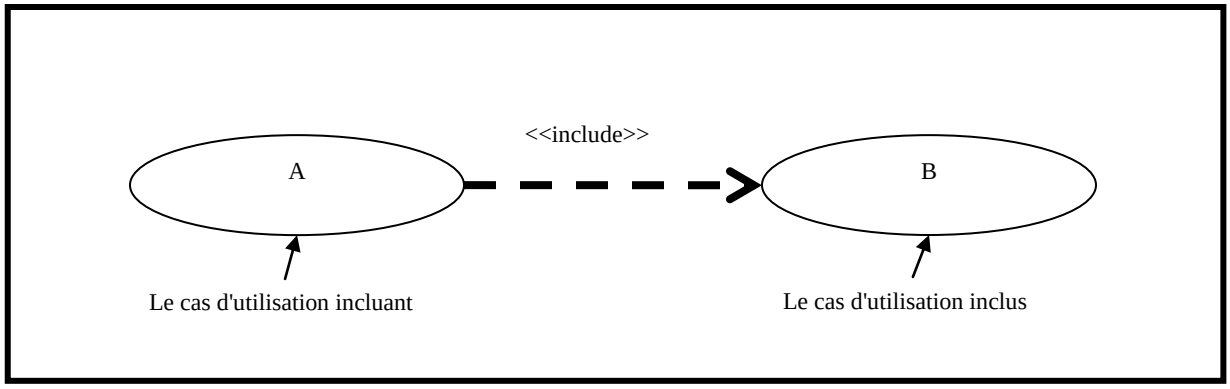

Figure 12 : La relation d'inclusion

2. L'aspect sémantique de la relation d'extension

La relation d'extension sert à enrichir un cas d'utilisation par un autre. Cet enrichissement est analogue à celui de la relation d'inclusion mais il est optionnel. L'extension se fait dans le

cas d'utilisation de base, en des points précis et prévus lors de la conception, appelés points d'extension [OMG07].

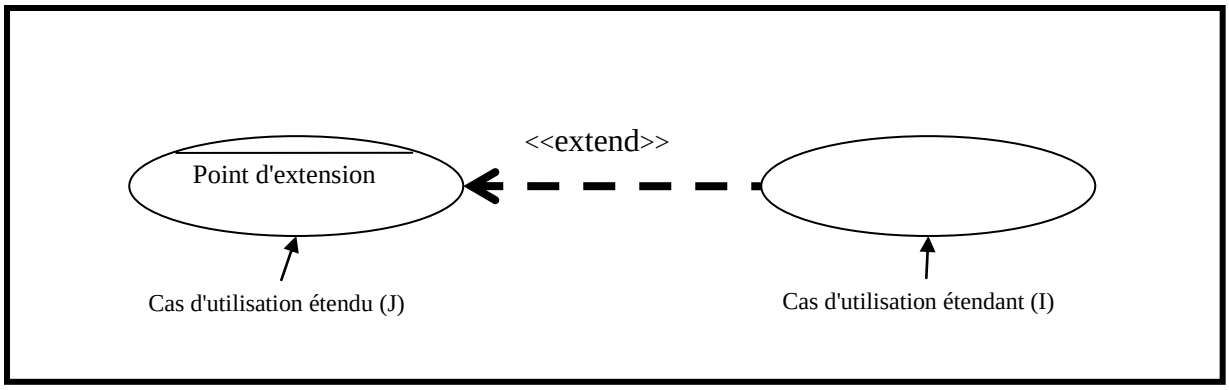

Figure 13 : la relation d'extension.

3. L'aspect sémantique de la relation Généralisation

Un cas A est une généralisation d'un cas B si B est un cas particulier de A. Cette relation de généralisation/spécialisation est présentée dans la plupart des diagrammes UML et se traduit par le concept d'***héritage*** dans les langages orientés objet [Lau07].

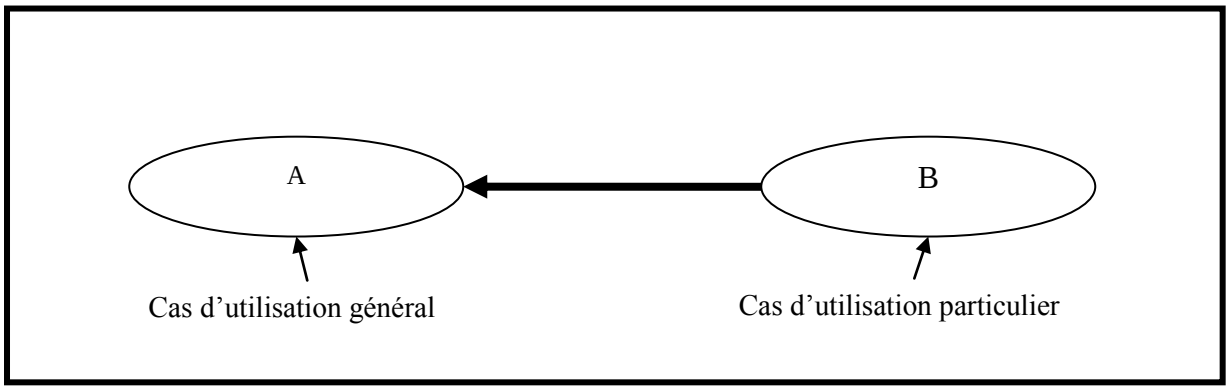

Figure 14 : la relation généralisation

III.2. Diagramme de séquence AUML

Le digramme de séquence AUML décrit les interactions entre agents. Il étend les diagrammes de séquence UML par l'introduction de quelques extensions supportant les envois de messages concurrents. Pour décrire les threads d'interaction, AUML a introduit trois façons permettant d'exprimer les threads multiples (figure 15). Figure 15(a) indique que tous les actes de communication CA-i (*CA-1,...CA-n)* sont envoyés en concurrence. Figure 15(b) présente le *'ou inclusif'* qui inclut une boîte de décision qui permet aux actes de communication CAs (zéro ou

plusieurs) d'être envoyés. Figure 15(c) présente le *'ou exclusif'* qui indique qu'un et un seul CA doit être envoyé.

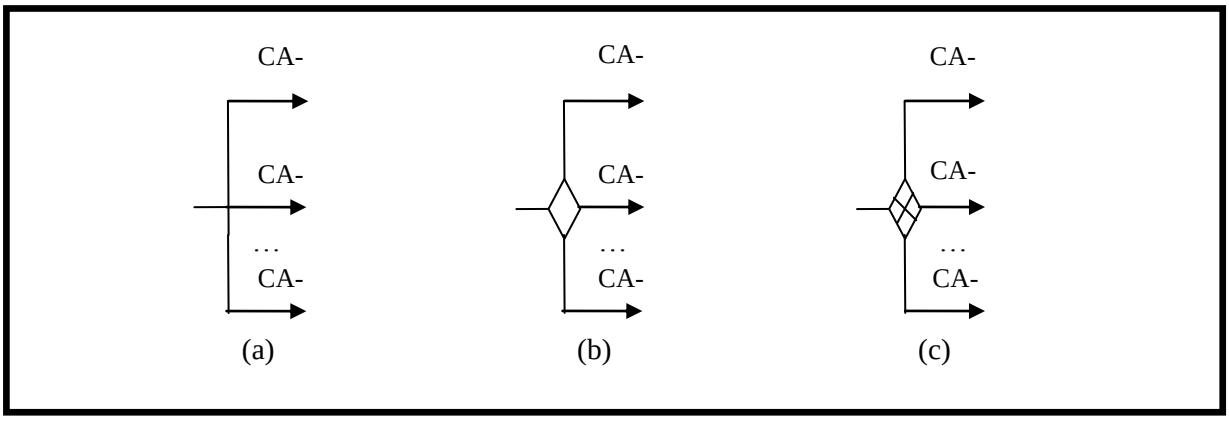

Figure 15 : Extensions recommandées supportant les threads concurrents d'interaction

IV. L'approche proposée

Dans cette section nous présentons notre approche qui nous permet d'obtenir une spécification formelle écrite en langage Maude. Le processus de cette approche est décrit comme suit : nous avons dans un premier temps étendu les diagrammes de cas d'utilisation d'UML afin de tenir compte des spécificités des systèmes multi-agents. Dans un second temps, les diagrammes de séquences AUML ont été utilisés pour décrire le comportement collectif entre les agents en interaction. Une fois qu'on a établi tous les diagrammes (cas d'utilisation et séquence AUML), une vérification suivie d'une validation doivent être réalisées pour assurer la cohérence inter et intra modèles. Finalement c'est la translation de la description semi-formelle, modélisée avec les diagrammes utilisés précédemment, en une spécification formelle implémentée en langage Maude.

IV.1. Extension des diagrammes de cas d'utilisation

Nous présentons dans cette partie quelques extensions que nous jugeons utiles aux diagrammes de cas d'utilisation d'UML pour tenir compte de quelques spécificités des SMA, et ce dans l'objectif de décrire les besoins fonctionnels des tels systèmes. La première extension proposée est une notation utilisée pour décrire les agents impliqués dans le système. Il s'agit de considérer un agent comme un acteur interne au système, c.-à-d, on peut associer au diagramme de cas d'utilisation d'UML un ensemble d'acteurs internes jouant les rôles des agents. Donc on distingue deux types d'acteurs au sein d'un même diagramme de cas d'utilisation étendu, à savoir,

les acteurs internes (les agents qui constituent le système) et les acteurs externes au système (les acteurs qui représentent toutes les entités externes au système).

Un cas d'utilisation est une interaction entre un acteur (acteur externe) et le système. Dans un contexte multi-agents, ça peut être aussi, une interaction entre agents (acteurs internes). Donc toute interaction entres agents sera spécifiée par un cas d'utilisation stéréotypé par l'ajout du stéréotype « Agent use case ».

1. **Les notations utilisées**

a) **Acteur interne :** c'est l'entité qui joue le rôle d'un agent dans le système. Sa notation (figure 16) est similaire à celle des acteurs externes sauf que la tête doit être carrée. En effet, cette notation est inspirée de la méthodologie MAS-CommonKADS [Igl97].

Figure 16 : Acteur interne (Agent)

b) **Acteur externe :** le même concept d'acteurs utilisés dans les diagrammes de cas d'utilisation d'UML. Sa notation est illustrée par la figure 17**.**

Figure 17 : Acteur externe

c) **Les cas d'utilisation :** Les cas d'utilisation sont représentés dans les digrammes de cas d'utilisation par la notation de la figure 18. Elle est la même notation que celle d'UML tout en y ajoutant le stéréotype « Agent use case ».

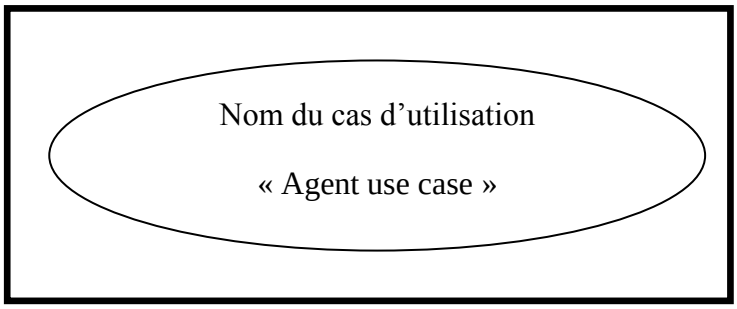

Figure 18 : Interaction entre agents

IV.2. Le passage vers le diagramme de séquence AUML

Les cas d'utilisation décrivent les différentes fonctionnalités de futur produit logiciel à un haut niveau d'abstraction élevé. La réalisation de ces fonctionnalités, est accomplie à l'aide de diagrammes d'interaction pour capturer les différents scénarios possibles. Parce que les diagrammes de séquences UML ne tiennent pas compte de spécificités des agents, nous avons opté pour les diagrammes de séquence AUML pour la réalisation de différents cas d'utilisation décrivant les besoins fonctionnels du système.

Le passage de diagramme de cas d'utilisation vers le diagramme de séquence AUML se fait selon les cas suivants :

1. Un seul cas d'utilisation

Chaque cas d'utilisation est réalisé par un digramme de séquence AUML.

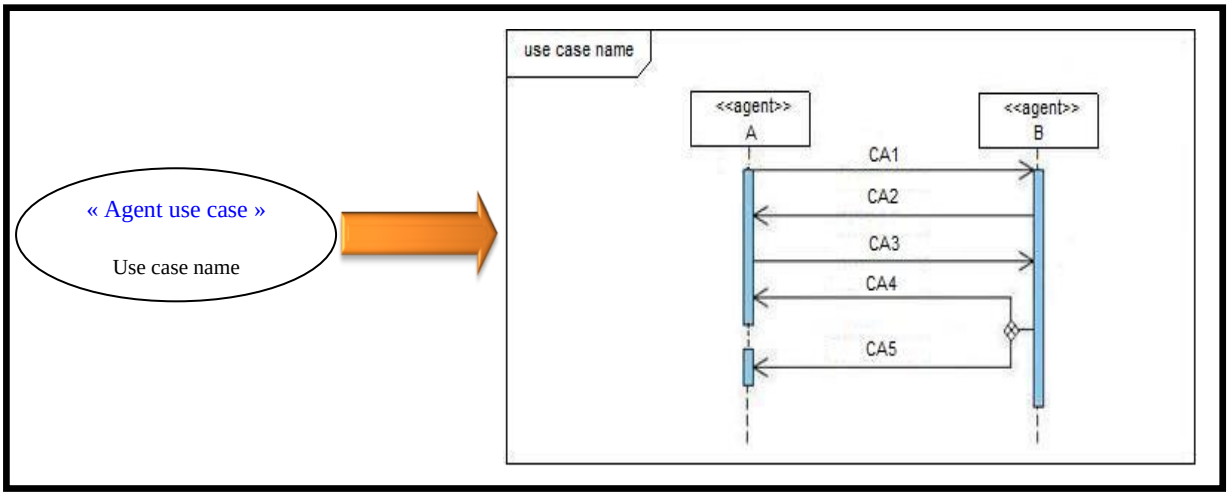

Figure 19 : Réalisation d'un cas d'utilisation à l'aide diagramme de séquence AUML

2. La relation inclusion stéréotypée « include »

Nous avons montré précédemment que l'aspect sémantique de la relation d'***inclusion*** signifie l'inclusion du comportement d'un cas d'utilisation vers un autre. Donc on peut regrouper deux cas d'utilisation qui sont reliés avec la relation « *include* » dans un seul diagramme de séquence AUML, en utilisant l'opérateur de référence ***ref***[3] dans le diagramme de séquence AUML (figure 20).

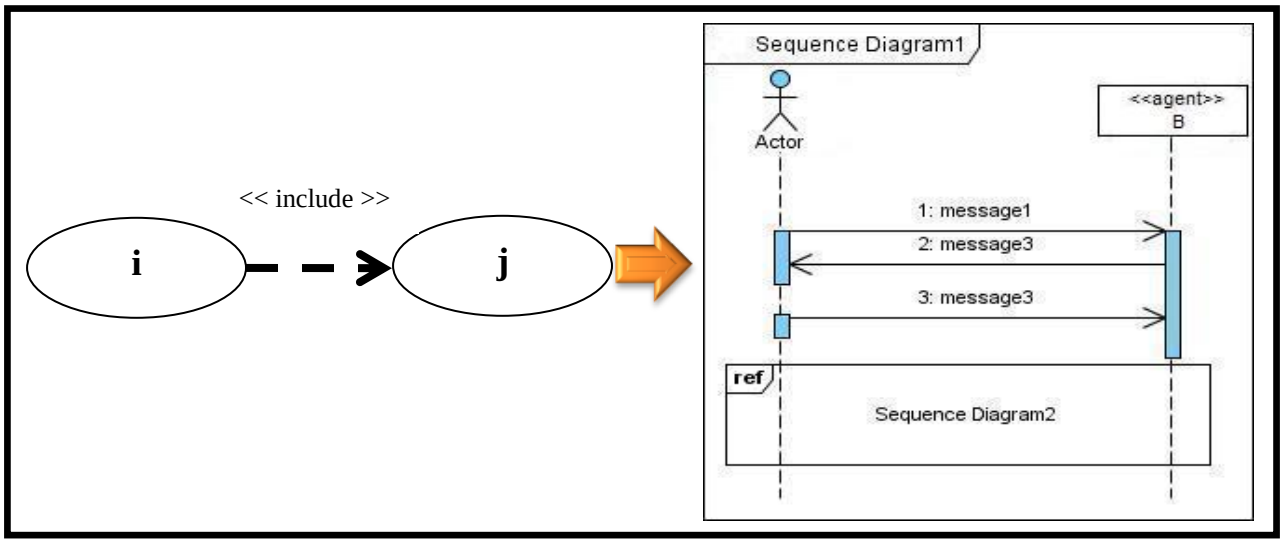

Figure 20 : Description de la relation inclusion à l'aide de diagramme de séquence AUML

NB : les scénarios des cas d'utilisation i et j sont représentés avec le diagramme de séquence AUML 1 et 2 respectivement.

3. La relation extension stéréotypée « extend »

On dit qu'un cas d'utilisation A étend un cas d'utilisation B lorsque le cas d'utilisation A peut être appelé au cours de l'exécution du cas d'utilisation B. L'extension peut intervenir à un point précis du cas étendu. Ce point s'appelle le point d'extension. Éventuellement associé à une contrainte indiquant le moment où l'extension intervient, une extension est souvent soumise à condition. Graphiquement, la condition est exprimée sous la forme d'une note. Donc l'exécution de scénario du cas d'utilisation ***étendant*** et lié à la satisfaction de la condition représentée dans le point d'extension. Donc le passage vers le diagramme de

[3] Une référence peut être vue comme un pointeur ou un raccourci vers un autre diagramme de séquence existant. (Voir figure 9)

séquence AUML se fait à l'aide de l'opérateur alternative ***alt***[4] combiné à celui de référence ***ref*** (figure 21)

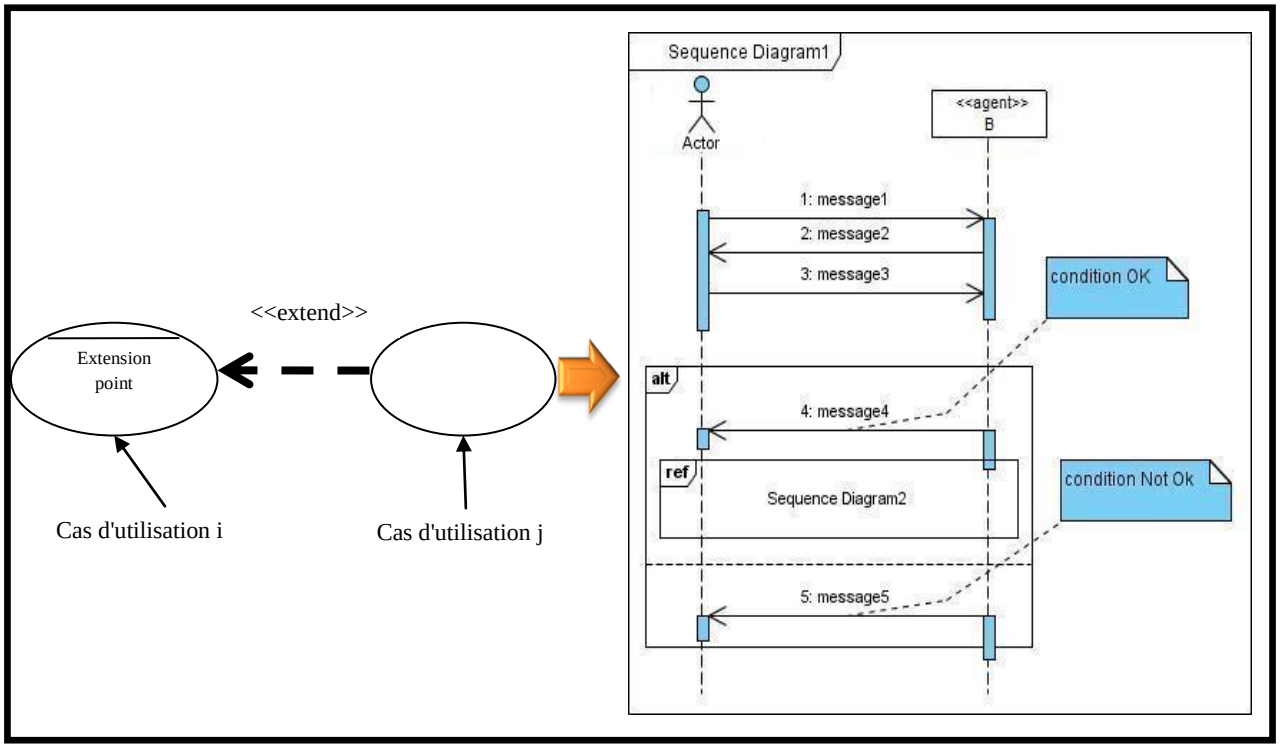

Figure 21 : la relation extension vers le diagramme de séquence AUML

NB : les scénarios du cas d'utilisation i et j sont représentés avec les diagrammes de séquence AUML 1 et 2 respectivement.

4. La relation de généralisation

La relation généralisation possède le même concept que l'héritage. Donc deux cas d'utilisation reliés avec cette relation, le cas d'utilisation qui possède des informations de plus le rend plus spéciale que l'autre. Le passage de la relation de généralisation vers un diagramme de séquence AUML se fait comme suit :

[4] L'opérateur alternative, ou ***alt***, est un opérateur conditionnel possédant plusieurs opérandes (l'équivalent d'une exécution à choix multiple)

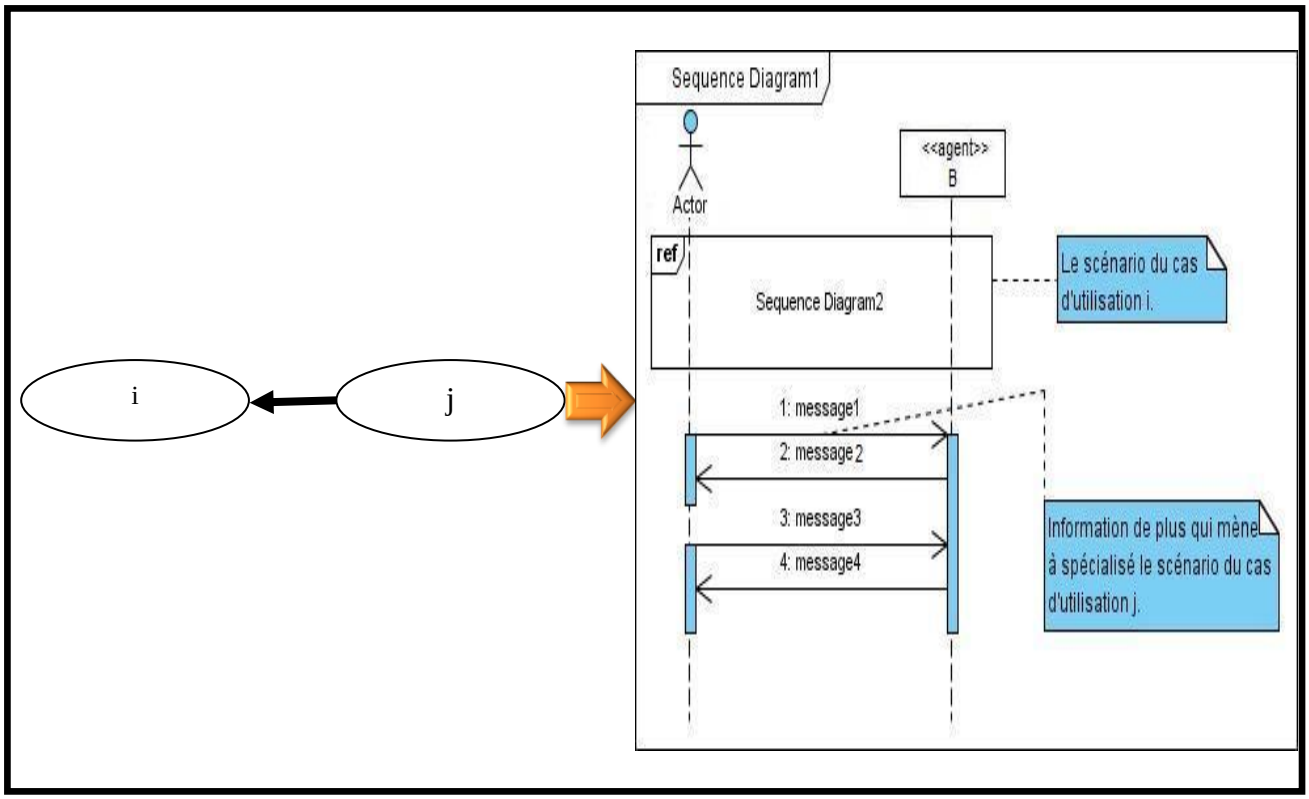

Figure 22 : La relation généralisation vers le diagramme de séquence AUML

NB : le diagramme de séquence AUML 1 et 2 représente les scénarios du cas d'utilisation *j* et *i* respectivement.

5. Processus de translation

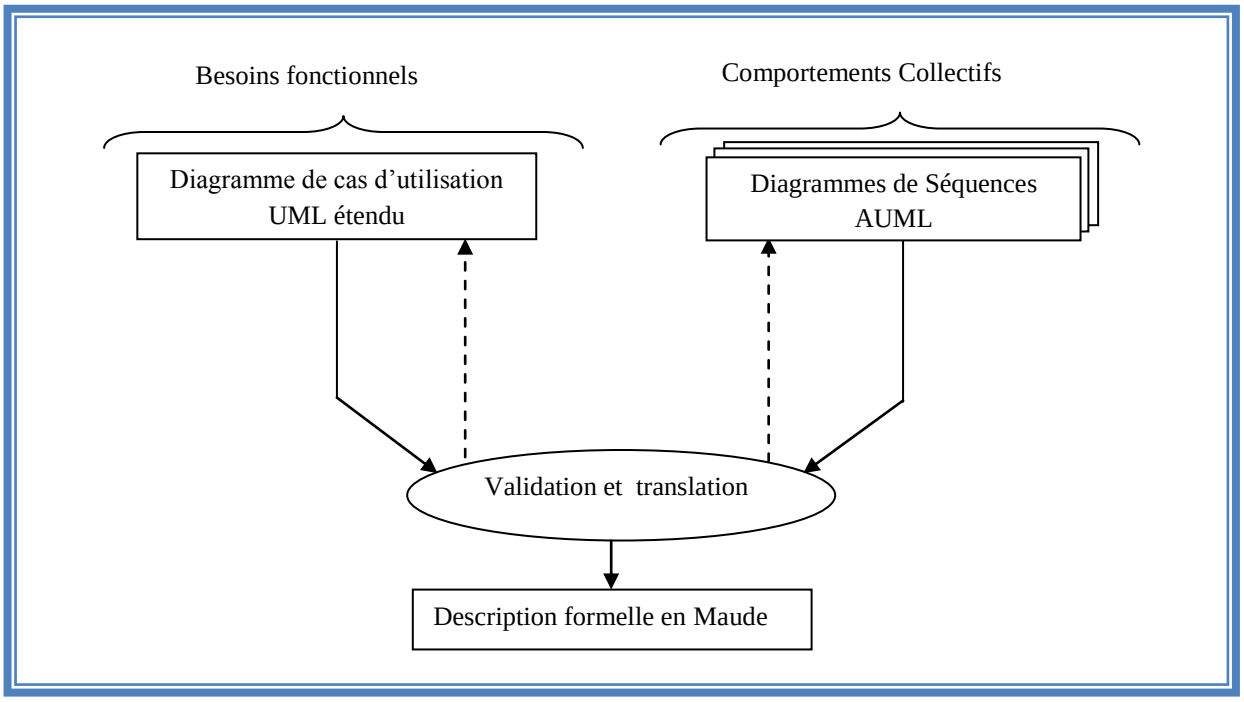

Figure 23 : Démarche générale de l'approche

La figure 23 illustre le processus de formalisation de besoins fonctionnels des SMAs. Partant d'une spécification semi-formelle de besoins fonctionnels décrite à l'aide de diagramme de cas d'utilisation UML étendu et de diagrammes de séquences AUML conjointement. Chaque cas d'utilisation dans le diagramme de cas d'utilisation est réalisé par un ou plusieurs diagrammes de séquences AUML. La démarche proposée permet, d'une part, d'analyser et de valider cette description et, d'autre part, de la translater dans une spécification Maude.

a) Description de différents modes d'interaction entre agents

Avant de détailler les différents modules de notre framework, nous décrivons dans cette section la description en Maude de différents modes d'interaction. Les trois modes d'interaction définis dans AUML (figure 15) sont tous supportés par Maude. Nous utilisons une seule règle de réécriture (voir figure 24) pour décrire la forme d'interaction illustrée par la figure (15.a).

```
rl [L] : < A1 :  C1 | PlayRole : R1, ...> < A2 : C2 | PlayRole : R2, ...>
    =>< A1 :  C1 | PlayRole : R1, ...> < A2 : C2 | PlayRole : R2, ...>
       M1(CA-1, ...)    M2(CA-2,...)  ... Mn(CA-n,....)
```

Figure 24 : Modélisation des threads concurrents d'interaction

Ceci indique clairement l'envoi spontané des messages *Mi* contenant comme paramètres les actes de communication *CA-i*. Dans le cas de *'ou inclusif'* et de *'ou exclusif'*, il n'existe pas une stratégie d'évaluation standard permettant de choisir une alternative parmi plusieurs. Ceci rend difficile de proposer une translation précise de ces modes d'interaction dans Maude. Leur translation reste un thème ouvert. Cependant, grâce à la flexibilité de Maude, nous pouvons recommander quelques solutions. Pour décrire la forme d'interaction concernant le *'ou inclusif',* nous proposons une de ces solutions pour capturer ce concept dans Maude. Il s'agit d'utiliser *m* règles de réécriture (m ≤ n) de la forme illustrée par la figure 25.

```
crl [Li] : < A1 :  C1 | PlayRole : R1, …> < A2 : C2 | PlayRole : R2, …>
      => < A1 :  C1 | PlayRole : R1, …> < A2 : C2 | PlayRole : R2, …>
         M1(CA-i1, …) M2(CA-i2, …)  …  Mk(CA-ik, …)  if Ci  .
```

Figure 25 : Modélisation du mode d'interaction '*ou inclusif* ' dans Maude en utilisant les conditions

En effet,, *i =1,..., m* et *CA-i1 CA-i2… CA-ik* sont des actes de communication parmi *CA-1 CA-2… CA-n*. Ils présentent les actes pouvant être envoyés spontanément. La condition *Ci* valide la règle *Li*. Donc, elle contrôle son exécution. Au lieu d'utiliser les conditions, une autre alternative considérée (figure 26) consiste à utiliser un message commun à toutes les règles *Lis*, de la manière suivante:

```
rl [Li] : < A1 :  C1 | PlayRole : R1, …> < A2 : C2 | PlayRole : R2, …> M(Vi)
   => < A1 :  C1 | PlayRole : R1, …> < A2 : C2 | PlayRole : R2, …>
      M2(CA-i1 . …)  M2(CA-i2. …)  …  Mk(CA-ik. …)  .
```

Figure 26 : Modélisation du mode d'interaction '*ou inclusif*' dans Maude en utilisant les messages

Dans cette solution, une seule instance du message *M* généré à l'avance permet de lancer une seule règle de réécriture parmi les règles *Lis*. Les valeurs *Vis* des paramètres de *M* permettent de sélectionner la règle *Li* parmi les autres.

Le '*ou exclusif*'' est décrit dans Maude de façon similaire à celle de *'ou inclusif'*. Dans ce cas, nous adoptons une solution basée sur les conditions et nous obtenons *n* règles. La forme de ces règles est décrite par la figure 27.

```
crl [Li] : < A1 :  C1 | PlayRole : R1, …> < A2 : C2 | PlayRole : R2, …>
 => < A1 :  C1 | PlayRole : R1, …> < A2 : C2 | PlayRole : R2, …> CA-i   if Ci  .
```

Figure 27 : Modélisation du 'ou exclusif' dans Maude en utilisant les conditions

Si *CA-i* est le message choisi qui va être envoyé, il est donc nécessaire que la condition *Ci* soit vérifiée pour la règle de réécriture appropriée *Li*, alors que toutes les autres conditions *Cj (j =1,.. ,n, j≠i)* sont fausses. La solution basée sur les messages dans le mode d'interaction *'ou inclusif'* peut être adaptée aussi pour implémenter le mode d'interaction *'ou exlusif'*. Dans ce cas, l'exécution de chaque règle *Li* consiste à la création d'un seul acte *CA-i* (voir figure 28).

```
rl [Li] : < A1 :  C1 | PlayRole : R1, …> < A2 : C2 | PlayRole : R2, …> M(Vi)
=> < A1 :  C1 | PlayRole : R1, …> < A2 : C2 | PlayRole : R2, …> Mi(CA-i , …)  .
```

Figure 28 : Modélisation du mode d'interaction *' ou exclusif'* dans Maude en utilisant les messages

b) Architecture du framework proposé

Durant le processus de translation de notre approche, nous avons développé un framework formel qui permet la formalisation des besoins fonctionnels de SMAs à l'aide du langage Maude. Comme il est illustré dans la figure 29. Notre framework est composé de plusieurs modules : cinq modules fonctionnels et trois modules orientés objet(OO).

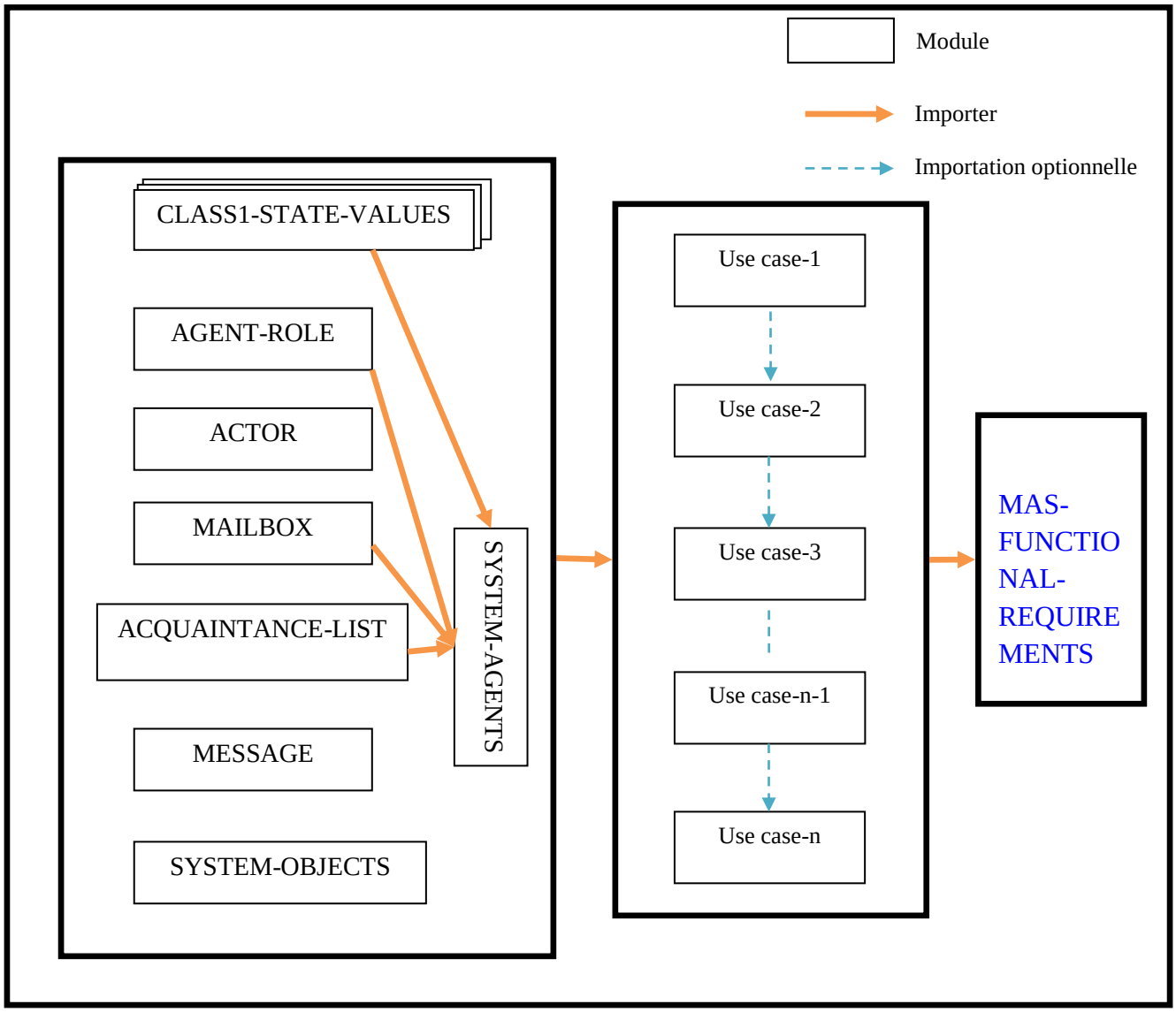

Figure 29 : L'architecture du framework formel proposé

✍ Les modules CLASSi-STATE-VALUES (figure 30) sont des modules fonctionnels qui décrivent et définissent les états de différentes classes d'agents définies dans le système et leurs types. Le nom de chaque module est la concaténation du nom de la classe d'agents et la chaîne de caractères *StateValue*.

```
(fmod AGENTS-STATE-VALUES is
sort AgentStateValue .
***partie de déclaration de différents états
…
endfm)
```

Figure 30 : Le module fonctionnel AGENT-STATE-VALUES

- Le module fonctionnel AGENT-ROLE décrit un type nommé *AgentRole* qu'on peut l'utiliser pour définir un ensemble de valeurs de rôle d'agent qui prennent ce type. Ces valeurs présentent en fait, les rôles pouvant être joués par les différents agents.

```
(fmod AGENT-ROLE is
sort AgentRole .
****partie de declaration de différents roles d'agents
…
endfm)
```

Figure 31 : Le module fonctionnel AGENT-ROLE

- ACTOR est un module fonctionnel qui définit un type nommé Actor . Dans ce module on détermine les différents acteurs qui interagissent avec le système.

```
(fmod ACTOR is
sort Actor .
***partie de declaration de différents acteurs
…
endfm)
```

Figure 32 : Le module fonctionnel ACTOR

- MAILBOX est un module fonctionnel qui permet de décrire l'état de la boîte aux lettres de l'agent : vide ou non. Ce module est formalisé comme suit :

```
(fmod MAILBOX is
 sort MailBox .
 ops Empty NotEmpty : -> MailBox . .
endfm)
```

Figure 33 : Le module fonctionnel MAILBOX

✍ La figure suivante décrit un module fonctionnel nommée ACQUAINTANCE-LIST qui permet de définir pour chaque agent une liste d'agents en interaction avec lui.

```
(fmod ACQUAINTANCE-LIST is
 sorts AcquaintanceList acquaintance .
 subsort acquaintance < AcquaintanceList .
op Head : AcquaintanceList -> acquaintance .
 op _:_ : acquaintance AcquaintanceList -> AcquaintanceList .
 op EmptyAcquaintanceList : -> AcquaintanceList .
 op IsEmptyAcquaintanceList : AcquaintanceList -> Bool .
 var Ac : acquaintance .
 var AcL : AcquaintanceList .
 eq Ac : EmptyAcquaintanceList = Ac .
 eq IsEmptyAcquaintanceList(EmptyAcquaintanceList) = true .
 eq IsEmptyAcquaintanceList(Ac : AcL) = false .
 eq Head(Ac : AcL) = Ac .
 eq Head(Ac) = Ac .
endfm)
```

Figure 34 : Le module fonctionnel ACQUAINTANCE-LIST

✍ Le module orienté objet MESSAGE (figure 35) est un module qui permet de définir tous les messages échangés entre les entités du système. Un message doit contenir les noms de l'émetteur et du receveur ainsi que son contenu (ligne [1]). Ce module décrit également la forme de l'événement interne traité par un agent (ligne [2]).

```
(omod MESSAGE is
including CONFIGURATION .
including STRING .
sort content .
msg Message : Oid Oid content -> Msg .  *** [1]
msg Event : Oid  content -> Msg .          ***[2]
endom)
```

Figure 35 : Le module Orienté Objet MESSAGE

✍ Le module SYSTEM-AGENTS (figure 36) est un module orienté objet dans lequel est définie la structure de la classe de base des agents impliqués dans le système. Cette classe a comme attributs, *PlayRole*, *State*, *MBox*, et *AcqList* pour contenir dans l'ordre, le rôle joué par l'agent, son état courant, sa boîte aux lettres et la liste de ses accointances.

```
(omod SYSTEM-AGENTS is
including CONFIGURATION .
including ACQUAINTANCE-LIST .
including AGENTS-STATES .
including AGENT-ROLE .
including MAILBOX .
sort AgentState .
class Agent |  PlayRole : AgentRole, State : AgentState, MBox  : MailBox,  AcqList : AcquaintanceList .
…
endom)
```

Figure 36 : Le module Orienté Objet SYSTEM-AGENTS

✍ Pour décrire les objets manipulés par les agents du système, nous proposons de définir le module orienté objet SYSTEM-OBJECTS (figure 37) dans lequel on déclare les différentes classes d'objets.

```
(omod SYSTEM-OBJECTS is
including CONFIGURATION .
***partie de déclaration de différentes classes d'objets
…
endom)
```

Figure 37 : Le module Orienté Objet SYSTEM-OBJECTS

✍ A chaque cas d'utilisation est associé un module orienté objet *Use-Casei* qui porte le même nom du cas d'utilisation correspondant. Dans chaque module *Use-casei* sont définies les règles de réécriture décrivant les différents scénarios d'interaction entre les agents définis dans les différents diagrammes de séquence AUML. Une fois générés, tous les modules *Use-Casei*, sont importés dans le module OO *MAS-FUNCTIONAL-SYSTEM-REQUIEREMENTS* (figure 38) représentant le module principal.

```
(omod MAS-FUNCTIONAL-REQUIREMENTS is
including Use-Case1 .
including Use-Case2 .
…
including Use-Casem .
endom)
```

Figure 38 : Le module Principale MAS-Requirements

IV.3. La génération de spécification MAUDE

1. Un seul cas d'utilisation

Un cas d'utilisation est décrit par un module orienté objet comportant les règles de réécritures en langage Maude implémentant les différents scénarios présentés par les diagrammes de séquences AUML correspondants (figure 39). Notons, que les règles peuvent être conditionnelles ou inconditionnelles.

```
(omod USE-CASE is
rl [1] : Configuration1 =>  Configuration2.
…
rl [m] : Configuration2m-1 => <Configuration2m .
endom)
```

Figure 39 : Un module orienté objet décrivant un cas d'utilisation

2. La relation d'inclusion stéréotypée « include »

Comme nous l'avons mentionné, l'aspect sémantique de la relation d'inclusion, qui est assimilée à l'inclusion d'un scénario dans un autre, se principe sera appliqué au niveau de la spécification MAUDE. Donc on aura une spécification formelle qui importe une autre spécification. La figure 40 montre la spécification de la relation d'inclusion entre les cas d'utilisation A-INCLUDING et B-INCLUDED. Le comportement du premier inclut celui du deuxième.

(omod INCLUDING-USE-CASE is including INCLUDED-USE-CASE . *** importation du module du cas d'utilisation Included use case. rl [1] : $Configuration_1$ => $Configuration_2$. … rl [n] : $Configuration_{n-1}$ => $Configuration_n$. endom)	(omod INCLUDED-USE-CASE is rl[1] : $Configuration_1$ => $Configuration_2$. … rl [m] : $Configuration_{2m-1}$ => $Configuration_{2m}$. endom)

Figure 40 : spécification Maude de deux cas d'utilisation reliés par la relation d'inclusion

3. La relation d'extension stéréotypée « extend »

L'exécution du scénario du cas d'utilisation étendant (***extending)*** est liée à la satisfaction de la condition *point d'extension* (cas d'utilisation étendant peut être appelé au cours de l'exécution du cas étendu (extended)). Dans ce cas on peut utiliser une règle de réécriture conditionnelle qui représente le point d'extension au niveau du diagramme du cas d'utilisation. La figure 41 présente la spécification Maude de la relation d'extension.

(omod EXTENDED-USE-CASE is
including EXTENDING-USE-CASE .
…
crl[k] : $Configuration_k$ => $Configuration_{k+1}$
if (condition in extension point is true) .
*** execution du scénario du EXTENDING-USE-CASE
crl[l] : $Configuration_L$ => $Configuration_{L+1}$
if (condition in extension point is false) .
** Échec d'exécution du scénario du cas d'utilisation EXTENDING-USE-CASE .
…
endom)
	(omod EXTENDING-USE-CASE is
	rl[1] : $Configuration_1$ => $Configuration_2$.
	…
	rl [m] : $Configuration_{2m-1}$ => $Configuration_{2m}$.
	endom)

Figure 41 : spécification Maude de deux cas d'utilisation reliés par la relation d'extension

4. La relation généralisation

Ce type de relation est également translaté avec MAUDE par l'importation de modules. Le module qui décrit le cas d'utilisation particulier (spécial) doit importer celui qui décrit le cas d'utilisation général tout en se basant sur la création des classes dérivées en utilisant le mot clé *subclass* dans le module spécial à partir des classes de base définies dans le modules général. Les instances des classes dérivées (qui sont définies dans le module spécial) peuvent réutiliser les règles de réécritures définies dans le module importé (le module général). Parfois, on ne crée pas des sous classes mais on propose des nouvelles opérations. Autrement dit, qu'on peut se limiter à définir des nouvelles règles de réécritures dans le module spécial pour spécialiser le comportement du cas d'utilisation général.

(omod GENERAL-USE-CASE is rl[1] : $Configuration_1$ => $Configuration_2$. … rl[n] : $Configuration_{2n-1}$ => $Configuration_n$. endom)	(omod SPECIAL-USE-CASE is including GENERAL-USE-CASE . *** appel du module GENERAL-USE-CASE . rl[1] : $Configuration_1$ => $Configuration_2$. … rl[m] : $Configuration_{2m-1}$ => $Configuration_m$. endom)

Figure 42 : spécification Maude de deux cas d'utilisation reliés par la relation généralisation

V. Étude de cas

Le système de la banque représente un système très claire, facile et compréhensible par tout le monde. La raison pour laquelle nous avons choisi cet exemple pour valider notre approche.

V.1 Description de système

Le système de banque est constitué de plusieurs entités qui ont comme objectif d'offrir plusieurs services aux différents clients. Parmi ces services : *consulter compte, effectuer des virements, retirer de l'argent, etc*. Nous voulons derrière cette étude de cas générer une spécification formelle écrite en langage Maude qui décrit les besoins fonctionnels de ce système.

V.2 L'application du processus de translation

1. Modélisation avec le diagramme de cas d'utilisation d'UML étendu

Dans le système de la banque on peut recenser trois rôles d'acteur interne (agent). Un agent qui joue le rôle d'un ***receveur*** (recipient) de la banque, un agent qui joue le rôle d'un ***distributeur de billets*** (ATM -Automatic Teller Machine), et un autre agent qui joue le rôle de navigateur (browser).

L'acteur externe qui va interagir avec le système de la banque est le ***client*** qui va bénéficier des services du système. Ces derniers représentent un ensemble de cas d'utilisation. On peut recenser dans ce système les cas d'utilisation suivants : *retirer de l'argent (Withdraw-money), authentification(Authenticate), effectuer un virement(Transfer-money), vérifier solde(Check amount in account), consulter compte(Consult account), consulter compte depuis internet(Consult account from internet).* Ainsi le cas d'utilisation qui représente l'interaction entre l'agent receveur et l'agent ATM c'est *recharger le distributeur (Reload money tickets)* par des billets.

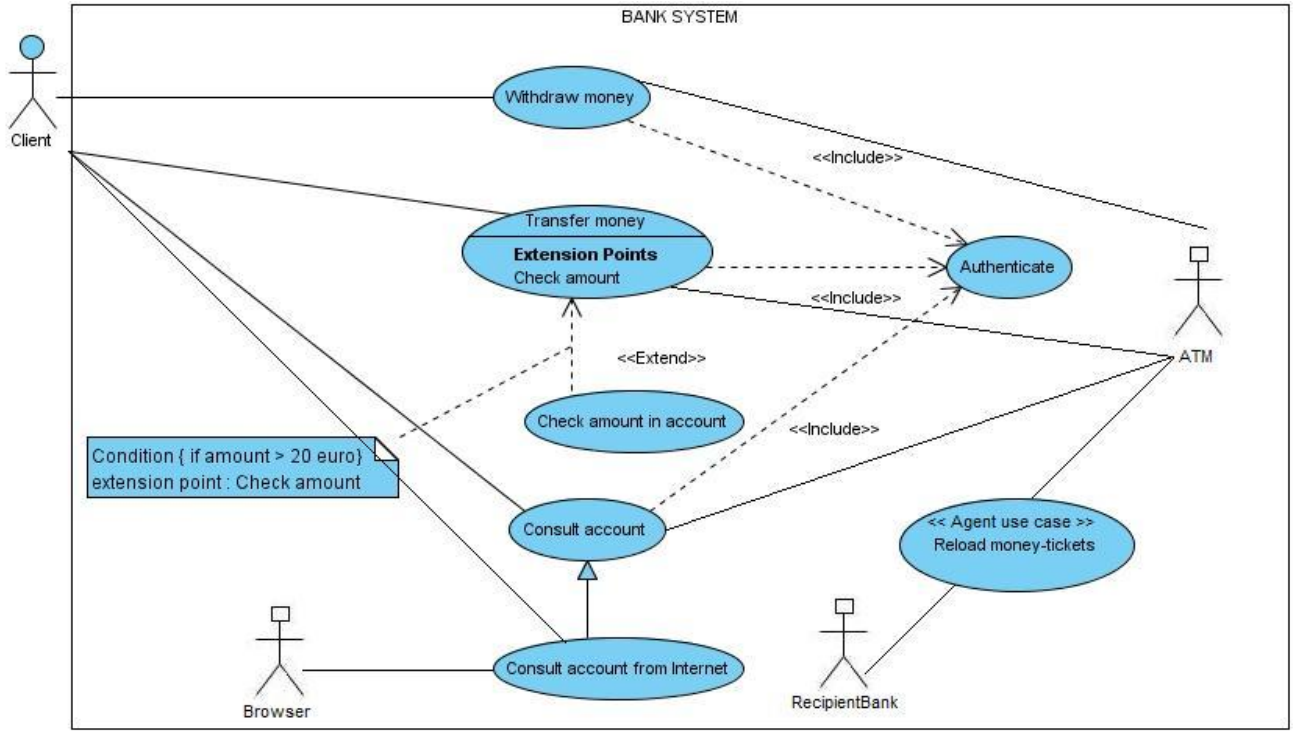

Figure 43 : Diagramme de cas d'utilisation étendu qui représente le système Banque

2. Diagrammes de séquence AUML

Après la construction du diagramme du cas d'utilisation, l'étape suivante est consacrée pour la représentation des scenarios à l'aide du diagramme de séquence AUML.

✍ ***Authentifier (authentication)***

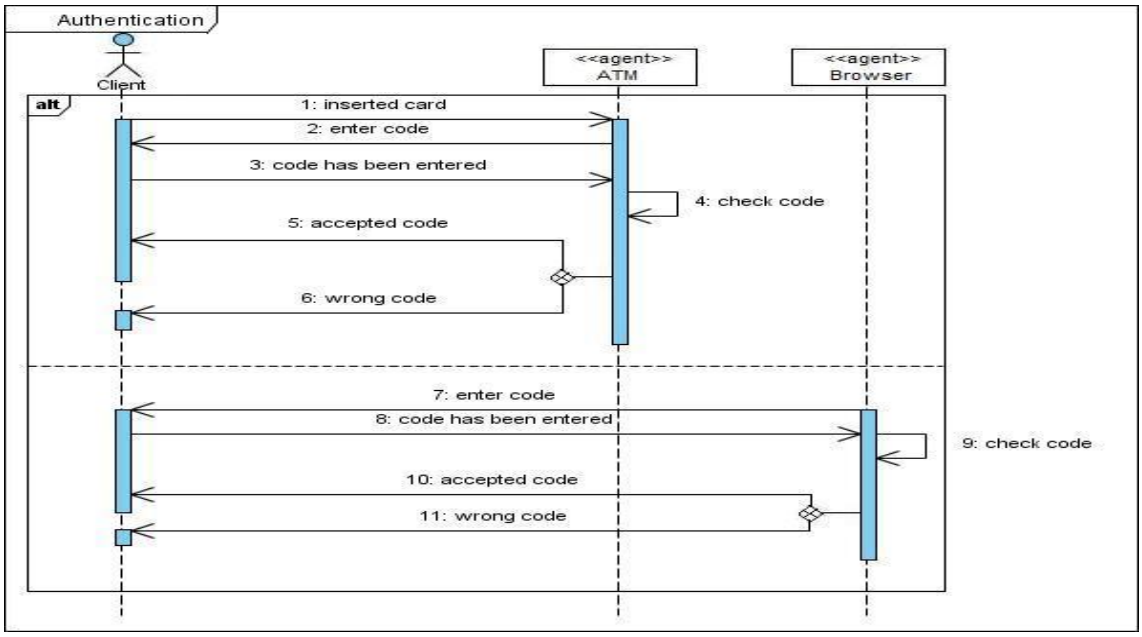

Figure 44 : Diagramme de séquence AUML relatif au cas d'utilisation *authentification*

✍ ***retirer de l'argent (withdraw money)***

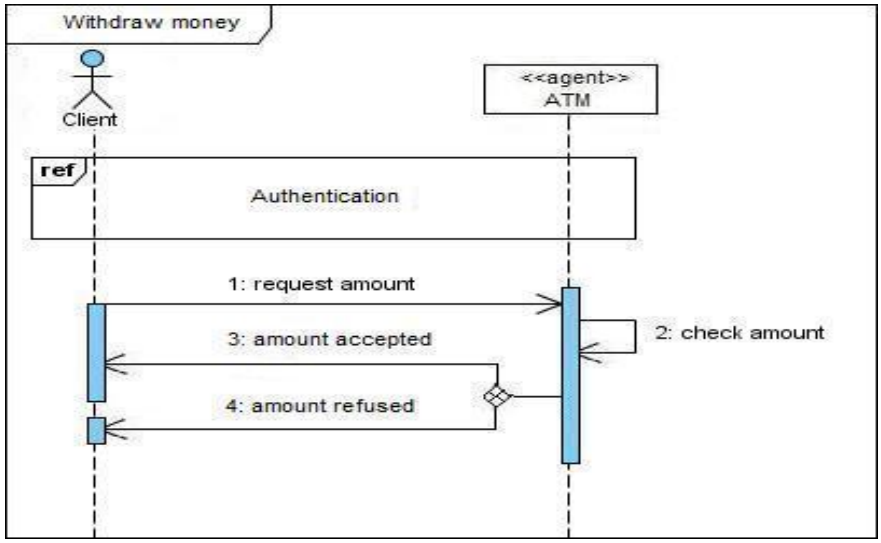

Figure 45 : Diagramme de séquence AUML relatif au cas d'utilisation *withdraw money*

✍ ***verifier solde (check amount in account)***

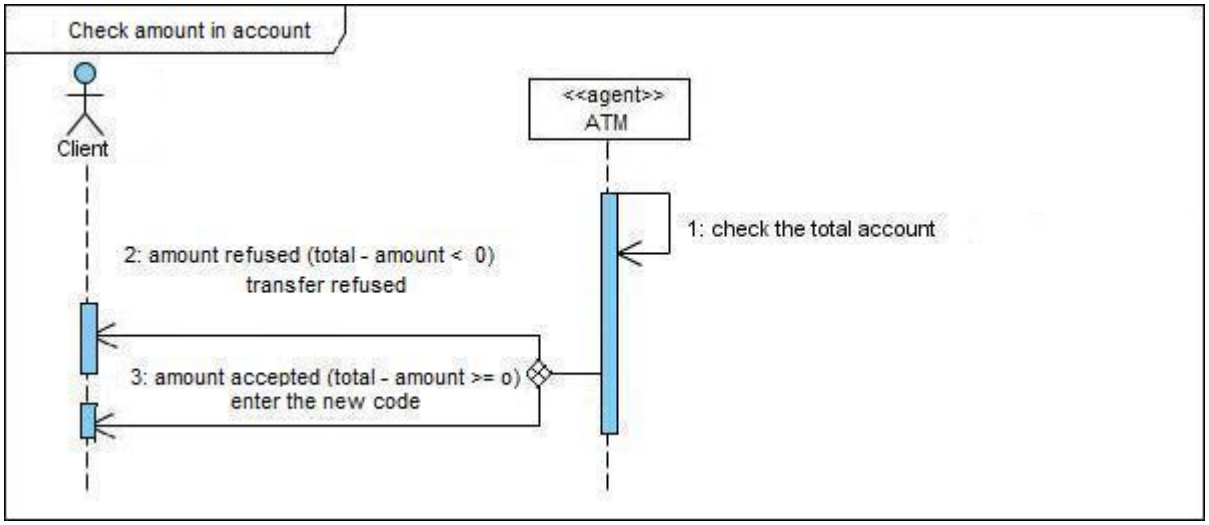

Figure 46 : Diagramme de séquence AUML relatif au cas d'utilisation *check amount in account*

✍ ***effectuer un virement (transfer money)***

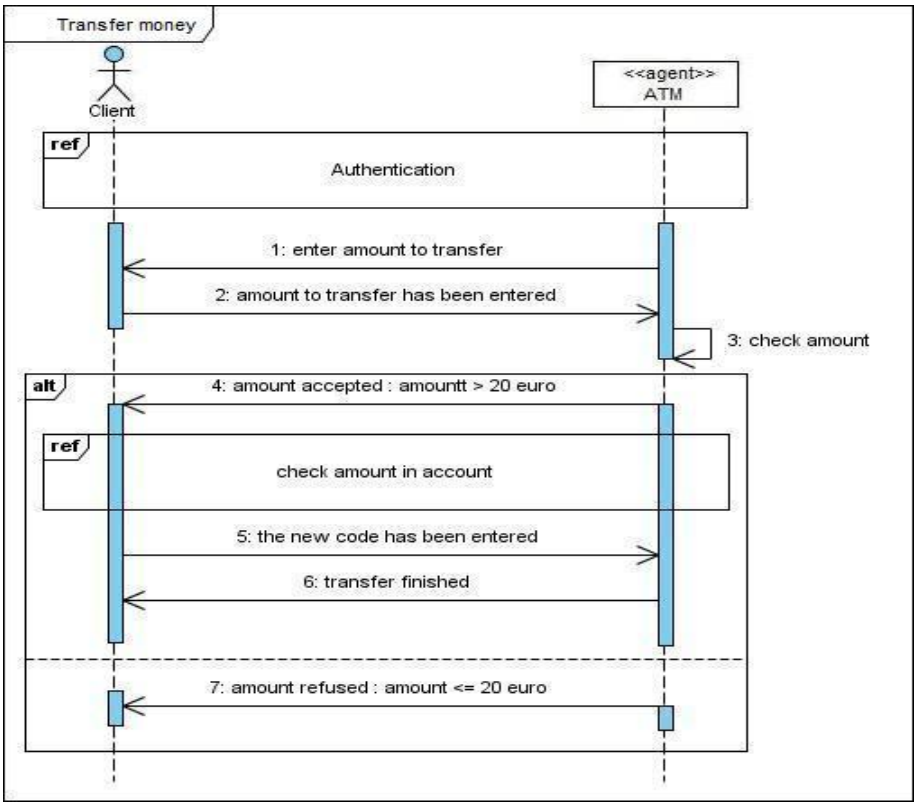

Figure 47 : Diagramme de séquence AUML relatif au cas d'utilisation *transfer money*

✍ ***consulter compte (consult account)***

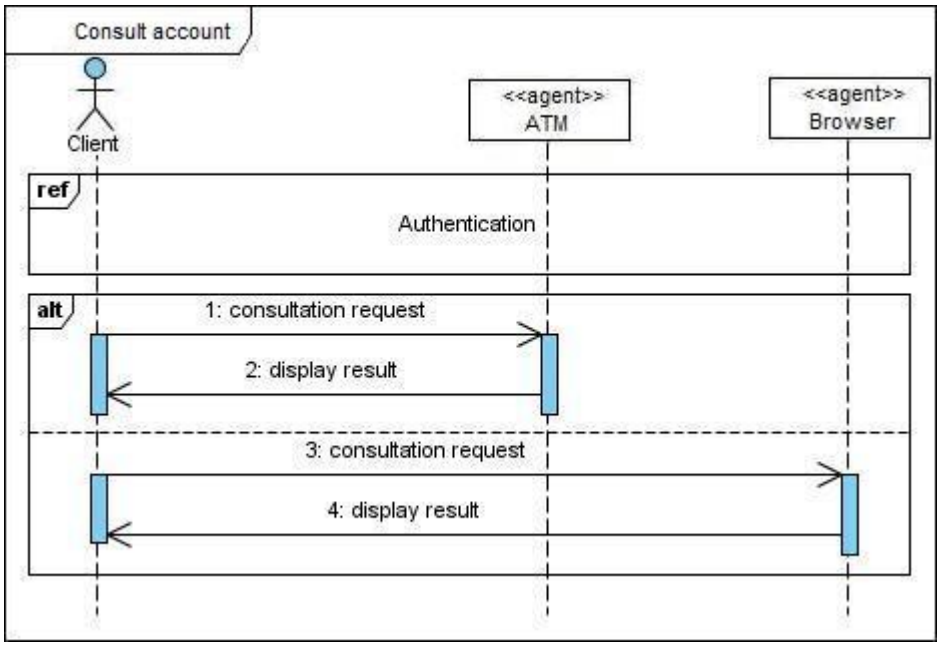

Figure 48 : Diagramme de séquence AUML relatif au cas d'utilisation *consult account*

✍ ***Consulter depuis internet (Consult Account from Internet)***

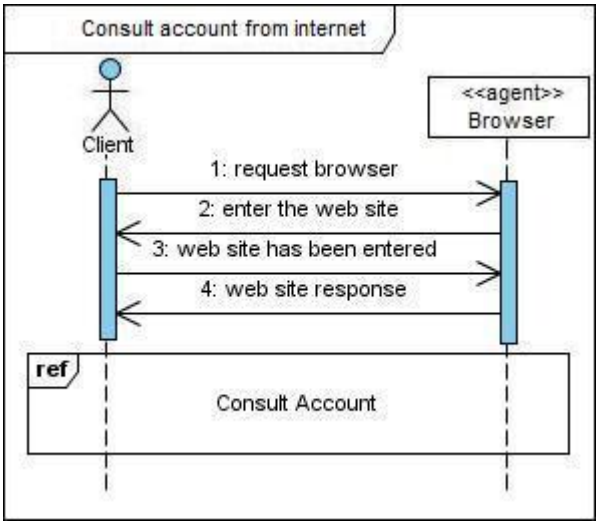

Figure 49 : Diagramme de séquence AUML relatif au cas d'utilisation *Consult Account from Internet*

✍ ***Recharger les billets(reload money-tickets)***

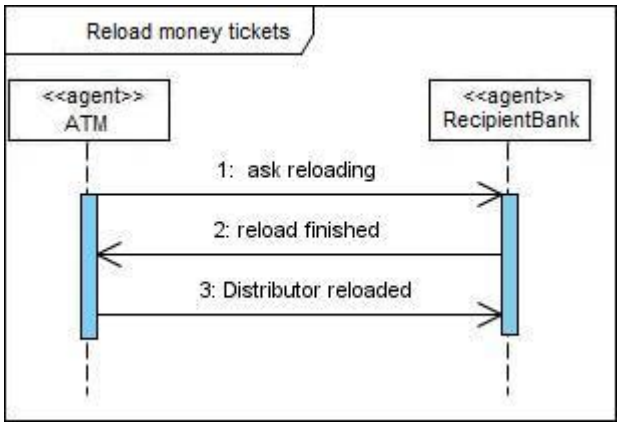

Figure 50 : Diagramme de séquence AUML relatif au cas d'utilisation *Reload money-tickets*

3. Génération de specification Maude

Après l'application du processus de translation proposé, nous obtenons :

✍ Les modules ATM-STATE-VALUES (figure 51), RECIPIENT-BANK-STATE-VALUES (figure 52), et BROWSER-STATE-VALUES (figure 53) qui décrivent respectivement les états de différents agents du système : ATM, RECIPIENT-BANK, et BROWSER.

```
fmod ATM-STATE-VALUES is
sort AtmStateValue .
ops StartA WorkA BreakDownA WaitA  EndOperationA : -> AtmStateValue .
endfm
```

Figure 51 : Le module fonctionnel ATM-STATE-VALUES

```
fmod RECIPIENT-BANK-STATE-VALUES is
sort RecipientStateValue .
ops StartR WorkR WaitR EndOperationR : -> RecipientStateValue .
endfm
```

Figure 52 : Le module fonctionnel RECIPIENT-BANK-STATE-VALUES

```
fmod BROWSER-STATE-VALUES is
 sort BrowserStateValue .
 ops StartB WaitB WorkB EndOperationB : -> BrowserStateValue .
endfm
```

Figure 53 : Le module fonctionnel BROWSER-STATE-VALUES

- **Module AGENT-ROLE :** ce module (figure 54) nous permet de définir les rôles joués par les agents de système. Ici trois rôles sont définis, à savoir, ATM, RECIPIENTBANK, et BROWSER.

```
fmod AGENT-ROLE is
sort AgentRole .
ops ATM BROWSER RECIPIENTBANK : -> AgentRole .
endfm
```

Figure 54 : Le module fonctionnel AGENT-ROLE

- **Module ACTOR :** Dans ce module sont définis les acteurs externes au système. Dans notre exemple, un seul acteur est défini, à savoir, l'acteur Client.

```
fmod ACTOR is
sort Actor .
op Client : -> Actor .
endfm
```

Figure 55 : Le module fonctionnel ACTOR

- Les module MAILBOX et ACQUAINTANCE-LIST tel qu'ils sont déinis précédemment dans le framework.

- **Module MESSAGE :** Ce module (figure 56) décrit tous les messages échangés dans le système banque. La ligne [1] présente les contenus de messages échangés entre les différentes entités du système.

```
omod MESSAGE is
including CONFIGURATION .
including ACTOR .
including STRING .
sort content .
subsort Actor < Oid .
subsort String < Oid .
ops insercard entercode enterthenewcode thenewcodehasbeenentered consultationtyperequest
consutatiotypeordinary codehasbeenentered acceptedcode wrongcode transferrefused
websitehasbeenentered websiteresponse consultationrequest displayresult requestamount
amountrefused passtotransfer continue amountaccepted withdrawamount askreloading reloadfinished
Distributorreloaded enteramounttotransfer amounthasbeenentered transferfinished checkcode
checkamount checkthetotalaccount requestbrowser enterthewebsite websitehasbeenentered
websiteresponse : -> content .    ***[1]
msg Message : Oid Oid content -> Msg .
msg Event : Oid  content -> Msg .
endom
```

Figure 56 : Le module orienté objet MESSAGE

- **Module BANK-AGENTS:** ce module (figure 57) importe les modules ACQUAINTANCE-LIST, ATM-STATE-VALUES, RECIPIENT-BANK-STATE-VALUES, BROWSER-STATE-VALUES, AGENT-ROLE, MAILBOX. En plus de la définition de la classe de base des agents, il contient également, la définition des identificateurs de différents agents Atm, Recipient et Browser (ligne[1]). La fonction *TargetState* de la ligne [2] permet de déterminer l'état destinataire d'un état d'agent donné. La ligne [3] décrit la classe WebAgent dérivée de la classe Agent. Elle se spécialise par l'attribut WebSite qui définit l'adresse du site web de la banque.

```
omod BANK-AGENTS is
including CONFIGURATION .
including ACQUAINTANCE-LIST .
including ATM-STATE-VALUES .
including RECIPIENT-BANK-STATE-VALUES .
including BROWSER-STATE-VALUES .
including AGENT-ROLE .
including MAILBOX .
```

```
sort AgentState .
subsort AtmStateValue  < AgentState .
subsort RecipientStateValue < AgentState .
subsort BrowserStateValue < AgentState .
including AGENT-ROLE .
including MAILBOX .
ops Atm Recipient Browser : -> Oid . ***[1]
op TargetState : AgentState -> AgentState .   *** [2]
…
class Agent | PlayRole : AgentRole, State : AgentState, MBox : MailBox, AcqList : AcquaintanceList .
class WebAgent | WebSite : WebAdress .                    ***[3]
subclass WebAgent < Agent .


endom
```

Figure 57 : Le module orienté objet BANK-AGENTS

- **Module BANK-OBJECTS :** dans ce module, on définit la classe d'objets *Account* **(ligne[2])** qui décrit un compte bancaire et qui a comme attributs : *bal* et *amount* qui représentent respectivement, le solde actuel et le montant à retirer. Dans cette étude de cas on a besoin de deux compte bancaires, pour cela, nous définissons dans ce module *Acc* et *Acc1* (ligne[2]), deux identificateurs de deux objet de la classe Account.

```
omod BANK-OBJECTS is
including CONFIGURATION .
ops Acc Acc1 : -> Oid . *** [1]
class Account | bal : Int, amount : Int .  ***[2]
endom
```

Figure 58 : Le module orienté objet BANK-OBJECTS

Après avoir présenté les modules nécessaires au développement des différents cas d'utilisation nous passons maintenant à la description formelle de ces derniers. Pour chaque cas d'utilisation est associé un module orienté objet implémentant les différents scénarios possibles qui sont décrits à l'aide de diagramme de séquence AUML correspondant. Chaque envoi de

messages entre deux entités au sein de diagramme de séquence est décrit à l'aide d'une règle de réécriture. Dans notre étude de cas comme il est montré par la figure 43, nous avons sept cas d'utilisation, d'où il nécessaire de générer sept modules orientés objet.

NB : dans ce qui suit sont présentées les parties essentielles de chaque module du système Banque (BANK SYSTEM).

- **Module AUTHENTICATION :** ce module (figure 59) implémente deux alternatives relativement différentes. En important ce module, on peut lancer le processus d'authentification à travers l'agent *ATM* ou l'agent *Browser*.

```
(omod AUTHENTICATION is

…

**********partie relative à l'autentification à travers l'ATM**********

rl[introcard] :

    Message(Client, Atm, insercard) < Atm : Agent | PlayRole : ATM, State : StartA, MBox : Empty,  AcqList
      :  Recipient >    =>

< Atm : Agent | PlayRole : ATM, State : WaitA, MBox : Empty, AcqList : Recipient >

 Message(Atm, Client, entercode) .                    ***[1]

rl[EnterCode] :

    Message(Atm, Client, entercode) => Message(Client, Atm, codehasbeenentered) .

rl[codehasbeenentred] :  Message(Atm, Client, entercode)

        < Atm : Agent | PlayRole : ATM, State : WaitA, MBox : Empty,  AcqList : Recipient >

           =>

       < Atm : Agent | PlayRole : ATM, State : WorkA, MBox : NotEmpty,  AcqList : Recipient >

                    Event(Atm, checkcode) .

**********Description du mode d'interaction "ou exclusif" en utilisant les Messages**********

rl[acceptedcode] :

                    Event(Atm, checkcode)

       < Atm : Agent | PlayRole : ATM, State : WorkA, MBox : NotEmpty, AcqList : Recipient >

          =>

      < Atm : Agent | PlayRole : ATM, State : WaitA, MBox : NotEmpty, AcqList : Recipient >
```

Message(Atm, Client, acceptedcode) .

rl[wrongcode] :

Event(Atm, checkcode)

< Atm : Agent | PlayRole : ATM, State : WorkA, MBox : NotEmpty, AcqList : Recipient >

=>

< Atm : Agent | PlayRole : ATM, State : EndOperationA, MBox : NotEmpty, AcqList : Recipient >

Message(Atm, Client, wrongcode) .

******************partie relative à l'autentification à travers le Browser (Internet)************

rl[websiteresposne] : ***Message(Browser, Client, websiteresponse)***

< Browser : Agent | PlayRole : BROWSER, State : WorkB, MBox : NotEmpty, AcqList : EmptyAcquaintanceList >

=>

< Browser : Agent | PlayRole : BROWSER, State : WaitB, MBox : NotEmpty, AcqList : EmptyAcquaintanceList > Message(Browser, Client, entercode) . ****** [2]***

rl[EnterCode] :

Message(Browser, Client, entercode) => Message(Client, Browser, codehasbeenentered) .

rl[codehasbeenentred] : Message(Browser, Client, entercode)

< Browser : Agent | PlayRole : BROWSER, State : WaitB, MBox : NotEmpty, AcqList : EmptyAcquaintanceList >

=>

< Browser : Agent | PlayRole : BROWSER, State : WorkB, MBox : NotEmpty, AcqList : EmptyAcquaintanceList >

Event(Browser, checkcode) .

**** le passage du mode d'interaction "ou exclusif" vers MAUDE en utilisant les Messages*

rl[acceptedcode] :

Event(Browser, checkcode)

< Browser : Agent | PlayRole : BROWSER, State : WorkB, MBox : NotEmpty, AcqList : EmptyAcquaintanceList >

=>

< Browser : Agent | PlayRole : BROWSER, State : WaitB, MBox : NotEmpty, AcqList :

```
EmptyAcquaintanceList >

                    Message(Browser, Client, acceptedcode) .

rl[wrongcode] :

                    Event(Browser, checkcode)

 < Browser : Agent | PlayRole : BROWSER, State : WorkB, MBox : NotEmpty,   AcqList :
EmptyAcquaintanceList >

      =>

 < Browser : Agent | PlayRole : BROWSER, State : EndOperationB, MBox : NotEmpty,   AcqList :
EmptyAcquaintanceList >

                    Message(Browser, Client, wrongcode) .

endom)
```

Figure 59 : Le module orienté objet AUTHENTICATION

Concernant l'authentification à travers l'ATM, nous prenons par exemple, la règle de réécriture de la ligne [1] initialisant ce processus. Cette dernière décrit la première interaction entre le client et l'ATM. Après avoir reçu un message indiquant l'insertion de la carte du client, l'agent ATM part de don état initial (*StartA*) en demandant de ce dernier de faire entrer son code, en lui envoyant un message, tout en passant en état d'attente.

A la différence de la première alternative, l'authentification à travers Internet nécessite la génération auparavant d'un message par le Browser indiquant qu'il est prêt à accomplir la tâche (voir ligne [2]). Ce dernier message est le résultat de l'interaction entre le client est le Browser dans le module *CONSULT-ACCOUNT-FROM-INTERNET* de la figure 64. Le Browser génère également un message demandant au client de faire entrer son code, tout en passant en état d'attente.

✍ Module WITHDRAW-MONEY

```
(omod WITHDRAW-MONEY is

including AUTHENTICATION .

...

rl [IncludingLink] : Message(Atm, Client, acceptedcode) => Message(Client, Atm, requestamount) .
rl[requestamount] :        Message(Client, Atm, requestamount)
          < Atm : Agent | PlayRole : ATM, State : WaitA, MBox : NotEmpty, AcqList : Recipient >
```

```
        =>
        < Atm : Agent | PlayRole : ATM, State : WorkA, MBox : NotEmpty,  AcqList : Recipient >
                        Event(Atm, checkamount) .
*** decsription du mode 'interaction "ou exclusif" dans  MAUDE en utilisant les conditions.
crl[amountaccepted] :                                                    ***[1]
                        Event(Atm, checkamount)
    < Atm : Agent | PlayRole : ATM, State : WorkA, MBox : NotEmpty, AcqList : Recipient >
    < Acc : Account | bal : T, amount : A >
       =>
    < Atm : Agent | PlayRole : ATM, State : EndOperationA, MBox : NotEmpty, AcqList : Recipient >
    < Acc : Account | bal : ( T - A), amount : A >
                    Message(Atm, Client, amountaccepted)
                                if T >= A .
crl[amountrefused] :                                                     ***[2]
                            Event(Atm, checkamount)
     < Atm : Agent | PlayRole : ATM, State : WorkA, MBox : NotEmpty, AcqList : Recipient >
     < Acc : Account | bal : T, amount : A >
        =>
    < Atm : Agent | PlayRole : ATM, State : EndOperationA, MBox : NotEmpty,  AcqList : Recipient >
   < Acc : Account | bal : T, amount : A >
                            Message(Atm, Client, amountrefused)
                                    if T < A .
endom)
```

Figure 60 : Le module orienté objet WITHDRAW-MONEY

Avant de retirer de l'argent, le client doit subir au processus d'authentification. Le module WITHDRAW-MONEY (figure 60) importe le module AUTHENTIFICATION pour vérifier le code du client. Si le processus de vérification termine avec succès en acceptant le code entré par le client, l'ATM passe à la vérification du montant. Si le montant est accepté, l'ATM accomplit l'opération de retrait, en exécutant la règle de réécriture de la ligne [1], sinon il exécute la règle de réécriture de la ligne [2], indiquant que le montant est refusé.

✍ Module CHECK-AMOUNT-IN-ACCOUNT

```
(omod CHECK-AMOUNT-IN-ACCOUNT is
...
rl[AmountAccepted] :
                        Message(Atm, Client, amountaccepted)
    < Atm : Agent | PlayRole : ATM, State : WaitA, MBox : NotEmpty,  AcqList : Recipient >
                        =>
    < Atm : Agent | PlayRole : ATM, State : WorkA, MBox : NotEmpty,  AcqList : Recipient >
                        Event(Atm, checkthetotalaccount)  .
*** le passge du mode d'interaction "ou exclusif" vers MAUDE en utilisant les conditions.
crl[refusedtransfer] :
                        Event(Atm, checkthetotalaccount)
    < Atm : Agent | PlayRole : ATM, State : WorkA, MBox : NotEmpty, AcqList : Recipient >
    < Acc : Account | bal : T, amount : A >
                =>
    < Atm : Agent | PlayRole : ATM, State : EndOperationA, MBox : NotEmpty,  AcqList : Recipient >
    < Acc : Account | bal : T, amount : A >
                        Message(Atm, Client, transferrefused)
                                if (T < A) .
crl[entercode] :
                    Event(Atm, checkthetotalaccount)
    < Atm : Agent | PlayRole : ATM, State : WorkA, MBox : NotEmpty,  AcqList : Recipient >
    < Acc : Account | bal : T, amount : A >
 =>
    < Atm : Agent | PlayRole : ATM, State : WaitA, MBox : NotEmpty, AcqList : Recipient >
    < Acc : Account | bal : T, amount : A >
                    Message(Atm, Client, enterthenewcode)
                                if ( T >= A ) .
endom)
```

Figure 61 : Le module orienté objet CHECK-AMOUNT-IN-ACCOUNT

Le module TRANSFER-MONEY (figure 62) importe les deux modules AUTHENTICATION et CHECK-AMOUNT-IN-ACCOUNT. Ce module implémente un transfert d'argent entre deux comptes bancaires (*Acc* et *Acc1*) (ligne [4]). Bien entendu, l'exécution de la règle de la ligne [4] requiert l'exécution de deux processus d'authentification et de vérification du montant dans le compte.

La règle de la ligne [1] décrit une sorte de liaison entre le résultat d'exécution avec succès du module AUTHENTIFICATION et le module TRANSFER-MONEY. En effet, l'activation du module *CHECK-AMOUNT-IN-ACCOUNT* dépend de l'exécution de la règle de la ligne [2] qui décrit le point d'extension. Le résultat d'exécution avec succès de ce dernier module constitue la partie gauche de la règle de la ligne [3] dont l'exécution lance le processus de transfert décrit par la règle de la ligne [4]. La règle de la ligne [5] s'exécute dans le cas où la condition du point d'extension n'est pas vérifiée.

✍ Module TRANSFER-MONEY

```
(omod TRANSFER-MONEY is
including AUTHENTICATION .
including CHECK-AMOUNT-IN-ACCOUNT .
...
rl[IncludingLink] :                                                    ***[1]
 Message(Atm, Client, acceptedcode) => Message(Atm, Client, enteramounttotransfer) .
*************************************************************************
rl[amounttotransfer] :
      Message(Atm, Client, enteramounttotransfer) => Message(Client, Atm, amounthasbeenentered) .
rl[amounthasbeenentered] :  Message(Client, Atm, amounthasbeenentered)
            < Atm : Agent | PlayRole : ATM, State : WaitA, MBox : NotEmpty,  AcqList : Recipient >
               =>
            < Atm : Agent | PlayRole : ATM, State : WorkA, MBox : NotEmpty,  AcqList : Recipient >
                                Event(Atm, checkamount) .
crl[amountaccepted] :                                                  ***[2]
          Event(Atm, checkamount)        < Acc : Account | bal : T, amount : A >
          < Atm : Agent | PlayRole : ATM, State : WorkA, MBox : NotEmpty,  AcqList : Recipient >
               =>
         < Atm : Agent | PlayRole : ATM, State : WaitA, MBox : NotEmpty, AcqList : Recipient >
         < Acc : Account | bal : T, amount : A >    Message(Atm, Client, amountaccepted)
          if (A > 20) .
*** ************Activation du module CHECK-AMOUNT-IN-ACCOUNT ***************.
rl[ReferenceLink] :                                                    ***[3]
            Message(Atm, Client, enterthenewcode) => Message(Client, Atm, thenewcodehasbeenentered) .

rl[transferfinished] :                                                 ***[4]
```

```
                Message(Client, Atm, thenewcodehasbeenentered)
                < Atm : Agent | PlayRole : ATM, State : WaitA, MBox : NotEmpty,  AcqList : Recipient >
                < Acc : Account | bal : T, amount : A >   < Acc1 : Account | bal : T1, amount : A1 >
                  =>
               < Atm : Agent | PlayRole : ATM, State : EndOperationA, MBox : NotEmpty, AcqList : Recipient
>
                < Acc : Account | bal : (T - A), amount : A >   < Acc1 : Account | bal : (T1 + A), amount : A1 >
                Message(Atm, Client, transferfinished) .


crl[AmountRefused] :                                                    ***[5]
                  Event(Atm, checkamount) < Acc : Account | bal : T, amount : A >
               < Atm : Agent | PlayRole : ATM, State : WorkA, MBox : NotEmpty,  AcqList : Recipient >
          =>
             < Atm : Agent | PlayRole : ATM, State : EndOperationA, MBox : NotEmpty,  AcqList : Recipient
>
          < Acc : Account | bal : T, amount : A >   Message(Atm, Client, amountrefused)
          if (A <= 20) .
endom)
```

Figure 62 : Le module orienté objet TRANSFER-MONEY

✍ **Module CONSULT-ACCOUNT :** Avant de répondre à la requête formulée par le client (ligne [2]), ce dernier doit subir une authentification en important le module correspondant (AUTHENTIFICATION) dans le module de la figure 63. L'exécution de ce module se termine en cas de succès par la génération d'un message d'acceptation de code entré par le client, ce qui permet de déclencher la règle de réécriture de la ligne [1]. Partant de son état source S, l'agent (ATM ou BROWSER, tout dépend de mode de consultation, via ATM ou via Internet) répond à la requête de consultation en affichant les résultats (ligne [2]), tout en changeant son état à l'aide de la fonction *TargetSate* définie dans le module BANK-AGENTS (figure 57).

```
(omod CONSULT-ACCOUNT is
including AUTHENTICATION .
…
rl [RequestConsultation] :                                              ***[1]
 Message(Ag, Client, acceptedcode) => Message(Client, Ag, consultationrequest) .
```

```
rl[DisplayResult] :                                          ***[2]
        Message(Client, Ag, consultationrequest)
        < Ag : Agent | PlayRole : R, State : S, MBox : NotEmpty, AcqList : Acl >
      =>
        < Ag : Agent | PlayRole : R, State : TargetState(S), MBox : NotEmpty, AcqList : Acl >
        Message(Ag, Client, displayresult) .
endom)
```

Figure 63 : Le module orienté objet CONSULT-ACCOUNT

✍ **Module CONSULT-ACCOUNT-FROM-INTERNET :** À la différence de la consultation via l'ATM, la consultation via Internet (Browser) nécessite d'entrer dans le site web correspondant et de confirmer de son fonctionnement avant de lancer le processus de consultation décrit dans le module importé CONSULT-ACCOUNT. Ces étapes préliminaires sont décrites par les règles de réécritures figurant dans le module CONSULT-ACCOUNT-FROM-INTERNET de la figure 64. Le module CONSULT-ACCOUNT-FROM-INTERNET spécialise le module CONSULT-ACCOUNT. L'agent Browser est une instance de la classe WebAgent qui est une classe dérivée de la classe Agent. Par conséquent le comportement décrit par les règles de réécritures définies dans le module CONSULT-ACCOUNT est hérité par la classe WebAgent.

```
(omod CONSULT-ACCOUNT-FROM-INTERNET is
including CONSULT-ACCOUNT .
…
rl[requestbrowser] :
            Message(Client, Browser, requestbrowser)
            < Browser : WebAgent | PlayRole : BROWSER, State : StartB, MBox : Empty,  AcqList : EmptyAcquaintanceList, WebSite : BA >
          =>
            < Browser : WebAgent | PlayRole : BROWSER, State : WaitB, MBox : Empty,  AcqList : EmptyAcquaintanceList, WebSite : BA  >
            Message(Browser, Client, enterthewebsite) .

rl[enterwebsite] :  Message(Browser, Client, enterthewebsite)
        < Browser : WebAgent | PlayRole : BROWSER, State : WaitB, MBox : Empty,  AcqList : EmptyAcquaintanceList, WebSite : BA  >
       =>
        < Browser : WebAgent | PlayRole : BROWSER, State : WorkB, MBox : NotEmpty,  AcqList :
```

```
EmptyAcquaintanceList, WebSite : BA  >
                        Message(Browser, Client, websitehasbeenentered) .
rl[websitehasbeenentreted] :
          Message(Browser, Client, websitehasbeenentered)
          < Browser : WebAgent | PlayRole : BROWSER, State : WorkB, MBox : NotEmpty,  AcqList : EmptyAcquaintanceList, WebSite : BA  >
          =>
          < Browser : WebAgent | PlayRole : BROWSER, State : WorkB, MBox : NotEmpty,  AcqList : EmptyAcquaintanceList, WebSite : BA  >
          Message(Browser, Client, websiteresponse) .
endom)
```

Figure 64 : Le module orienté objet CONSULT-ACCOUNT-FROM-INTERNET

✍ **Module RELOAD-MONEY-TICKETS :** Ce module (figure 65) montre à travers ses règles de réécriture le processus de rechargement des billets accompli par l'interaction entre l'agent receveur de la banque (Recipient) et l'agent ATM.

```
(omod RELOAD-MONEY-TICKETS is
...
rl [1] :                  Message(Atm, Recipient, askreloading)
  < Recipient : Agent | PlayRole : RECIPIENTBANK, State : WorkR, MBox : NotEmpty, AcqList : Atm >
  =>
  < Recipient : Agent | PlayRole : RECIPIENTBANK, State : WorkR, MBox : NotEmpty, AcqList : Atm >
                          Message(Recipient, Atm, reloadfinished) .
rl [2] :                  Message(Recipient, Atm, reloadfinished)
   < Atm : Agent | PlayRole : ATM, State : BreakDownA, MBox : NotEmpty, AcqList : Recipient >
              =>
   < Atm : Agent | PlayRole : ATM, State : EndOperationA, MBox : NotEmpty,  AcqList : Recipient >
                          Message(Atm, Recipient, Distributorreloaded) .
rl [3] :                  Message(Atm, Recipient, Distributorreloaded)
     < Recipient : Agent | PlayRole : RECIPIENTBANK, State : WorkR, MBox : NotEmpty, AcqList : Atm >
          =>
     < Recipient : Agent | PlayRole : RECIPIENTBANK, State : EndOperationR, MBox : NotEmpty, AcqList : Atm > .
endom)
```

Figure 65 : Le module orienté objet RELOAD-MONEY-TICKETS

Tous les modules présentés au dessus décrivant les différents cas d'utilisation du système bancaire sont importés dans le module orienté objet principal MAS-FUNCTIONAL-

REQUIREMENTS de la figure 66 et ce dans l'objectif de décrire les besoins fonctionnels du système en entier.

```
(omod MAS-FUNCTIONAL-REQUIREMENTS is
including AUTHENTICATION .
including WITHDRAW-MONEY .
including CHECK-AMOUNT-IN-ACCOUNT .
including TRANSFER-MONEY .
including CONSULT-ACCOUNT .
including CONSULT-ACCOUNT-FROM-INTERNET .
including RELOAD-MONEY-TICKETS .
endom)
```

Figure 66 : Le module orienté objet MAS-FUNCTIONAL-REQUIREMENTS

VI. La validation de la spécification générée

La spécification formelle générée dans le cadre de notre approche a été validée à l'aide de l'outil Full Maude 2.4, intégré dans l'environnement ECLIPSE. La logique de réécriture offre une grande souplesse en termes de simulation d'une spécification, en particulier, la liberté de choisir la configuration initiale. Ce choix joue un rôle primordial dans la validation de la description d'un système. En utilisant toute la description du système, nous pouvons valider une partie du système sans mettre en cause le reste.

Pour une validation de la description Maude générée, nous proposons de valider deux cas d'utilisations essentiels : le transfert d'argent et la consultation de compte travers internet.

La figure 67 illustre une partie de la spécification générée. Elle visualise d'une part, le module orienté objet TRANSFER-MONEY de la figure 62, et d'autre part, la réécriture illimitée d'une configuration initiale. Dans cette configuration, nous avons l'agent jouant le rôle ATM dans son état initial *StartA,* avec une boîte aux lettres vides et a comme accointances l'agent Recipient. Par ailleurs, cette configuration contient deux objets (Acc et Acc1) décrivant deux comptes bancaires dont les contenus sont respectivement 100 et 200 (euro par exemple). La transaction bancaire décrite par ce module, sert à transférer la somme de 40 euro à partir de *Acc* vers *Acc1*. Cette opération nécessite tout d'abord l'insertion de la carte bancaire, ce qui décrit dans cette configuration par le message *Message(Client, Atm, InserCard).*

Le résultat de la réécriture illimitée montre que le solde du compte bancaire *Acc* ait diminué de 40 euro (il est devenu 60 euro), tandis que le solde du compte bancaire *Acc1* a augmenté par la même somme, d'où il est devenu 240.

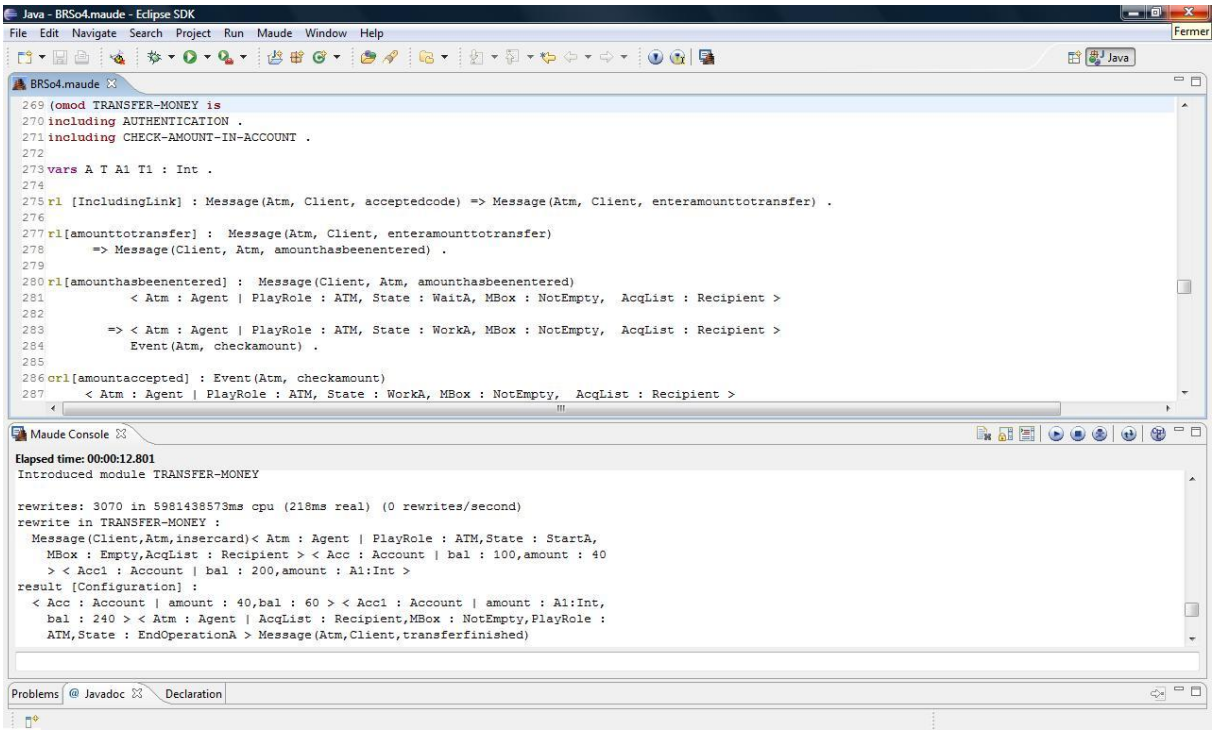

Figure 67 : validation du processus de virement bancaire

Nous avons aussi validé le processus de consultation de compte bancaire à travers Internet. La figure 68, montre le module CONSULT-ACCOUNT-FROM-INTERNET. Elle montre également la réécriture illimitée d'une configuration initiale et le résultat obtenu. La configuration initiale décrit l'agent jouant le rôle BROWSER dans son état initial, avec une boîte aux lettre vide et une liste d'accointance vide (dans cet exemple le browser n'a pas d'accointances), avec comme exemple, le site web de la banque (WWW.BEA.DZ). L'arrivée du message *Message (Client, Browser, requestbrowser)* sert à initialiser le processus de consultation à travers internet. Le résultat exprime que processus de consultation a terminé avec succès en affichant le résultat au client.

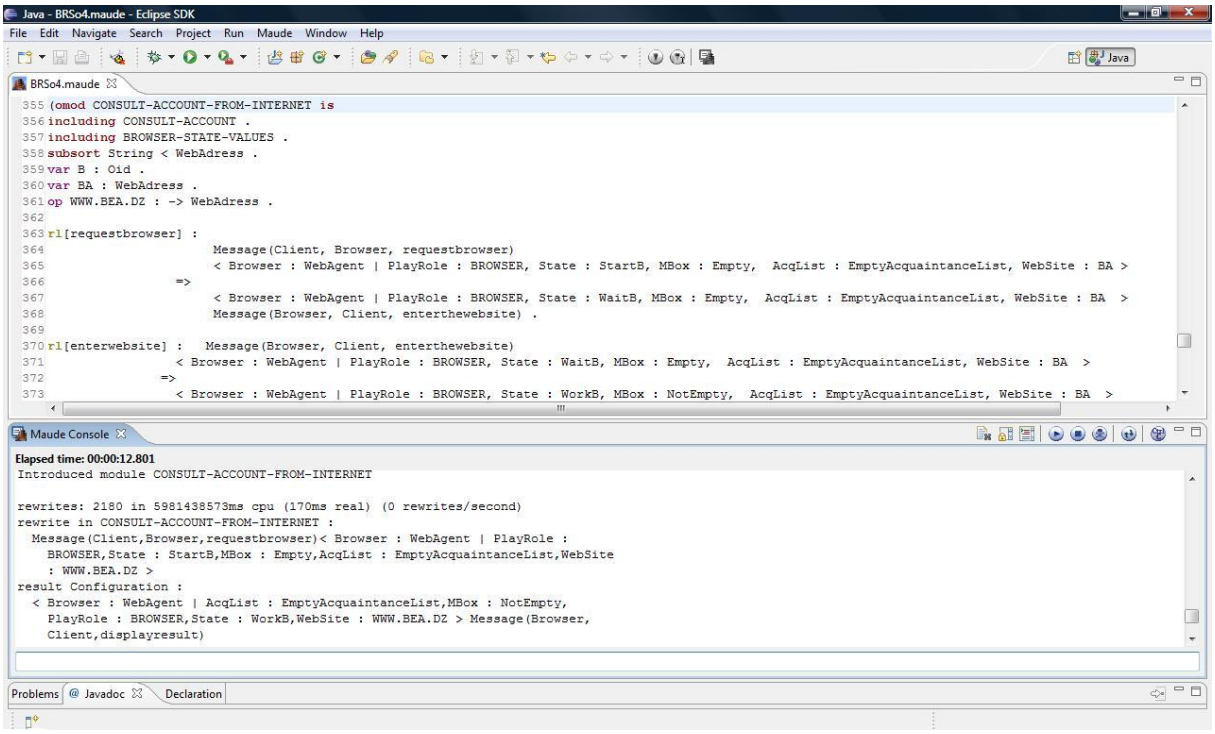

Figure 68 : validation du processus de consultation de compte bancaire à travers Internet

VII. Conclusion

La spécification des besoins fonctionnels de systèmes complexes, en l'occurrence, les systèmes multi-agents, est une activité très importante dans le développement de logiciel de qualité. Notre approche permet d'une part de capturer les besoins fonctionnels d'un SMAs à l'aide de cas d'utilisation d'UML étendu et diagramme de séquence AUML, et d'autre part de translater cette description graphique dans une description formelle écrite en Maude. Cette description nous offre plusieurs avantages comme la compréhension approfondie du système est d'autres avantages assez intéressants. Par ailleurs, elle offre une base solide pour les processus de vérification du système.

CONCLUSION ET PERSPECTIVES

La phase analyse représente une activité très importante dans le cycle de développement des applications logiciels puisqu'elle a comme but d'éviter de produire des applications qui ne répondent pas aux besoins d'utilisateurs. Plusieurs approches ont été proposées pour spécifier les besoins fonctionnels des usagés, parmi elles on trouve les spécifications formelles qui permettent de nous guider à prévoir une bonne qualité du futur produit logiciel. A ce stade, quelques travaux ont été émergés dans la littérature, décrivant les besoins fonctionnels de SMAs de façons informelles ou à la limite semi-formelles. Dans ce mémoire, nous proposons une approche générique permettant, d'une part, de capturer les besoins fonctionnels d'un SMA à l'aide d'une description conjointe de besoins fonctionnels à l'aide de diagrammes de cas d'utilisation et de diagramme de séquence AUML et d'autre part, de translater cette description graphique dans une description formelle Maude. Le langage Maude est supporté par un outil, ce qui nous permettra de valider la spécification générée via une simulation.

Nous envisageons, comme perspectives de notre travail les points suivants :

- Tenir compte des aspects structurels des SMA.
- Application de notre approche aux systèmes basés sur des modèles d'agents bien déterminés.
- Étendre notre approche afin de supporter les systèmes multi-agents ouverts où le nombre des agents n'est pas connu a priori.

Bibliographie

[Abr80] J-R. Abrial. " The Specification Language Z: Syntax and Semantics ". Programming Research Group Oxford University 1980.

[Ada99] E. Adam, C. Kolki. " Étude comparative de méthodes de génie logiciel en vue du développement de processus administratifs complexes ". 1999.

[Afi98] AFIA/PRC-13. " Systèmes Multi-agents, Architectures de Systèmes d'Agents ". 1998

[Ala88] H. B. Alan and L. Gasser. " An analysis of problems and research in DAI ". In Alan H. Readings in Distributed Artificial Intelligence, pages 3–36. Morgan Kaufmann Publishers: San Mateo, CA, 1988.

[Alo04] F. Alonso, S. Frutos, L. Martinez and C. Montes. " SONIA : A Methodology for Natural Agent Development". Proceedings of the 5th International Workshop on Engineering in the Agents World 2004.

[Att92] A.Attoui. " L'utilisation de la logique de réécriture pour la spécification et la validation des systèmes d'informations ". Congrès INFORSID92. Clermont-FD, 19-22 Mai 1992.

[Bar95] M. Barbuceanu et M. S. Fox. " COOL: a language for describing coordination in multi-agents systems". ICMAS, P. 17-24, 1995.

[Bau01] B. Bauer, J. P. Müller, J. Odell. " Agent UML: A Formalism for Specifying Multiagent Software Systems ". International Journal on Software Engineering and Knowledge Engineering (IJSEKE), Vol. 11, No. 3, pp.1-24, 2001 Engineering, 2001.

[Bon88] A. H. Bond et L. Gasser. " Reading in Distributed Artificial Intelligence ". Morgan Kaufmann, San Mateo Californie, 1988.

[Boo91] G. Booch. " Object-Oriented Design with Applications ". Benjamin/Cummings, Redwood City, CA, 1991.

[Bor96] P. Borovansky, C. Kirchner and H. Kirchner, PE. Moreau, and M.Vettek. " ELAN: A logical framework based on computational systems ". In J. Mesegue, editor, proc. First Int. workshop on rewriting logic and its Applications, volume 4 of electronic Notes in Theoritical Computer Science. Elsevier, 1996.

[Bra87] M. Bratman. " Intention, plans, and practical reason ". Harvard University Press, 1987.

[Bra88] M. Bratman, D. Israel, and M. Pollack. " Plans and resource bounded practical reasonning. Computational Intelligence ". 4, pages 349-355, 1988.

[Bri01] J. Briot et Y. Damazeau. " Principes et architecture des systèmes multi-agents ". Hermès Sciences Publications, 2001.

[Buh95] R.J.A. Buhr. " Use Case Maps: A New Model to Bridge the Gap Between Requirements and Detailed Design «. OOPSLA'95 Real Time Workshop, October 1995, p. 4

[Bur96] B. Burmeister. " Models and methodology for agent-oriented analysis and design ". In K Fischer, editor, Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems. DFKI Document D-96-06. 1996.

[Bus04] S. Bussmann, N.R Jennings and M.Wooldridge. " Multiagent Systems for Manufacturing Control, A Design Methodology ". Springer Verlag 2004.

[Cha01] D. B. Chaib et I. Jarras et B. Moulin. " Systèmes Multi-Agents : Principes généraux et application ". 2001.

[Cha02] D. B. Chaib et I. Jarras. " A perçu sur les Systèmes Multi-Agents ". Série scientifique 2002s-67 Montréal Juillet 2002.

[Cha99] D. B. Chaïb. " Agents et Systèmes Multiagents ". (IFT 64881A). Notes de cours. Département d'informatique, Faculté des sciences et de génie, Université Laval, Québec. Novembre 1999.

[Cla96] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. " Principles of Maude ". In José Meseguer, editor, Procedings of the first International work group on Reewriting Logic, volume 4 of electronic notes in Theoretical computer science, page 65-89. Elsevier, September 1996.

[Cla01] D. Claude. " Systèmes d'aide a la décision temps réel et distribuer : modélisation par agents ". Thèse doctorat présentée à l'université du Havre. Spécialité : Informatique. Présenté et soutenue publiquement le 5 octobre 2001.

[Cla09] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, C. Talcott. " Maude Manual (Version 2.4) ". October 2008 (Revised February 2009) .

[Col96] A. Collinot, A. Drogoul, and P. Benhamou. " Agent oriented design of a soccer robot team ". In Proceedings of the Second International Conference on Multi-Agent systems(ICMAS-96),pages 41-47, Kyoto, Japan, December 1996.

[Dan07] D.H Dang. " Validation of System Behavior Utilizing an Integrated Semantics of Use Case and Design Models ". In Claudia Pons, editor, Proceedings of the Doctoral Symposium at the ACM/IEEE 10th International Conference on Model-Driven Engineering Languages and Systems (MoDELS 2007). Nashville (TN), USA, October 1st, 2007. CEUR, Vol-262, 2007.

[Del01] S. A. DeLoach, M. Woodldridge and C. H. Sparkman. " Multiagent Systems Engineering ". The International Journal of Software Engineering and Knowledge Engineering, 11(3), pp. 231-258, June 2001.

[Dem95] Y. Demazeau. " From Interactions to Collective Behaviour in Agent-Based Systems ". Proceeding of the First European Conference on Cognitive Science, Saint-Malo, 1995, p. 117-132.

[Duk95] Duke, Roger, Rose, Gordon, Smith and Graeme. " Object-Z: a Specification Language Advocated for the Description of Standards ". Computer Standards and Interfaces. University of Queensland, Australia. 1995.

[Dur89] E. H. Durfee et V. R Lesser. " Negotiating task decomposition and allocation using partial global planning ". dans L. Gasser et M. Huhnes (Eds), Distributed Artificial Intelligence Volume II, Pitman Publishing Morgan Kaufman, 1989.

[Eal99] M. Ealmmari, W. Lalonde. " An Agent-Oriented Methodology : High-Level and Intermediate Models ". Proceedings of AOIS-1999. Heidelberg (Germany), June 1999.

[Ekk96] R. Ekkart, J. Grabowski, and P. Graubmann. " Tutorial on message sequence charts (MSC) ". In Proceedings of FORTE/PSTV'96 Conference, October 1996.

[Eli96] A. Elisabeth, T. M. Margaret and C. Jiang. " A methodology for developing agent based systems for enterprise integration ". In D. Luckose and Zhang C., editors, Proceedings of the First Australian Workshop on DAI, Lecture Notes on Artificial Intelligence. Springer-Verlag: Heidelberg, Germany, 1996.

[Fer95] J. Ferber. " Les systèmes multi-agents, vers une intelligence collective ". InterEditions, 1995.

[Fer95b] J. Ferber. "Les systèmes multi agents». InterEditions, 1995.

[Fis97] M. Fisher, J. M¨uller, M. Schroeder, G. Staniford, and G. Wagner. " Methodological foundations for agent-based systems ". In Proceedings of the UK Special Interest Group on Foundations of Multi-Agent Systems (FOMAS). Published in Knowledge EngineeringReview(12)3,1997,1997.

[Flo02] A. M. Florea. " Agents et Systèmes Multi-Agents ". Université de polytechnique Bucarest. 2002.

[Fut94] K. Futatsugi and T. Sawada. " Cafe as an extensible specification environment ". In Proc. Of Kunming international CASE Symposium, Kunming, China, November, 1994.

[Fut98] K.Futatsugi and R.Diaconescu. " Cafeobj report ". AMAST Series, World scientific, 1998.

[Gas92] L. Gasser and J. P. Briot. " Object-based concurrent processing and distributed artificial intelligence ". Distributed Artificial Intelligence: Theory and Praxis, pages 81–108. Kluwer Academic Publishers: Boston, MA, 1992.

[Geo87] M. P. Georgeff and A. L. Lansky. " Reactive reasoning and planning ". In The Proceedings of AAAI-87, pages 677-682, Seattle, 1987.

[Ger03] M.-P. Gervais. " ODAC : An Agent-Oriented Methodology Based on ODP ". Journal of Autonomous Agents and Multi-Agent Systems, vol. 7, pp. 199-228 2003.

[Giu01] F. Giunchiglia, J. Mylopoulos and A Perini. " The Tropos Software Development Methodology: Processes, Models and Diagrams ". Technical Report DIT-02-008, Informatica e Telecomunicazioni, University of Trento 2001.

[Gla96] N. Glaser. " Contribution to Knowledge Modelling in a Multi-Agent Framework the Co-MoMAS Approach ". PhD thesis, L'Universtit´e Henri Poincar´e, Nancy I, France, November 1996.

[Gli98] P. Glize, M.P. Gleizes et Camps Valérie. " Une théorie de l'apprentissage fondée sur l'auto-organisation par coopération ". 1998.

[Gol94] C. V. Goldman, J. S. Rosenschein. " Emergent Coordination through the Use of Cooperative Stage-Changing Rule ". Proceedings of the Twelve National Conference on Artificial Intelligence, Seattle, Washington. 1994.

[Hin95] M. G. Hinkey and J. P. Brown. " Application of Formal Methods ". Prentice-Hall, 1995.

[Hog93] T. Hogg, B. A. Huberman. " Better than the best : The power of cooperation, Lectures in complex systems ". pp. 165-184, Addison-Wesley Addison Wesley 1993.

[Huh87] M. Huhns, U. Mukhopadhyay, and L. M. Stephens. " DAI for document retrieval : The MINDS project ". In M. Huhns, editor, Distributed Artificial Intelligence, pages 249-284. Pitman Publishing : London and Morgan Kaufmann : San Mateo, CA, 1987.

[Igl99] C. A. Iglesia, M. Garijo and J.C. Gonzalez. " A survey of agent oriented methodologies ". Proceedings of the 5th International Workshop on Agent Theories, Architectures, and Languages, MÜLLER, J.P., SINGH, M.P. and RAO, A.S. (Eds) 1999.

[Iso89] ISO 8807. " LOTOS, A Formal Description Technique Based on the Temporal Ordering of Observational Behavior ". 1989.

[Itu94] ITU-T. Z100 (1993). CCITT. " Specification and description language (SDL) ". Technical report, ITU-T, June 1994.

[Jac92] I. Jacobson, M. Christerson, P. Jonsson, and G. O¨ vergaard. " Object-Oriented Software Engineering. A Use Case Driven Approach ". ACM Press, 1992.

[Jen98] N.R. Jennings, K. Sycara, and M. Wooldridge. "A Roadmap of Agent Research and Development ". Int Autonomous Agents and Multi-Agent Systems, vol. 1, n° 1, pp. 7-38. 1998.

[Joe05] Q. Joël. " Introduction aux systèmes multi-agents ". LIRMM et CERIC Montpellier France 2005.

[Kin96] D. Kinny, M. Georgeff, and A. Rao. " A methodology and modelling technique for systems of BDI agents ". In W. van der Velde and J. Perram, editors, Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World MAAMAW'96, (LNAI Volume 1038). Springer-Verlag: Heidelberg, Germany, 1996.

[Kir95] C. Kirchner and H Kirchner, and M.Vittek. " Designing constraint logic programming language using computational systems ". In V.saraswet and P van henteryck, Editors, Principales and practice of constraint programming. The new papers, pages 133-160, MIT press, 1995.

[Kir98] C. Kirchner and H Kirchner. " Rewriting Logic and its Applications ". (eds.). proc. 2nd Int Workshop ENTCS, North Holland, 1998.

[Lab06] O. Labarthe. " Modélisation et Simulation Orientées Agents de Chaines Logistiques dans un Contexte de Personnalisation de Masse : Modèles et Cadre Méthodologique ". Thèse doctorat Discipline : Informatique Université Paul Cézanne AIX-MARSEILLE III présentée et soutenue publiquement le 30 Octobre 2006.

[Lab93] S. Labidi et W. Lejouad. " De l'intelligence Artificielle distribuée aux Systèmes Multi-agents ". Rapport de Recherche de l'INRIA N° 2001, 1993.

[Lau07] Laurent AUDIBERT. " UML 2.0 ". Institut Universitaire de Technologie de Villetaneuse Département informatique.

[Mar03] B. Marjorie. " Un Simulateur Multi-Agent pour l'Aide à la Décision d'un Collectif : Application à la Gestion d'une Ressource Limitée Agro-environnementale ". Thèse doctorat discipline informatique Université Paris IX-Dauphine Page 62 mai 2003.

[Mar93] N. Marti-Oliet, J. Messeguer. " Rewriting logic as a logical and semantic framework ". report SRI-CSL 93-05, Menlo Park, CA94025, and Center for the study of language and Information Stanford University, Stanford, CA 94305. 1993.

[Mar95] N. Marti-Oliet, J. Messeguer. " From Abstract Data types to logical Framework ". In E. Astesiano, G.Reggio (eds.) : Recent Trends in Data Type Specification, lecture notes in computer science, 906, Springer-Verlag, 1995, pp.48-80.

[Mar96] C. Marie, G. Bruno, S. Françoise, B. Gille. " Précis de génie Logiciel ". Edition MASSON ISBN 1995.

[Mes00] 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, FMOODS2000, 2000.

[Mes90] J. Meseguer. " Rewriting as a unified model of concurrency ". In Proceedings of the Concur'90 Conference, Amsterdam, Pg 384-400, Springer LNCS Vol. 458, 1990.

[Mes92] J. Meseguer. " Conditional rewriting logic as a unified model of concurrency ". Theoretical Computer Science, 96:73–155, 1992.

[Mes96] J. Messeguer. " rewriting logic and its applications ". (ed). Proc. First Intl. Workshop ENTCS North Holland, 1996.

[Mes98] J. Meseguer. " Membership algebra as a logical framework for equational specification ". In Francesco Parisi-Presicce, editor, Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT'97, Tarquinia, Italy, June 3, 1997, Selected Papers, volume 1376 of Lecture Notes in Computer Science, pages 1861. Springer, 1998.

[Mir92] E. Miriad. " Approcher la notion de collectif ". In Journée Systèmes Multi-Agents PRC-GDR Intelligence Artificelle, Nancy, Décembre 1992.

[Mok07] F. Mokhati. " Vérification et Validation des Systèmes Multi-Agents : Une approche formelle et fédérative ". Thèse de doctorat, université Badji Mokhtar-Annaba, janvier 2007.

[Mou96] B. Moulin et B. D. Chaib. " An overview of distributed artificial intelligence ". dans G.M.P. O'Hare et N. R. Jenning (Eds), foundation of distributed AI, 3-54, John Wiley & Sons, Chichster, Grand-Bretagne, 1996.

[OMG07] Unified Modeling Language: Superstructure version 2.1.1 by Object Management Group (with change bars) formal/2007-02-03.

[Pad02] L. Padgham and M. Winikoff. " Prometheus: A Methodology for Developing Intelligent Agents ". In proceedings of the Third International Workshop on Agent-Oriented Software Engineering, at AAMAS'02, July 2002.

[Pet05] C. O. Peter. " Formal Modeling and Analysis of Distributed System in Maude ". Course notes/book for the introductory formal methods course INF 3230 - Formal modeling and analysis of communicating systems at the Department of Informatics, University of Oslo. Nov29, 2005.

[Qyu05] N. T. Quynh-Nhu, C. Graham. " Comparison of Ten Agent-Oriented Methodologies ". Agent oriented methodologies -Idea Group Publishing USA- ISBN 1-59140-587-4 (ebook) 2005.

[Rao91a] A. S. Rao and M. P. Georgeff. " Asymmetry thesis and side-effect problems in linear time and branching time intention logics ". In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91), pages 498-504, Sydney, Australia, 1991.

[Rao91b] A. S. Rao and M. P. Georgeff. " Modeling rational agents within a BDI architecture ". In R. Fikes and E. Sandewall, editors, Proceedings of Knowledge Representation and Reasoning (KR&R-91), pages 473-484. Morgan Kaufmann Publishers: San Mateo, CA, April 1991.

[Rao92] A. S. Rao and M. P. Georgeff. " An abstract architecture for rational agents ". Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning, pages 439-449, Cambridge, MA, 1992. Morgan Kaufmann.

[Rob99] A. Roberto F. Mendez. " Towards the standardization of Multi-Agent Systems Achitectures : An Overview ". ACM Crossroads'special issue on Intelligent Agents, summer 1999.

[Rsc97] Rational Software Corporation. " Unified Modelling Languaje (UML) version 1.0 ". Rational Software Corporation, 1997.

[Rum91] J. Rumbaugh, M.Blaha, W. Premerlani, F. Eddy, and V. Lorensen. " Object-Oriented Modeling and Design ". Prentice-Hall, 1991.

[Rus95] S. J. Russell and P. Norvig. "Artificial Intelligence : A Modern Approach". Prentice Hall, (second edition 2003). ISBN 0137903952. 1st edition, January 1995.

[Rus97] S. J. Russell. " Rationality and intelligence ". journal of Artificial Intelligence, Vol 94 pages 57-77. 1997.

[Sab01] A. Sabas. " Systèmes Multi-Agents : Une Analyse Comparative des Méthodologies de Développement Vers la convergence des méthodologies de développement et la standardisation des plateformes SMA ". Mémoire Présenté Comme exigence partielle de la Maîtrise en mathématiques et informatique appliquées à l'Université du Québec à Trois-Rivières Octobre 2001.

[Sch94] A. T. Schreiber, B. J. Wielinga, J. M. Akkermans, and W. Van de Velde. " CommonKADS: A comprehensive methodology for KBS development ". Deliverable DM1.2a KADSII/ M1/RR/UvA/70/1.1, University of Amsterdam, Netherlands Energy Research Foundation ECN and Free University of Brussels, 1994.

[Sco99] A. D. Scott. " Multiagent Systems Engineering : A Methodology And Language for Designing Agent Systems ". 1999.

[Sek95] M. Sekaran, S. Sen. " To help or not to help ". Seventeenth Annual Cognitive Sciences Conference, Pitsburg, Pennsylvannia. 1995.

[Sho93] Y. Shoham. " Agent-oriented programming. Artificial Intelligence ". 60(1):51–92, March 1993.

[Sin94] M. P. Singh. " Multi-agent Systems: A Theoretical Framework for Intentions ". Know-How,and Communications (LNAI Volume 799). Springer-Verlag: Heidelberg, Germany, 1994.

[Tai03] T. Taibi, D. C. L. Ngo. " Formal Specification of Design Patterns – A Balanced Approach ". Journal of Object Technology, Vol. 2, No. 4, July-August 2003, pp. 127-140.

[Tra01] E. Tranvouez. " IAD et ordonnancement : une approche coopérative du ré-ordonnancement par systèmes multi-agents ". Thèse de Doctorat de L'Université de Droit, d'Economie et des Sciences d'Aix-Marseille III, mai 2001.

[Wir90] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. " Designing Object-Oriented Software ". Prentice-Hall, 1990.

[Woo98] M. Woodridge and N. R. Jennings. " Pitfalls of Agent-oriented Development ".In Proceedings of Second International Conference on Autonomous Agents (Agent98) è Minneapolis/St Paul, MN, May 1998.

[Woo00] M. Wooldridge, N.R. Jennings, and D. Kinny. " The Gaia Methodology For Agent-Oriented Analysis and Design ". Journal of Autonomous Agents and Multi-Agent Systems 3 (3) 285-312. 2000.

[Woo00a] M. Wooldridge and N. R. Jeennings. " Agent-Oriented Software Engineering ". in Handbook of Technology (ed. J. Bradshaw) AAAI/MIT Press 2000.

[Woo00b] M. Wooldridge, G. Weiss. " Multiagent Systems A Modern Approach to Distributed Artificial Intelligence ". Intelligent Agents, editor: G. Weiss, MIT Press, pages 27--77., 2000.

[Woo02] M. Wooldridge, " Introduction to multi agents systems ". John Wiley& Sons, 2002.

[Zam03] F. Zambonelli, N.R. Jennings and M.Wooldridge. " Developing Multiagent Systems: The Gaia Methodology ". ACM Transactions on Software Engineering and Methodology, vol. 12, n° 3, pp. 317-370. 2003.