République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Larbi Tébessi - Tébessa
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département : Mathématiques et Informatique

Mémoire de fin d'étude
Pour l'obtention du diplôme de MASTER
Domaine : Mathématiques et Informatique
Filière : Informatique
Option : Systèmes d'information

Thème

Un modèle formel des besoins fonctionnels des systèmes multi-agents à l'aide de Maude

Présenté Par :
Cheriet Amira

Devant le jury :

Mr Haouam Med Yassine	*MCB*	*Univérsité Larbi Tébessi - Tébessa*	*Président*
Mr Menassel Yahia	*MAA*	*Univérsité Larbi Tébessi - Tébessa*	*Examinateur*
Mr Hamidane Fathi	*MAA*	*Univérsité Larbi Tébessi - Tébessa*	*Encadreur*

Date de soutenance : 22/06/2019

بسم الله الرحمن الرحيم

Je dédicace ce travail

A ma mère et A mon père

Auxquels je dois ce que je suis. Que dieu protège mon père, et dieu bénisse ma mère.

A mes sœurs ***Rayene, Afnane, Zaineb.***

A mon frère ***Ahmed.***

A ma tante Souad.

À mes chères copines.

A mes amis

Mes collègues.

Et A une personne qui m'a donnée l'effort pour que je travail, un spéciale remerciement pour tout.

A toute ma famille.

« Amira Cheriet »

Résumé

Victor Hugo

« *L'éducation, c'est la famille qui la donne ; l'instruction, c'est l'Etat qui la doit. »*

Ce mémoire présente une approche systématique permettant de translater les besoins fonctionnels d'un système multi-agents décrits par les diagrammes de la méthodologie PASSI à savoir : modèle de description des besoins, modèle d'identification d'agents, modèle d'identification de rôle et modèle d'identification de taches dans une spécification formelle Maude. Notre approche propose un processus de translation à partir des modèles cités auparavant vers une spécification formelle écrite en langage Maude. 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 présentés dans la méthodologie PASSI à l'aide de Maude, et (2) l'intégration de la validation formelle et de la cohérence des modèles, dès la phase d'élicitation des besoins, dans un processus de développement des SMAs.

Mots Clés :

Besoins fonctionnels, méthodologie PASSI, SMA, Spécification formelle, Agent UML, Logique de Réécriture, Maude.

Abstract

This thesis presents a systematic approach for translating the functional needs of a multi-agent system described by the PASSI methodology diagrams, namely: requirements description model, agent identification model, role identification model. and pattern identification tasks in a Maude formal specification. Our approach proposes a translation process based on the models mentioned above to a formal specification written in Maude language. Once elaborated, the different diagrams are validated to ensure inter and intra model consistency. The Maude language, based on the rewriting logic, provides a solid and formal basis for the specification and programming of competing systems. The main motivations of this work are: (1) the formalization of the functional needs of a multi-agent system presented in the PASSI methodology using Maude, and (2) the integration of formal validation and coherence. models, right from the elicitation of needs phase, in a process of developing MAS.

Key words :

Functional requirements, PASSI methodology, MAS, Formal specification, Agent UML, Rewriting logic, Maude.

ملخص

تقدم هذه الرسالة مقاربة منهجية لترجمة الاحتياجات الوظيفية لنظام متعدد الوكلاء الموصوف في مخططات منهجية باسي ، وهي: نموذج وصف المتطلبات ، ونموذج تحديد الوكيل ، ونموذج تحديد الدور. ومهام تحديد النمط في مواصفات مود الرسمية. يقترح نهجنا عملية ترجمة تستند إلى النماذج المذكورة أعلاه لمواصفات رسمية مكتوبة بلغة مود. مرة واحدة وضعت ، يتم التحقق من صحة المخططات المختلفة لضمان الاتساق بين وداخل النموذج. توفر لغة مود ، استنادًا إلى منطق إعادة الكتابة ، أساسًا متينًا رسميًا لمواصفات وبرمجة الأنظمة المتنافسة. الدوافع الرئيسية لهذا العمل هي: (1) إضفاء الطابع الرسمي على الاحتياجات الوظيفية لنظام متعدد الوكلاء المقدمة في منهجية باسي باستخدام مود ، و (2) دمج التحقق الرسمي والتماسك. النماذج ، مباشرة من مرحلة استنباط الاحتياجات ، في عملية تطوير الانظمة متعددة الاعوان.

الكلمات الدالة *:*

الاحتياجات الوظيفية, منهجية باسي, مواصفات رسمية, نظام متعدد الأعوان عون لغة النموذج الموحدة, منطق إعادة الكتابة, مود.

Remerciement :

Mireille Sitbon :

« L'informatique, ça fait gagner beaucoup de temps... à condition d'en avoir beaucoup devant soi ! »

Mes remerciements s'adressent à Mon encadrant de projet Mr ***Hemidane Fathi*** *pour ses précieux conseils et leur travail, qui m' a donné tout le soutien moral et technique nécessaire.*

Mes remerciements s'adressent à tout le staff de la banque extérieur d'Algérie, Agence de Tébessa et spécialement à mon encadrant de Stage, qui nous a donné tout le soutien moral et technique nécessaire pour que ce travail sera lieu aujourd'hui.

Mes vifs remerciements à mes professeurs qui ont cru en moi, qui m' ont apporté tout le soutien nécessaire et n'ont pas cessé de saturé l'avidité technique par leurs connaissances en Informatique pendant cinq ans d'études.

Mon remerciement va aussi à tous les membres de l'équipe de master système d'information.

Mes remerciements s'adressent à tous les membres de jury pour leur temps, pour qu'ils jugent mon travail.

Un grand merci va à tout le staff de la faculté de Science Exacte et spécialement à toute l'équipe de département de Math et Informatique.

Un grand merci s'adresse à ma famille pour sa précieux conseils, qui m'a témoigné mon travaille pendant les 5 ans et m'a donné tout le soutien moral.

Enfin, un merci spécial à tous nos collègues d'études pour leurs énormes efforts pendant toutes les années d'études qu'on a passé ensemble.

Table des matières

Liste des figures

Liste des tableaux

Introduction Générale :

Dave Barry :

« Le cycle d'obsolescence des ordinateurs est devenu si rapide, que dans les hypermarchés informatiques du futur, il y aura des décharges juste en face des caisses enregistreuses. »

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 permettant le passage d'un modèle d'écrivant les besoins fonctionnels des SMA via la méthodologie PASSI[Cos et al 14] vers une spécification dite formelle écrite en 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 quatre diagrammes de la méthodologie PASSI à savoir : modèle de description de domaine, modèle d'identification d'agents, modèle d'identification de rôles, modèle d'identification de taches. Dans la deuxième étape, les diagrammes considérés doivent subir une validation inter-modèle 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 manuscrit :

Le reste de ce manuscrit est organisé en quatre chapitres :

Chapitre 1 : présente une vue sur la logique de réécriture et le langage Maude.

Chapitre 2 : Le deuxième chapitre présente les notions de base des systèmes multi-agents ainsi que les méthodologies d'analyse et de conception des SMAs.

Chapitre 3 : Une étude comparative de méthodologies présentées dans le chapitre 2.

Chapitre 4 : Le quatrième chapitre présente l'approche que nous proposons pour la description formelle des besoins fonctionnels des systèmes multi-agents présenté par la méthodologie PASSI.

.

Chapitre 1:

La logique de réécriture et le langage MAUDE

Jean Dion :
« Le principe de l'évolution est beaucoup plus rapide en informatique que chez le bipède. »

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 c'est une logique qu'était développée par **José Meseguer** et son groupe au laboratoire d'informatique SRI International [1] comme une conséquence des travaux sur les logiques générales. C'est une logique qu'était basée sur une sémantique totale et parfaite, elle permet de décrire les systèmes concurrents. 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 (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].

A. La Théorie de Réécriture :

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

- Σ est une signature algébrique
- E est un ensemble d'équations
- L est un ensemble d'étiquettes (Label)
- R est un ensemble de règles de réécriture, elles sont de la forme :
  - règle de réécriture inconditionnelle : **R : [t] → [t'].**
  - règle de réécriture conditionnelle : **R : [t] → [t'] if Cond.**

B. Règles de Déduction :

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. On utilise ces règles de déductions dans la définition de la théorie 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 $[t] \in T_{\Sigma, E}(X)$:

  $[t] \rightarrow [t']$

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

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

- **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,

  $$\frac{[w_1] \rightarrow [w'_1] \ldots [w_n] \rightarrow [w'_n]}{[u_1(\bar{w}/\bar{x})] \rightarrow [v_1(\bar{w}/\bar{x})] \ldots [u_k(\bar{w}/\bar{x})] \rightarrow [v_k(\bar{w}/\bar{x})]}$$

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

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 :**

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

C. 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

$[t] \rightarrow [t']$:

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 $[t] \rightarrow [t']$ 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 $[t] \rightarrow [t']$ est obtenue après plusieurs étapes ($[t] \rightarrow [t1] \rightarrow \ldots \rightarrow [tn] \rightarrow [t']$) 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) $[t] \rightarrow [t']$ 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.

D'un autre coté, 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 (utilisation des équations et des règles simples).
- **Expressif :** on a la possibilité d'exprimer naturellement un grand champ d'applications s'étendant de simples systèmes déterministes à des systèmes concurrents de grande complexité.
- **Performant :** ce langage est très performent car il s'intéresse de :
    - La spécification exécutable
    - La programmation réelle

On trouve que le langage MAUDE est présent dans plusieurs domaines tel 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**].** Le langage MAUDE existe en deux versions :

- **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.

A. Syntaxe de base du langage MAUDE :

1. Les identificateurs :

Les éléments de base dans chaque langage sont les identificateurs, ils sont utilisés pour nommer des modules. . En général un identificateur dans MAUDE est une chaîne de caractère d'ASCII tel que :

- Ne contient pas d'espace.
- **'{', '}', '(', ') ', '[', '] ' et ','** sont des caractères spéciaux qui séparent une séquence de caractères en différents identificateurs.
- Le caractère ' ' ' indique que le blanc ou les caractères spéciaux ne séparent pas la séquence de caractère, il est situé devant tout caractère spécial ou bien entre deux chaîne vides.

2. Les sorts :

Représentent la déclaration des types de données (généralement appelés *sortes* dans les spécifications algébriques). Les sortes sont organisées via une relation de sous typage *subsort***.**

La syntaxe de déclaration : 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) .

On peut aussi déclaré plusieurs sortes en utilisant le mot clé **sorts,** comme suit :

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

Exemple : sort Zéro .

Ou bien : **sorts** Zéro Nznat Nat .

On peut pas trouvé les caractères ':', '.', ',', '[', ou ']' dans les identificateurs utiliser pour les sortes.

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) .

Plusieurs relations de subsort peuvent être déclarées en utilisant le mot clé **subsorts** , comme suit : **subsorts (Sort1) . . . (Sorti1) < . . . < (Sortn1) . . . (Sortnin) .**

Les sortes les plus utilisées	La définition
Nat	**Les nombres naturels**
Zéro	**La constante 0**
NzNat	**Les nombres naturels différents de zéro**

Table 1.1 : Quelques des sortes utilisés

Ex : subsort Zéro < Nat .

3. Les opérateurs :

Le mot clé **op** c'est le mot utilisé pour la déclaration des opérateurs dans un langage MAUDE

a) Déclaration des opérateurs :

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)] .

ex : op zero : -> Zero .

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

Si la liste d'argument est vide, l'opérateur s'appelle une constante. Le nom de l'opérateur est une chaîne de caractères qui peut se composer de plusieurs identificateurs. Le tirait de soulignement (_) c'est très important 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*

b) Surcharge d'opérateur :

Le mot surcharge des opérateurs dans le langage MAUDE signifie que nous pouvons avoir plusieurs déclarations d'opérateurs pour le même opérateur avec des arguments et des résultats différents.

Ex : **op _+_ : NzNat Nat -> NzNat .**

sort Nat3 .

ops 0 1 2 : -> Nat3 .

op _+_ : Nat3 Nat3 -> Nat3 .

dans cet exemple on peut voir la surcharge d'op _+_ .

4. Les variables :

Le mot clé **var** ou **vars** est utilisé pour déclarer des variables dans MAUDE.

La syntaxe des variables :

Les variables se composent d'un identificateur (le nom de la variable), suivi de deux points (:)avec un espace (blanc) avant et après et d'un autre identificateur (exprimant la sorte). Par exemple : **var N : Nat** signifie que la variable N est de sorte Nat .

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

Ex : **vars N M : Nat .**

5. Les termes :

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

6. Les commentaires :

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

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

B. Les modules de langage MAUDE :

Les modules sont les unités de base dans le langage MAUDE. On trouve trois types de modules dans ce langage :

- Modules fonctionnels
- Modules systèmes
- Modules objets

1. Modules fonctionnels :

a. Définition :

Ce module est basé sur la définition des types de données et des opérations, c'est un programme à caractère équationnel avec une syntaxe qui définit un certain nombre de sortes, leurs éléments et les fonctions sur ces sortes. D'un coté de spécification, on peut dire que ce module est une théorie équationnelle basé sur une sémantique algébrique.

Syntaxe de déclaration :

La déclaration d'un module fonctionnel est comme suit :

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

Chaque module fonctionnel a un nom simple ou composé lié par un trait d'union, qui va jouer le role d'identificateur dans ce langage.

b. Les équations:

On trouve deux types d'équation : équation incondionnelle et équation condionnelle.

- **Equation incondionnelle:**

Le mont clé eq est utilisé pour la déclaration d'une équation 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. La déclaration est comme suit :

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

ex : eq N + zéro = N .

Remarque : il faut que t et t' soyer de même sorte, pour que l'équation soit exécutable, toute variable dans le terme *t'* doive également apparaître dans *t*.

- **Equation condionnelle:**

Le mont clé ceq est utilisé pour la déclaration d'une équation, 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 d'une équation conditionnelle est comme suit :

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

if (EqCondition-1) ∧ . . . ∧ (EqCondition-k) [(StatementAttributes)] .

Exemple : 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) .

c. Les Attributs d'opérateur :

Les attributs d'opérateur sont des attributs utilisés pour donner des informations de plus sur le coté sémantique, syntaxique, pragmatique … On les met les attributs ent '[','] entre la sorte de résultat et le point final.

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

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

Remarque : 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.

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

Exemple : red Barcelone vs RealMadrid .

Le résultat après la réécriture est Barcelone.

2. Le module système:

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].

Cette 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

Remarque : 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 (-**)**.

Ex : mod AUTOMATE-ETAT-FINI is

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

Endm

Dans la partie déclaration on trouve :

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.

La différence entre le module fonctionnel et le module système dans la 6eme partie (la déclaration des règles).

b. Les Règles inconditionnelles :

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. Toutes variables, sortes et opérations utilisées dans la partie droite d'une règle doivent apparaître dans sa partie gauche. La règle inconditionnelle est présentée sous la syntaxe suivante:

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

c. Les Règles conditionnelles :

Une règle conditionnelle est présentée sous la syntaxe suivante:

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

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

Remarque: 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.

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
```

3. Le module orienté objet :

A. Définition :

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].

La représentation des objets sera comme suit : < O : C **| attr-1, attr-2, … , attr-n** >

La représentation d'une spécification d'un objet sera : < O : C **| a1 : v1, a2 : v2, … , an : vn** >

B. Les classes :

le mot-clé ***class*** joue le rôle de déclaration d'une classe**,** suivi par le nom de la classe, la barre `**|** ', et 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 forme finale d'une déclaration sera la suivante :

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

C. Message :

On peut remarqué que la déclaration des messages et des opérations c'est la même chose, on se basant sur le mot clé msg si on a un seul message et msgs si on a plusieurs messages et la sorte de résultat c'est Msg.

Ex : msg from_to_transfer_ : Oid Oid Nat -> Msg .

D. Héritage :

Ce langage supporte la notion de l'héritage en utilisant le mot clé subclass comme suit: *subclass C1 < C2* . Donc la classe fille va acquérir toutes spécifications d'une classe mère.

Ex : class SavingAccount | rate : Float .

subclass SavingAccount < Account .

Remarque :

- dans Full-Maude, les modules sont mis entre parenthèses
- l'exécution de la spécification écrite à l'aide d'un module O.O possède la structure suivante : (rew <Configuration> .)

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)
```

4. 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]. Il existe 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 2:

Les systèmes multi-agents

Griffo :

« L'informatique n'est qu'un outil, comme un pinceau ou un crayon. »

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].

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. Les systems multi-agents:

A. Le concept d'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.

La définition reprise dans la communauté des systèmes multi-agents est la suivante: "un agent est une entité physique ou virtuelle capable d'agir (ou communiquer) sur son environnement de manière autonome et ce en fonction de perceptions et connaissances partielles. » [Woo00a]

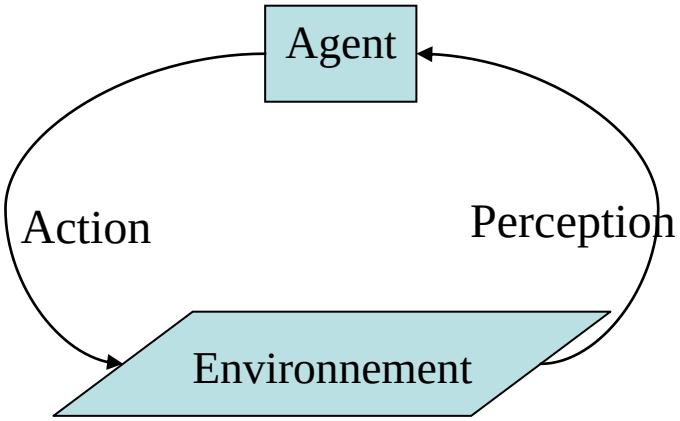


Figure 2.1. Description d'un agent [Woo00a]

Chaque agent a un comportement guidé par un ou plusieurs objectifs individuels, la présence de ce dernier dans un environnement est afin d'effectué une ou plusieurs taches selon ces compétences.

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.

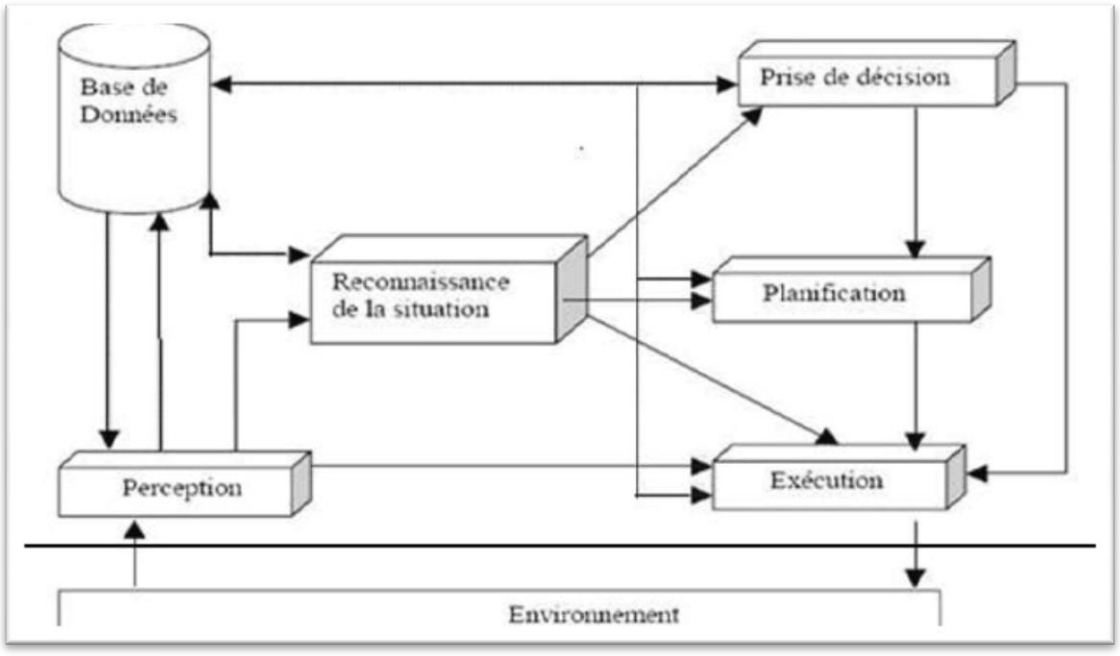


Figure 2.2. Architecture d'agent [Flo02].

4. 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).

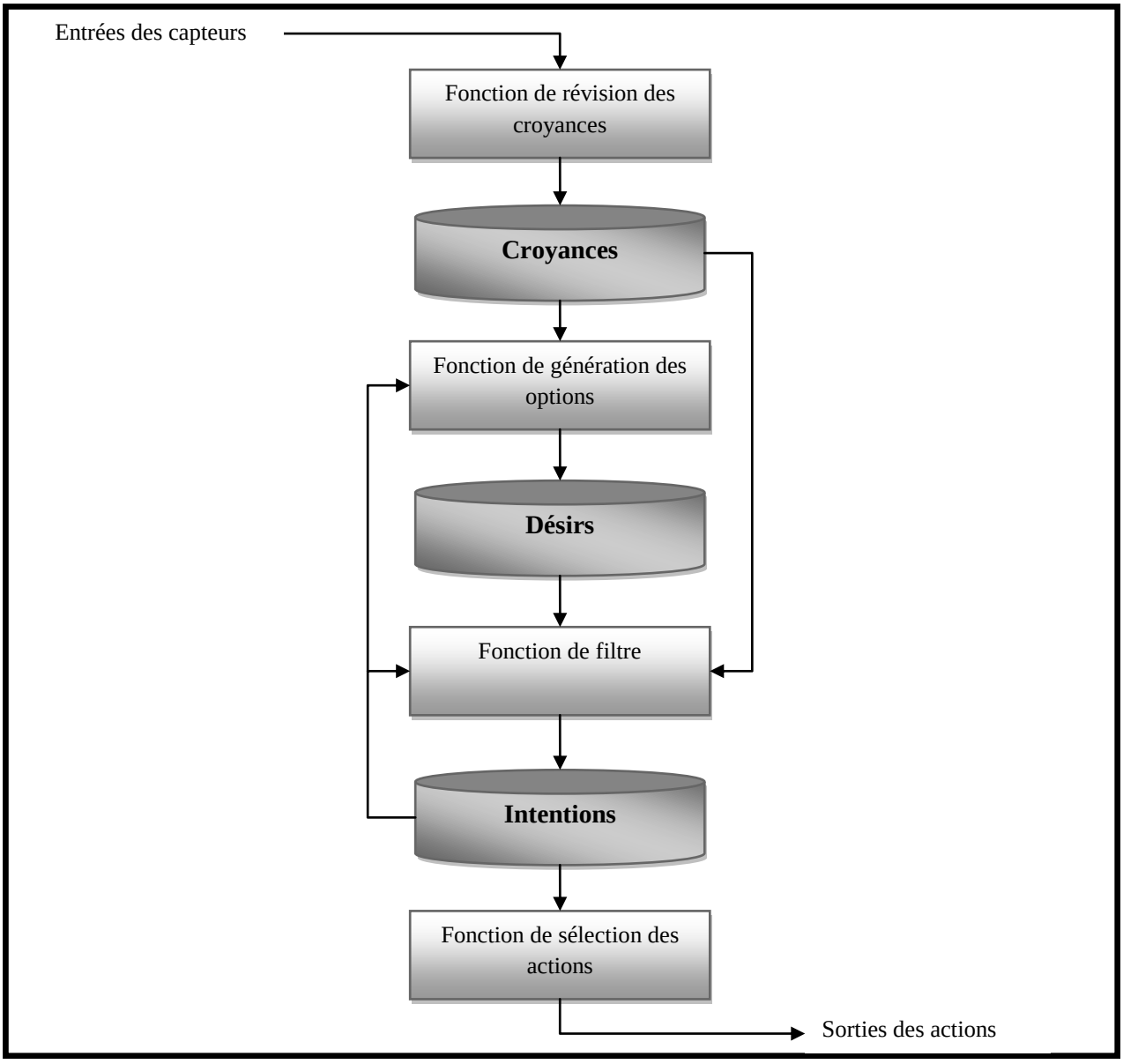


Figure 2.3. Architecture BDI d'un agent[Bra87, Bra88, Geo87, Rao91a, Rao91b, Rao92, Sin94].

5. 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.

B. Les systèmes multi-agents :

La présence d'un groupe des agents dans un ensemble ce n'est pas l'équivalence d'un système multi-agent. Un système multi-agent c'est-à-dire la prise en compte de problématiques organisationnelles et collectives portant sur les entités, la dynamique et la structure [Arl04].

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] :

- ***A*gents :** 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.
- ***O*rganisations :** 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).

La différence entre un système mono-agent et un système multi-agent que on peut trouve un agent a un rôle ou une tache à remplir vis-à-vis des autres agents.

Pour que on peut réaliser la dynamique du système on doit pris en compte :

Un système multi-agent est donc un système composé d'agents autonomes ayant pour but de fournir une fonction collective. Les mécanismes permettant d'obtenir cette fonction prennent en compte les enjeux sociaux entre agents, comme la coopération.

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 de développement 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 multi-agents [Igl99].

A. Les méthodologies issues de l'ingénierie de connaissance :

A ce niveau la plusieurs méthodologies ont été proposée qui sont basées sur l'ingénierie de connaissance à savoir : CoMoMAS et MAS-CommonKADS.

- ✓ **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).

B. Les méthodologies issues de l'orienté objet :

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.

Par exemple : ODAC (2003), MaSE (2001), MASSIVE(2001), …

✓ La méthodologie MaSe :

MaSE, à été développée par Deloach et al., signifie Multiagent Software Engineering et est un exemple d'approche complète de développement de systèmes multi-agents de l'analyse au déploiement, avec de nombreux modèles graphiques et une approche logique [Del99]; [Del, Woo00] ; [Del al.01].

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 :**

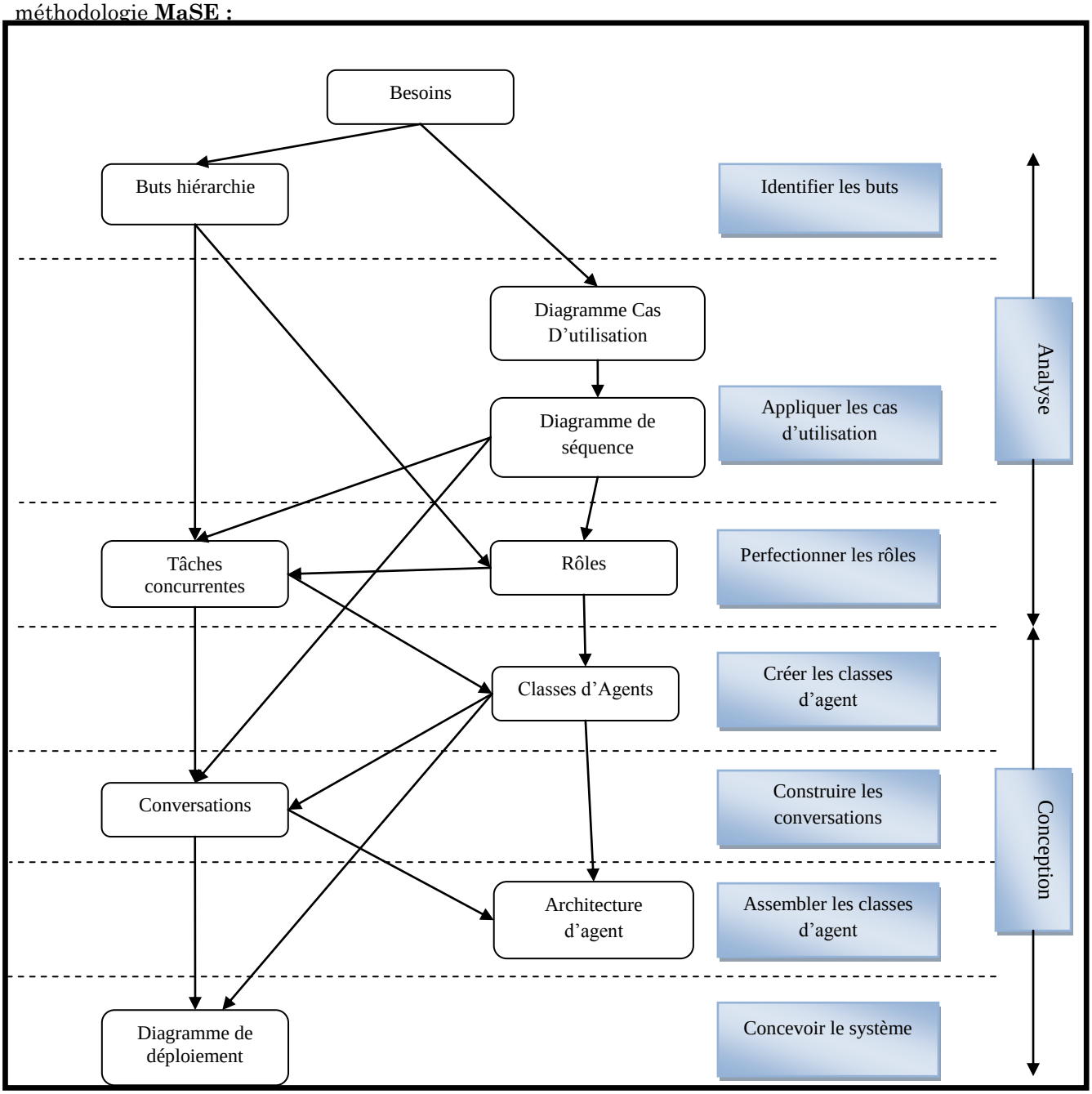


Figure 2.4. Les phases de la méthodologie MaSE [Del99]; [Del, Woo00] ; [Del al.01].

C. Les méthodologies basée agent :

Les méthodologies centrées sur les agents (Hlim [Eal99], Gaia [Woo00], Tropos[Giu01], Prometheus [Pad02] et DACS [Bus04], PASSI[Cos et al, 14]) 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].

✓ 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].

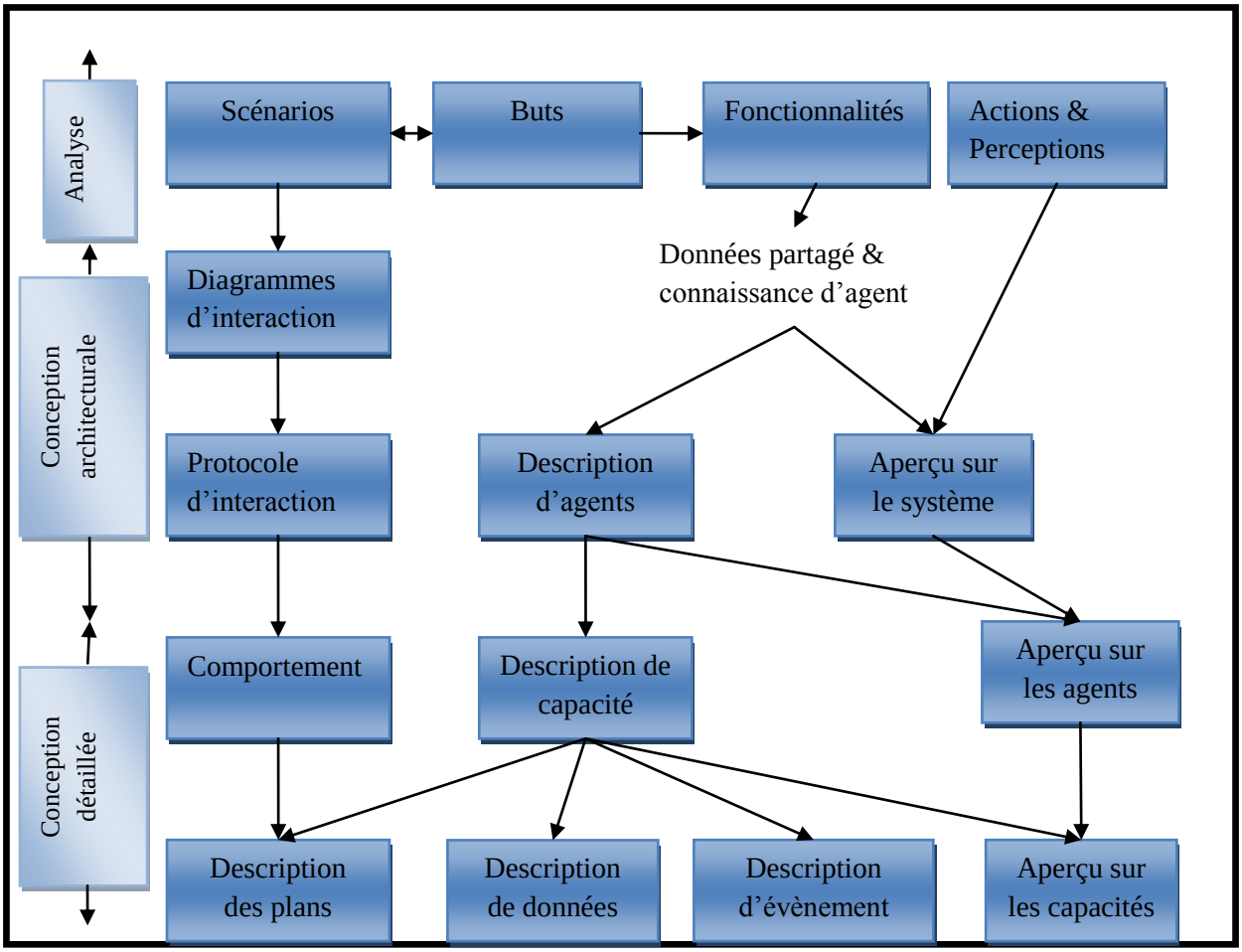


Figure 2.5. Les phases de la méthodologie Prometheus [Pad02]

- **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.

Prometheus possède ses propres outils à savoir: JDE et PDT. JDE (Jack Development Environment) permet d'éditer les modèles proposés durant toys le cycle de développement. Seulement quelques vérifications de consistance sont faites sur certains modèles. PDT (Prometheus Design Tool) manipule les descripteurs et vérifie partiellement les interactions entre les objets qui construits les modèles [Pad, Win02].

IV. La méthodologie PASSI [Cos et al 14]:

A. Définition:

PASSI de Cossentino, qui signifie Process for Agent Societies Specification and Implementation, est une méthodologie pas-à-pas intégrant des modèles de conception et concepts provenant de l'ingénierie orientée objet et de l'intelligence artificielle en utilisant UML [Cos01]; [Cos et Pot02]. Le processus de développement est supporté par l'outil PTK ainsi que rational Rose.

PASSI réutilise fortement UML. Par exemple, le modèle des besoins est exprimé grâce à des diagrammes de cas d'utilisation, de paquetages stéréotypés, de séquences (associés aux cas d'utilisation) et d'activité. Le modèle de société d'agents utilise des diagrammes de classes et de séquences. Les modèles d'implémentation d'agent et de codage utilisent des diagrammes de classes et d'activités classiques. Enfin, le modèle de déploiement utilise des diagrammes de déploiement UML.

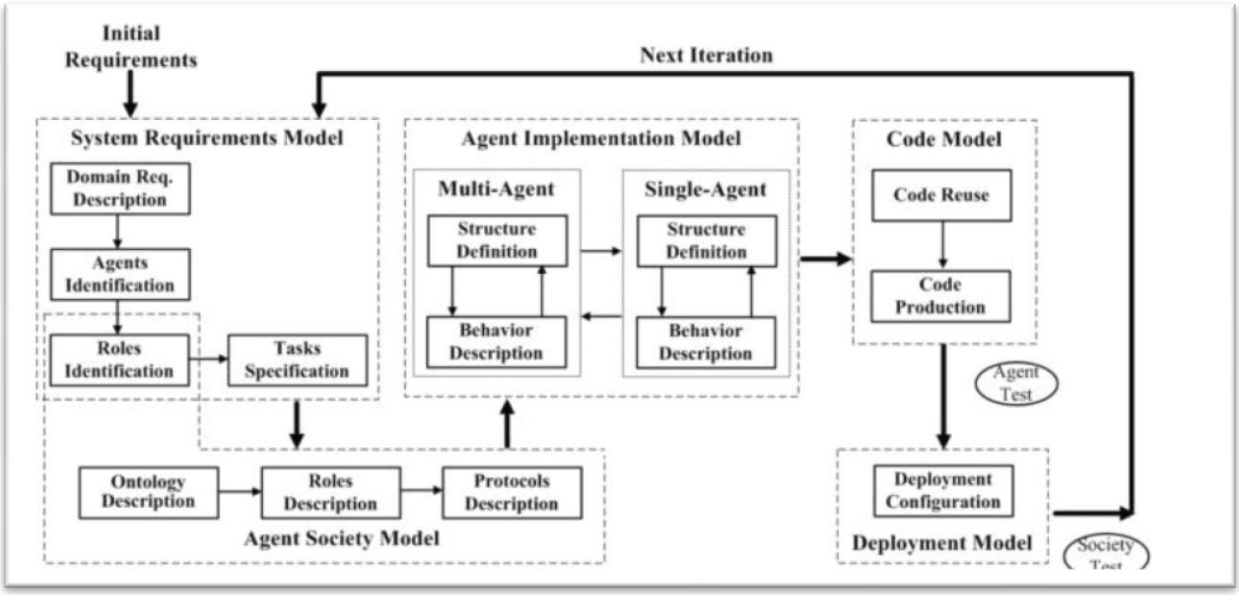


Figure 2.6. Les cinq modèles de la méthodologie PASSI [Cos et al 14]:

Le processus :

Le processus de PASSI est composé de cinq modèles, chaque modèle possède ces propres étapes

i. **Le modèle de besoin :**
   - la description du domaine
   - l'identification d'agents
   - l'identification des rôles
   - la spécification des taches

ii. **Le modèle de société d'agents :**
   - La description de l'ontologie
   - La description de rôles
   - La description des protocoles

iii. **Le modèle d'implémentation d'agents :**
   - La définition de la structure d'agents
   - La définition de comportement des agents

iv. **Le modèle de codage :**
   - la réutilisation de code
   - la complétion de code

v. **Le modèle de déploiement :**
   - La configuration de déploiement

B. Le modèle de besoins :

Le modèle de besoins représente la première étape du processus de développement chez la méthodologie PASSI. Il à comme un objectif de réalisé un système qui répond au attente des utilisateurs. Plusieurs diagrammes ont été proposés à ce niveau.

- **La description de domaine :**

  Dans cette étape on utilise le diagramme de cas d'utilisation d'UML (cas d'utilisation étendu), on va présenter toutes les besoins du système plus la description du contexte.

- **Identification d'agents :**

  Cette étape commence à partir de diagramme de cas d'utilisation utilisé dans l'étape précédente.

  Un agent peut assurer plusieurs rôles fonctionnels lors d'interaction avec d'autres agents pour atteindre ses objectifs. Donc on peut dire que l'identification des agents peut être un cas d'utilisation ou un ensemble (sur le cadre de la décomposition fonctionnel), on basant sur un diagramme bien détaillé de fonctionnalités de système à travers le regroupement d'un ou plusieurs cas d'utilisations dans des paquetages stéréotypés pour l'objectif de fournir un nouveau diagramme.

Chaque paquetage définit les fonctionnalités d'un agent spécifique. On remarque que les relations entre les cas d'utilisations du même agent sont conformes à la syntaxe UML et stéréotypes habituel (include, extends) par contre le stéréotype communication est utilisé entre agents. Donc on peut dire que ce diagramme est d'orienté les relations de communications entre agents de l'initiateur vers le participant.

- **Identification des rôles :**

  Un rôle c'est un ensemble des taches effectuées par l'agent pour atteindre un objectif, cette étape est basée sur l'exploration de tous les chemins de communication inter-agent dans le diagramme d'identification d'agents.

  - Chaque chemin décrit un scénario dans lequel des agents interagissent pour obtenir le comportement de système, un chemin de communication c'est une relation « communication » (diagramme de séquence).
  - Chaque objet du diagramme représente un rôle et nous le nommons avec une syntaxe **<nom-rôle> : <nom-agent>**
  - Un agent peut participer dans différentes scénarios en jouant des rôles différents, comme il peut jouer un rôle dans le même scénario.
  - Les messages de diagrammes de séquences signifient :
    - ✓ Evénement générer par l'environnement externe
    - ✓ Communication entre les rôles d'un ou plusieurs agents
    - ✓ Les messages spécifient le rôle et d'autre donnée à fournir ou recevoir, la spécification avec détaille est présenté dans l'étape de description de l'ontologie

- **La spécification des taches :**

  La tache est définie comme une unité intentionnelle de comportement individuel ou interactif. Donc le plus important c'est le comportement de chaque agent afin de la décomposer en tache. L'unité de travail logique c'est une encapsulation de certaine fonctionnalité à partir des taches. Chaque agent a un diagramme d'activité composé de deux couloirs :

  - **Couloir droit :** contient un ensemble d'activités qui représentent les taches d'agent
  - **Couloir gauche :** contient un ensemble d'activités représentant les autres agents en interaction
  - Donc on va attendre les informations sur les rôles
  - Les relations entre les activités signifient :
    - ✓ Message entre des taches de même agent
    - ✓ Message entre des taches des autres agents (interaction)

V. 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. Dans le chapitre suivant on va ouvrir une porte sur une comparaison de ces méthodologies à fin qu'on puisse choisir la méthodologie idéal sur la quel qu'on doit se basé pour la formalisation des besoins fonctionnels.

Chapitre 3 :

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

Alain Rémond :

« *L'informatique est géniale : les e-mails, même si vous n'y répondez pas, ça ne prend pas de place. »*

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 des 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.

- 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.

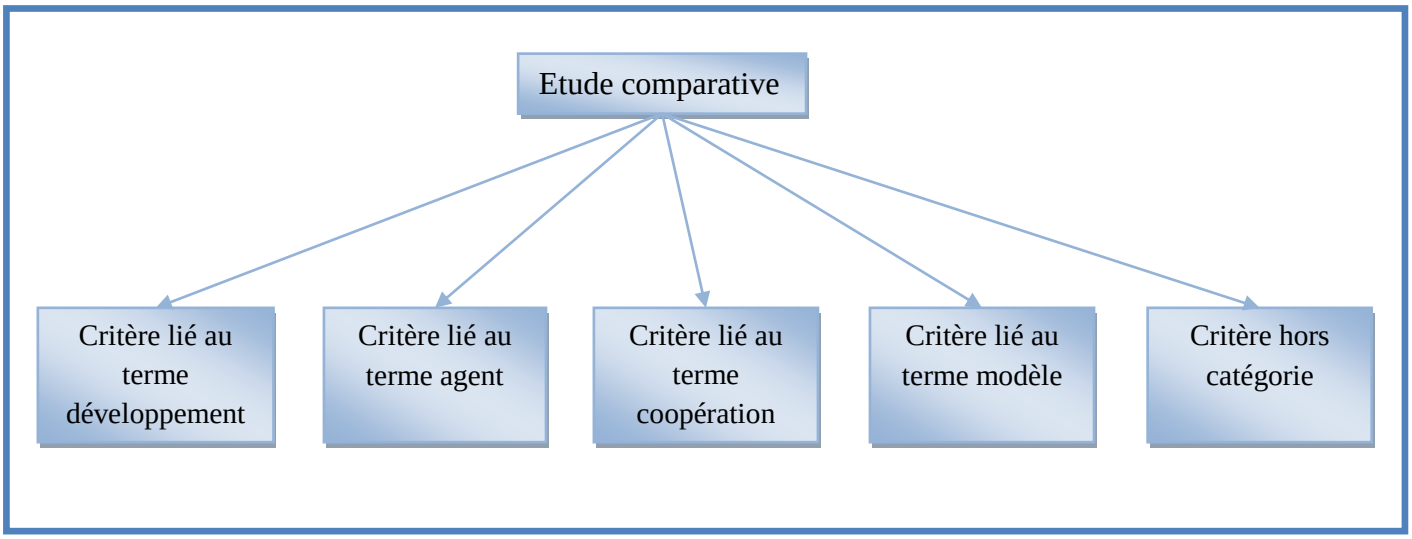


Figure 3.1. Structure de critères de comparaison

A. 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]

B. 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.

C. 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.

D. 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].

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].

E. 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 quatre 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].

A. Catégorie développement :

La comparaison des quatre méthodologies tout en se basant sur les critères relatifs a la catégorie développement est illustrée par la table 3 .1.

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	PASSI
Etape de processus	Analyse	Oui	oui	oui	Oui
	Modélisation	Oui	oui	oui	Oui
	Spécification	Oui	oui	oui	Oui
	Conception	Oui	oui	oui	Oui
	Validation		non		Oui
	Vérification	Oui	possible		Oui
Modèles De développement	Cascade	Non	non		
	Incrémentale	Non	oui		
	Spirale	Oui			
	V	Non	non		
Approche De développement	Descendante	Oui	oui		
	ascendante	Oui	oui	oui	
	Evolutive	Non	oui		Oui
Disponibilité	Analyse	OOSE, MSC,	AgML	OOSE	Rational

Des Supports Logciels		RDD, SDL, HMSC, CML, OMT, CommonKADS			Rose, PTK
	Modélisation	OOSE, MSC, RDD, SDL, HMSC, CML, OMT, CommonKADS	AgML	OOSE	Rational Rose, PTK
	Spécification	OOSE, MSC, RDD, SDL, HMSC, CML, OMT, CommonKADS	AgML	OOSE	Rational Rose, PTK
	Conception		AgML, AgDL, UML	OOSE, AUML	AUML Rational Rose PTK
	Validation				
	Vérification		AgDL		

Table 3.1. Comparaison selon les critères liés au terme processus de développement

B. Catégorie agent :

La table 3.2 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	PASSI
Nature d'agent	Homogène	Non	non	non	non
	Hétérogène	Oui	p	p	p
Type d'agent	agents intelligents (ou BDI)	Oui	p	oui	oui
	agents interfaces ou personnels	Oui	p		**oui**
	Agent Mobile	Oui	p		Oui
	Agent d'information	Oui	oui		oui
Caractéristique d'agent	Adaptabilité	P	p	p	oui
	Autonomie	Oui	p	p	oui
	comportement coopératif	Oui	p	oui	oui
	capacité déductive	Oui	p	oui	oui
	habileté de	Oui	p	oui	oui

	communication				
	Mobilité				
	Réactivité	Oui	p	oui	oui
	Personnalité	Oui	p		

Table 3.2. Comparaison selon les critères liés au terme agent

C. Catégorie coopération :

La table 3.3 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	PASSI
Modes de Communication	Direct	Oui	oui	p	p
	Indirect		non	p	p
	Synchrone	Oui	p	p	P
	Asynchrone	Oui	p	p	p
Langage de communication	Signes	Oui	p		
	actes de discours	Oui	p	oui	oui
	Autres	Non	non	non	non
Type de contrôle	Centralisé	P			
	hiérarchique	Oui			
	Distribué	Oui	p	p	oui

Table 3.3. Comparaison selon les critères liés au terme coopération

D. Catégorie modèle :

Chaque méthodologie utilise des modèles pour la modélisation des SMA et chaque modèle a 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. Le tableau 3.4 représente cette évaluation.

	Méthodologies qui utilisent les cas d'utilisation			
Méthodologies / **Critères**	MAS-CommonKADS	MaSE	Prometheus	PASSI
Concurrence	Non	Oui	Non	Oui

Réutilisabilité	P	Oui		oui
Nombre de modèle	7	4		5
Cohérence de modèle	P	P	P	oui
Complétude des modèles	P	Non	P	oui
Complexité de modèle	15	6		

Table 3.4. Comparaison selon les critères liés au terme modèle

E. Critères hors Catégorie :

La table 3.5 montre la comparaison entre les quatre 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.

OA : signifie que la méthodologie utilise l'approche Orienté Agent.

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	PASSI
taxonomies de spécification	S-Form	S-Form	S-Form	S-Form

Système ouvert	Non	Non	Non	
Supporter la notion ontologie	Oui	Non	Non	oui
Approches utilisées	IC & OO	OO	OO	OO & OA
Taille du SMA	NS	<= 10 classes d'agent	N'importe quelle taille	N'importe quelle taille
Domaine d'application	Indé	Indé	Indé	Indé
Classification des méthodologies	IC	OO	BA	BA

Table 3.5. 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 quatre 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).

À travers les résultats obtenus à partir des tableaux comparatifs, on peut conclure que la méthodologie PASSI est la meilleure pour notre travail, où elle couvre toutes les étapes de processus de développement plus qu'elle est basée agent et évolutif.

Chapitre 4 :

Génération d'une spécification formelle: de PASSI vers Maude.

Alain Rémond :

« *L'informatique est géniale : les e-mails, même si vous n'y répondez pas, ça ne prend pas de place. »*

I. Introduction :

Le génie logiciel orienté agent est, actuellement, un domaine de recherche très actif. Dans cette dernière décennie plusieurs méthodologies de développement des systèmes multi-agents (GAIA[Woo00], Prometheus [Pad02], etc) ont été émergées dans la littérature dans le but de faciliter le développement des applications multi-agents. Ces méthodologies ont certes apportés des réponses importantes au processus de développement des SMA, cependant, l'aspect méthodologique n'est pas encore maîtrisé. En effet, aucune de ces méthodologies ne tiennent compte de la formalisation des besoins fonctionnels du futur système. 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 son coût [Dan07]. Par conséquent, l'utilisation de notations formelles permet de produire des descriptions rigoureuses et précises et de supporter ainsi leur processus de vérification et de validation [Tai03].

Nous présentons dans ce chapitre une approche permettant de translater les besoins fonctionnels d'un SMA décrits à l'aide d'un ensemble de diagrammes de la méthodologie PASSI 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 et 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 quatre diagrammes de la méthodologie PASSI à savoir : modèle de description de domaine, modèle d'identification d'agents, modèle d'identification de rôles, modèle d'identification de taches. Dans la deuxième étape, les diagrammes considérés doivent subir une validation inter-modèle afin de vérifier la consistance du système. La troisième étape consiste à la génération d'une description Maude à partir des diagrammes utilisés.

II. Les taxonomies des approches de spécification :

On distingue trois classes d'approches à savoir :

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.

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 technique comme semi-formelle [Mar96].

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, les approches formelles ont plusieurs avantages, elles permettent :

- De prouver que les programmes sont corrects puisque ils sont basés sur une logique mathématique.
- La validation des composants de la spécification.
- La vérification du passage d'une spécification à une autre.
- Une compréhension approfondie du système.

III. Les diagrammes utilisés dans la spécification des besoins de la méthodologie PASSI [Cos et al 14] :

Dans la méthodologie PASSI on trouve quatre diagrammes qui représentent la phase de spécification des besoins, chaque diagramme a son spécificité.

1) Le diagramme de description du domaine:

Ce diagramme permet d'exprimer les besoins fonctionnels des utilisateurs et de décrire le contexte du système grâce à une hiérarchie de cas d'utilisation [Mas12](diagramme de cas d'utilisation étendu).

- **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. L'inclusion nous permet de décomposer un cas d'utilisation complexe en sous cas plus simple. Cette relation est symbolisée par le stéréotype «include» [Lau07, OMG07].
- **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. Cette relation est symbolisée par le stéréotype «extend».
- **La généralisation :** 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].
- **La communication :** c'est une relation entre deux cas d'utilisations de différents agents ou un agent et un cas d'utilisation. Dans le premier cas il présente une importation partielle d'un cas d'utilisation vers un autre (pas d'importation global), et dans le deuxième cas se sont des actes de communication échangés entre l'agent interface et l'agent demandeur du service [Cos et al 14].

Exemple : le diagramme suivant montre une description de domaine pour une librairie

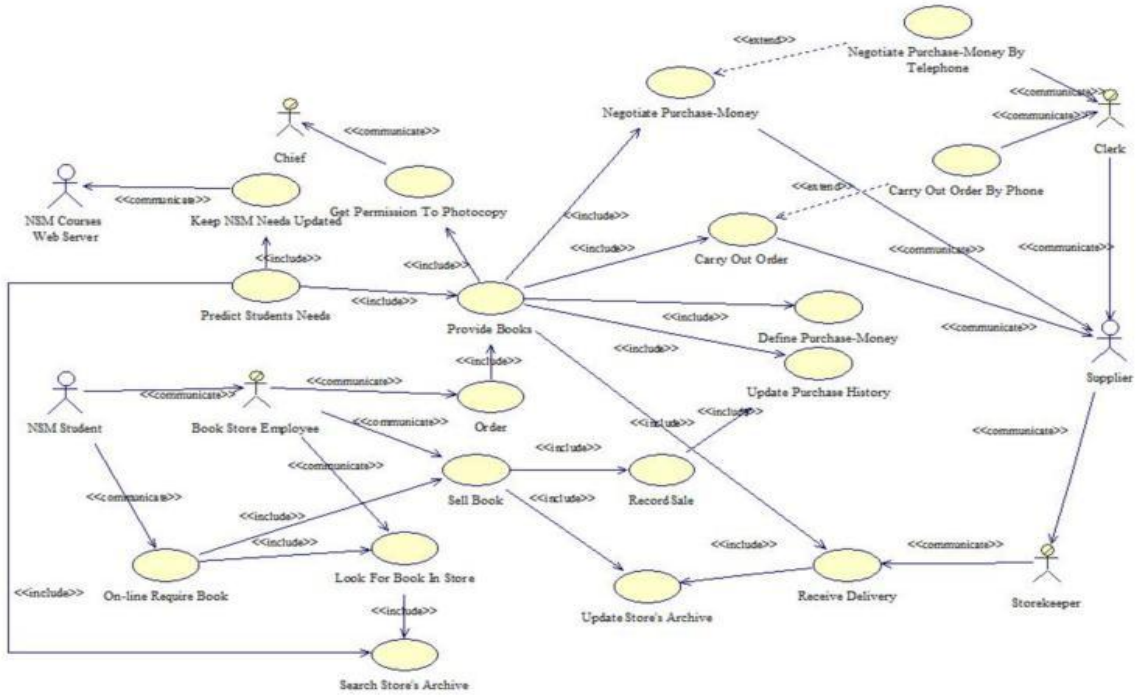


Figure4.1. le diagramme de description de domaine d'une librairie [Mas12]

2) Le diagramme d'identification des agents :

Dans cette étape on va regrouper les cas d'utilisations en paquetages, chaque paquetage représente un ensemble de services assuré par un agent particulier. [Mas12]

Exemple : le diagramme suivant montre l'identification des agents pour une librairie

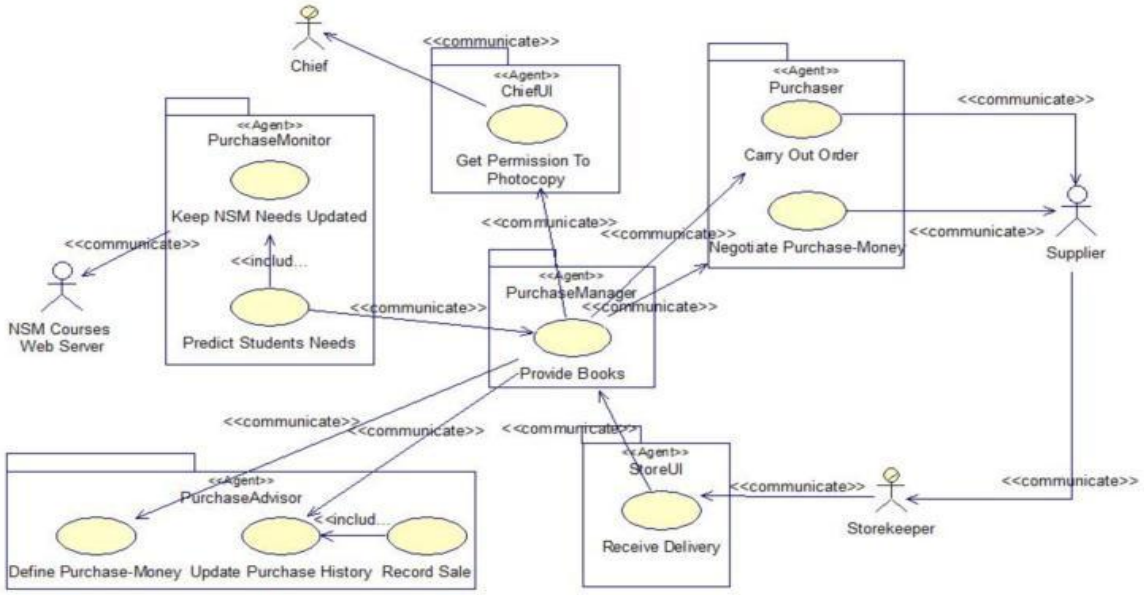


Figure4.2. diagramme d'identification d'agents d'une librairie [Mas12]

3) Le diagramme d'identification des rôles :

C'est précisément le rôle du diagramme de séquence AUML. Les rôles identifiés jouent le rôle d'une description fonctionnelle et comportementale et une représentation des relations entre agents, cette étape est commune entre le module de spécification des besoins et de la définition de la société d'agents [Mas12].

Exemple : le diagramme suivant montre l'identification des rôles pour le scénario d'achats des livres annoncé par l'agent moniteur d'achat (PurchaseMonitor).

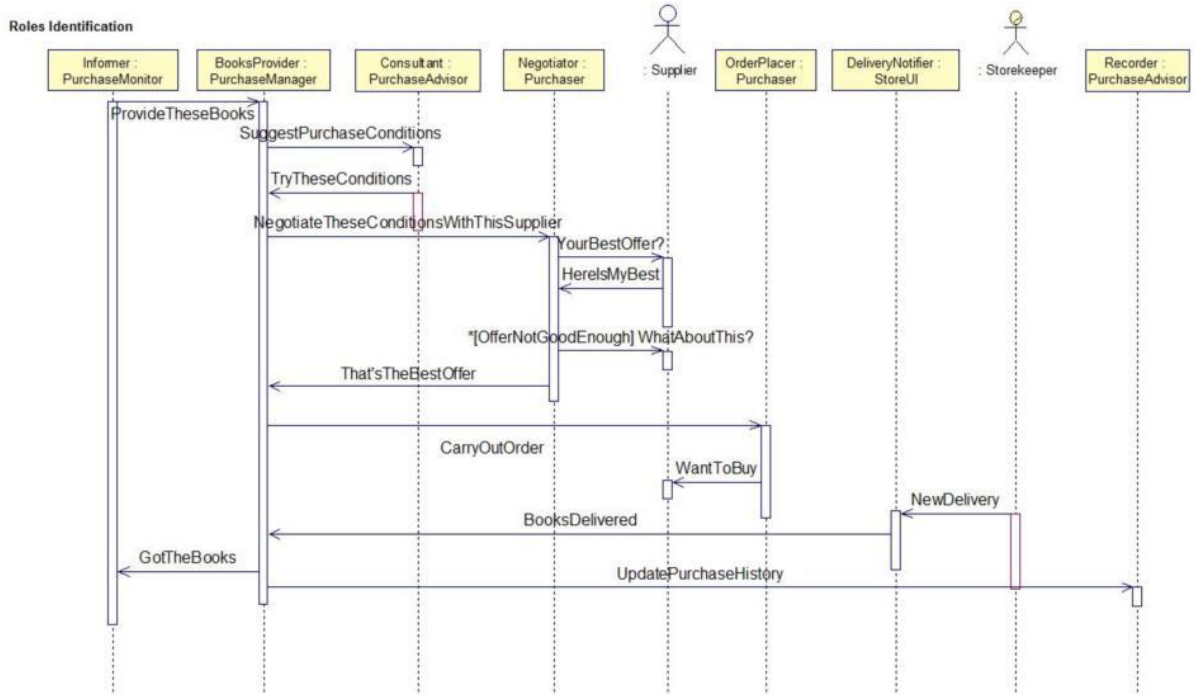


Figure4.3. diagramme d'identification des rôles dans l'achat des livres annoncés par l'agent PurchaseMonitor [Mas12]

4. Le diagramme d'identification des taches :

Ce diagramme permet de définir les activités des agents en reliant aux fonctionnalités fournies par le système [Mas12].

Ex : le diagramme suivant montre l'identification des taches pour le moniteur des achats dans la librairie

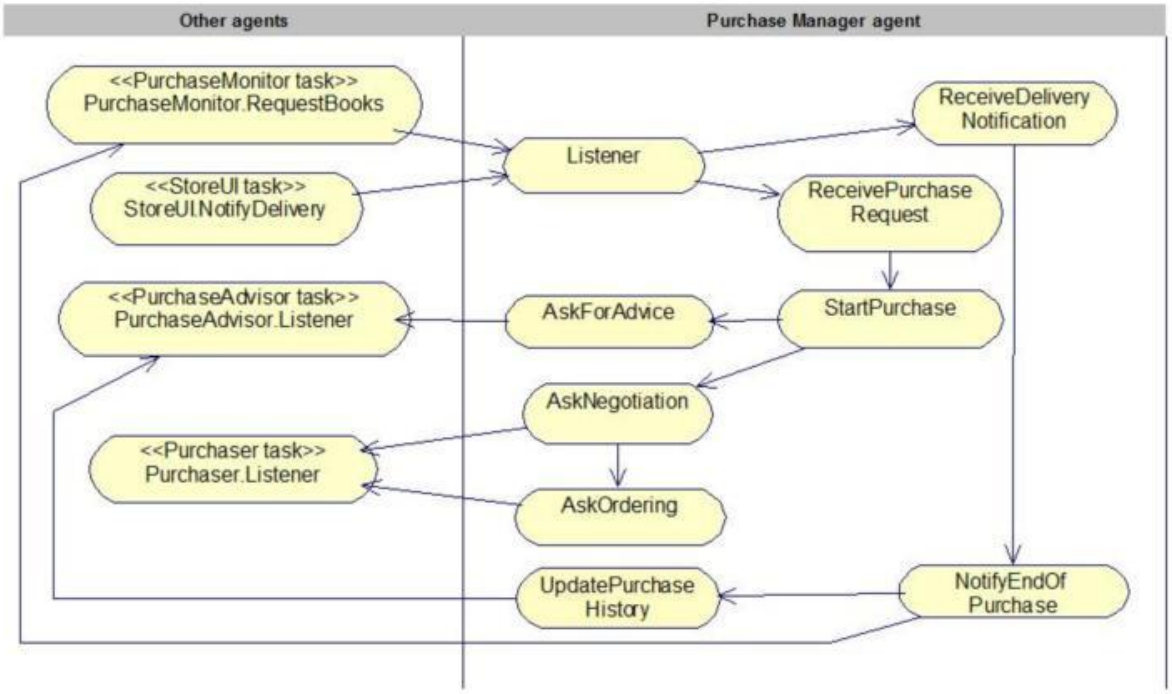


Figure4.4. diagramme d'identification des taches d'une librairie [Mas12]

IV. Le 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 4). Figure 4(a) indique que tous les actes de communication CA-i (*CA-1,...CA-n)* sont envoyés en concurrence. Figure 4(b) présente le *'ou inclusif'* qui inclut une boîte de décision qui permet aux actes de communication CAs (zéro or plusieurs) d'être envoyés. Figure 4(c) présente le *'ou exclusif'* qui indique qu'un et un seul CA doit être envoyé.

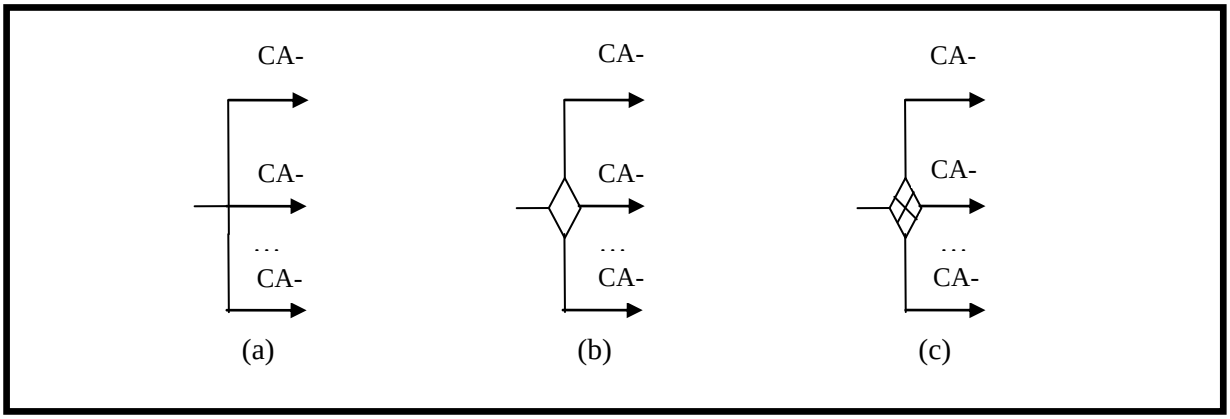


Figure 4.5. Extensions recommandées supportant les threads concurrents d'interaction

V. L'approche proposée :

a. Passage vers un diagramme de séquence AUML :

❖ **Un seul cas d'utilisation [Ham et al 10]**

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

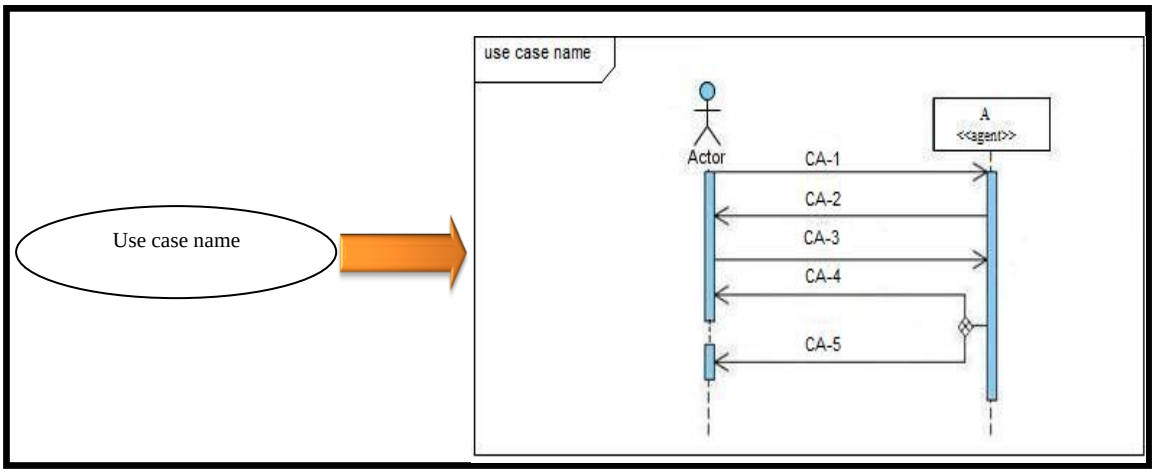


Figure4.6. passage d'un cas d'utilisation vers AUML

❖ **La relation inclusion stéréotypée « include » [Ham et al 10]**

La présentation de la relation « include » avec l'AUML :

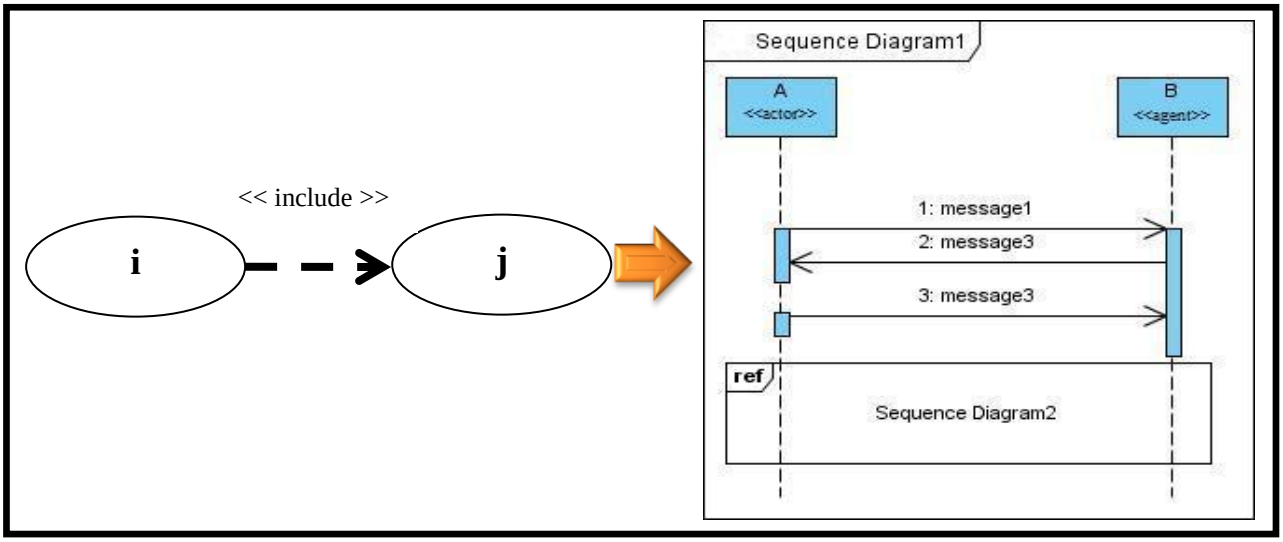


Figure4.7. passage d'une relation « include » vers AUML

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

❖ **La relation extension stéréotypée « extend » [Ham et al 10] :**

La présentation de la relation « extend» avec l'AUML :

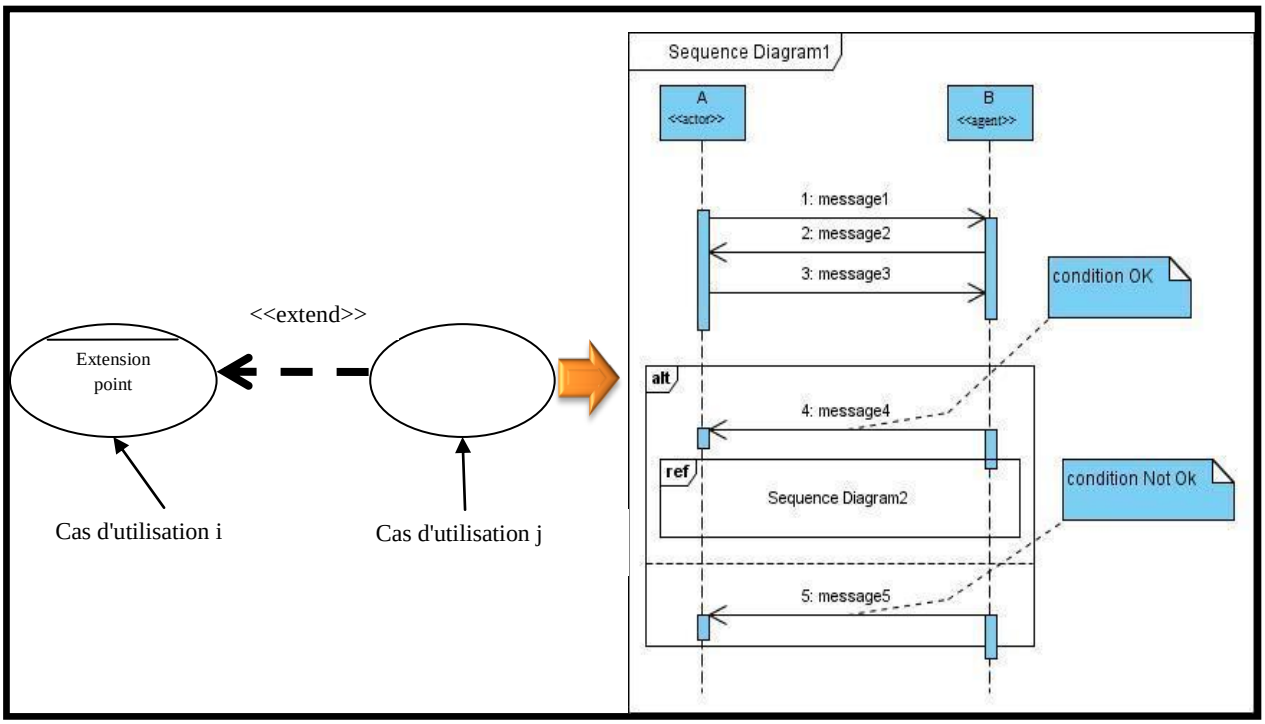


Figure4.8. passage d'une relation « extend » vers AUML

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

❖ **La relation de generalization [Ham et al 10]**

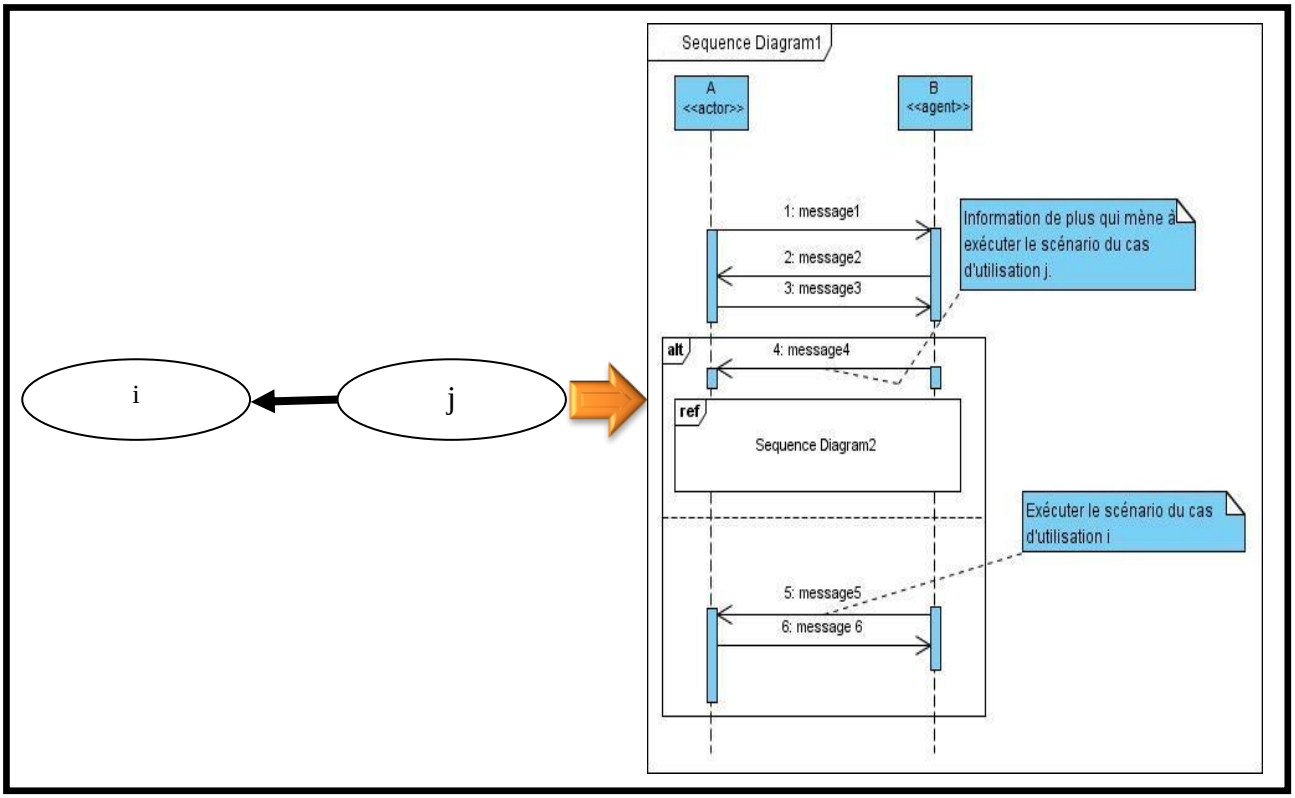


Figure4.9. passage d'une généralisation vers AUML

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

- **La relation de communication:**

La présentation de la relation stéréotypée << communicate>> avec l'AUML :

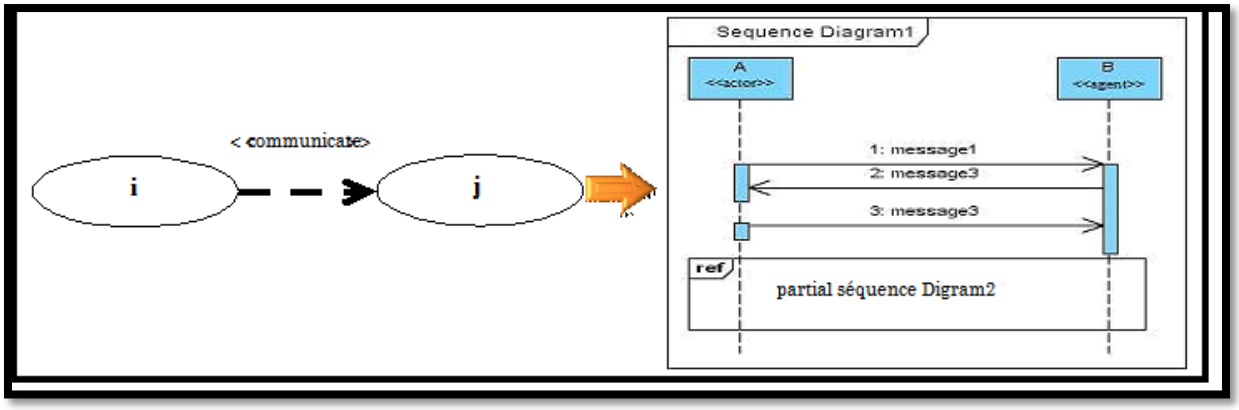


Figure4.10. passage d'une communication vers AUML

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

VI. Le processus de translation :

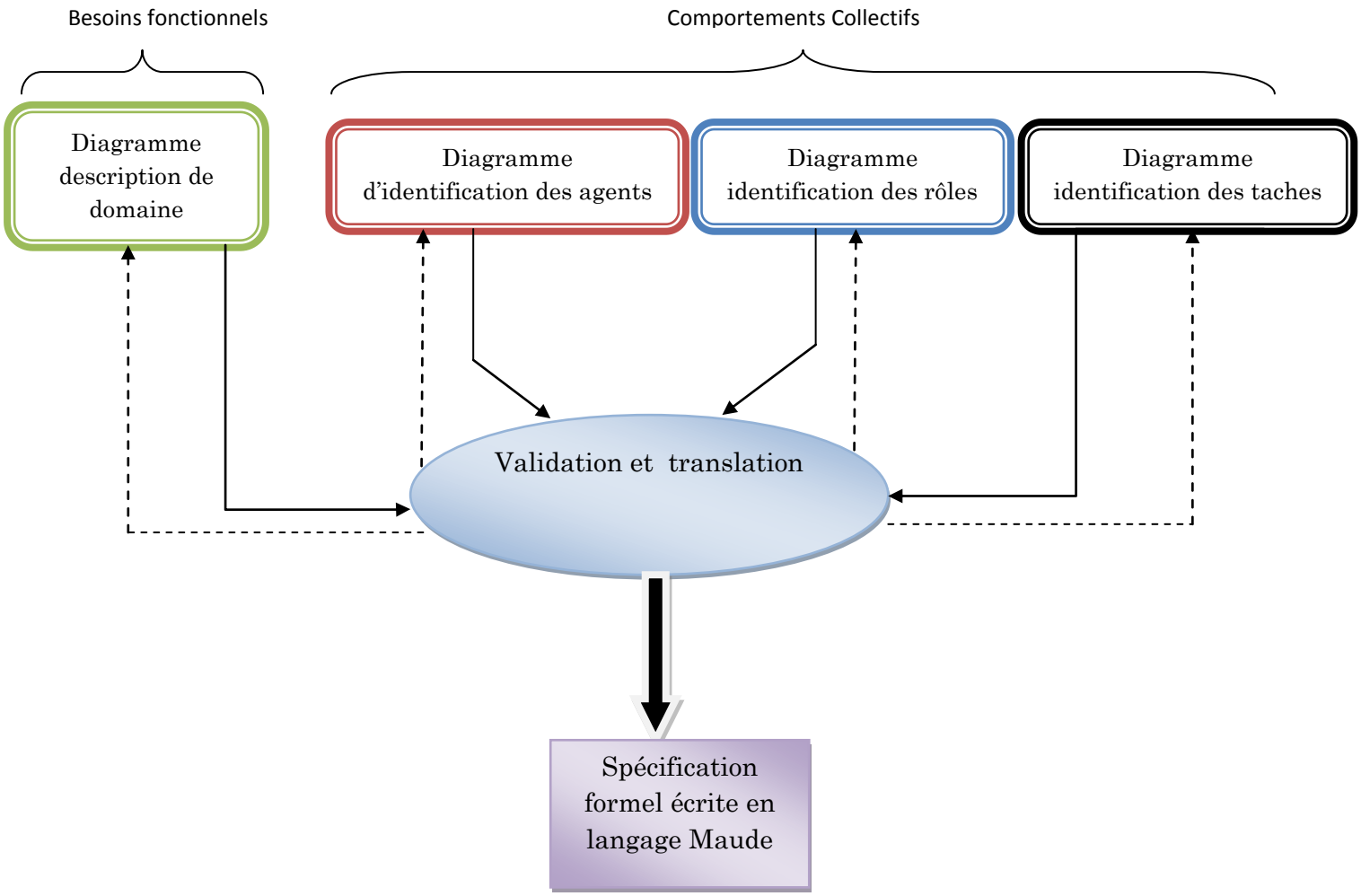


Figure 4.11. Démarche générale de l'approche

La figure 4.11 illustre le processus de formalisation de besoins fonctionnels des SMAs à partir des différents diagrammes présentés dans la première phase d'analyse de besoins de la méthodologie PASSI. Partant d'une spécification semi-formelle de besoins fonctionnels décrite à l'aide de diagramme de cas d'utilisation et de diagrammes de séquences AUML ainsi que le diagramme de rôles et taches. 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.

VII. Le 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 illustré sns la méthodologie PASSI à l'aide du langage Maude. Comme il est indiqué dans la figure 4.12 Notre framework est composé de plusieurs modules modules fonctionnels et systèmes.

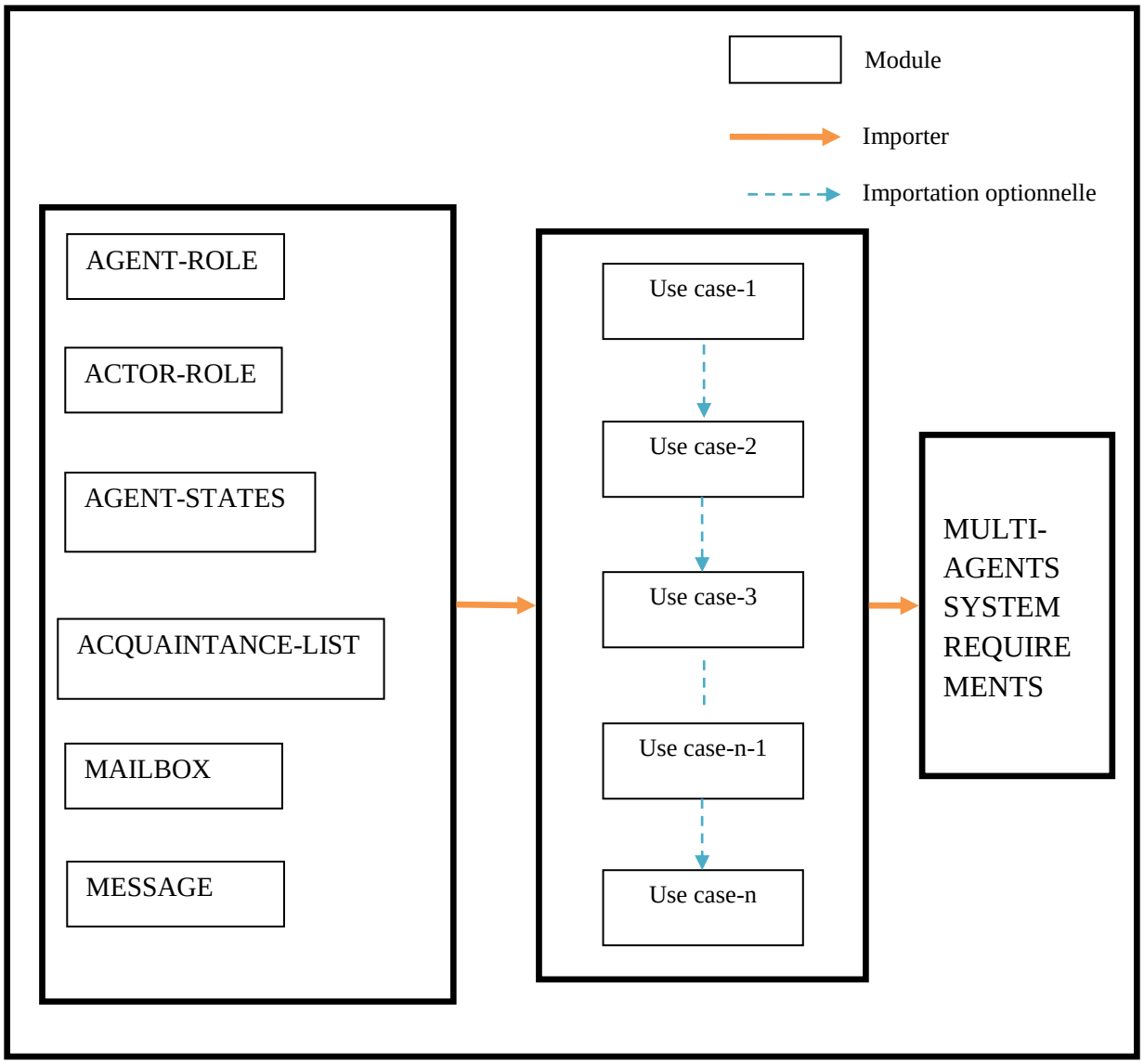


Figure 4.12.L'architecture du framework proposé

✍ Le module fonctionnel AGENT-ROLE d'écrit un type nommé *TypeRoleAg* qu'on peut l'utilisé pour définir un ensemble de valeur de rôle d'agent qui prennent ce type.

```
fmod AGENT-ROLE is

sort TypeRoleAg .

endfm
```

Figure 4.13. Le Module Fonctionnel AGENT-ROLE

- La figure 4.14 montre un module fonctionnel qui d'écrit un type nommé *TypeRoleAc* qu'ont peut l'utilisé par la suite pour définir un ensemble de valeur de rôle d'acteur qui prennent ce type.

```
fmod ACTOR-ROLE is
sort TypeRoleAc .
endfm
```

Figure 4.14. Le Module Fonctionnel ACTOR-ROLE

- Le module AGENTS-STATES et un module fonctionnel qui d'écrit un type nommé *AgentState.* Ce dernier doit être prennent par un ensemble de valeurs qui d'écrit les états des agents.

```
fmod AGENTS-STATES is
sort AgentState .
endfm
```

Figure 4.15. Le Module Fonctionnel AGENT-STATES

- La figure suivante d'écrit 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 4.16. Le Module Fonctionnel ACQUAINTANCE-LIST

- MAILBOX est un module fonctionnel qui permet de d'écrit l'état du courrier de l'agent : vide ou non. Ce module est formalisé comme suit :

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

Figure 4.17. Le Module Fonctionnel MAILBOX

- Le module système MESSAGE est un module qui permet de définir tous les messages échangé entre les entités du système.

```
mod MESSAGE is
including CONFIGURATION .
protecting STRING .
sort content .
op Message : Oid Oid content -> Msg .
endm
```

Figure 4.18. Le Module Système MESSAGE

- A chaque cas d'utilisation est associé un module système *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*, ils sont importés dans le module système *MUTI-AGENTS-SYSTEM-REQUIEREMENTS* 4.18 représentant le module principal.

```
mod MAS-REQUIREMENTS is
protecting Use-Case1 .
protecting  Use-Case2 .
```

```
…
protecting  Use-Casem .
endm
```

Figure 4.19. Le Module Principale MAS-Requirements

VIII. La génération d'une spécification Maude :

- **Un seul cas d'utilisation [Ham et al 10] :**

La translation d'un cas d'utilisation isolé (n'est pas en relation avec d'autres cas d'utilisation) en langage Maude est l'implémentation des différents scénarios présentés par les diagrammes de séquences AUML correspondant à l'aide de règles de réécriture sous forme d'un module système.

```
mod USE-CASE is
rl [1] : <Configuration> =>  <Configuration> .
…
rl [m] : <Configuration> => <Configuration> .
endm
```

Figure4.20. génération d'un seul cas en Maude

- **La relation d'inclusion stéréotypée « include » [Ham et al 10] :**

Comme nous l'avons mentionné, l'aspect sémantique de la relation d'inclusion qui est assimiler à l'inclusion d'un scénario dans un autre, ce concept sera appliquer au niveau de la spécification Maude donc on aura une spécification formelles qui importe une autre spécification le résultat sera comme suit :

```
mod A-INCLUDING is
including B-INCLUDED. *** appel du comportement du cas d'utilisation B.
rl [1] : <Configuration> =>  <Configuration> .
…
rl [m] : <Configuration> => <Configuration> .
endm
```

```
mod B-INCLUDED is
rl[1] : <Configuration> => <Configuration> .
…***le scénario du cas d'utilisation included.
rl [2] : <Configuration> => <Configuration> .
endm
```

Figure4.21. génération d'une relation d'include en Maude

- **La relation d'extension stéréotypée « extend » [Ham et al 10] :**

L'exécution de la relation du scénario du cas d'utilisation ***extended*** est liée à la satisfaction de la condition (point d'extension) (cas d'utilisation extending peut être appelé au cours de l'exécution du cas extended). Donc la spécification formelle qui représente le scénario du cas d'utilisation extended sera exécutée si la ***variable*** dans condition est vraie. Donc l'exécution du scénario du cas d'utilisation ***extending*** est liée à la satisfaction de la variable inclus dans le point d'extension. Dans ce cas en peut utiliser une règle de réécriture conditionnelle qui représente le point d'extension au niveau du diagramme du cas d'utilisation. La structure générale de la spécification Maude générer dans ce cas se faite comme suit :

mod EXTENDED-USE-CASE is
including EXTENDING-USE-CASE .
… début du scénario du cas d'utilisation extended.
crl[1] :<Configuration> => <Configuration>
if (condition in extension point is true)
… *** execution du scénario du EXTENDING-USE-CASE
crl[2] :<Configuration> => <Configuration>
if (condition in extension point is false)
…*** le scénario du EXTENDING-USE-CASE ne sera pas exécuter.
endm
	mod EXTENDING-USE-CASE is
	rl[Begin] :<Configuration> => <Configuration>.
	…***le scénario du cas d'utilisation Extended.
	rl[end] : <Configuration> => <Configuration> .
	endm

Figure4.22. génération d'une relation d'extend en Maude

- **La relation generalization [Ham et al 10] :**

Ce type de relation sera translaté avec MAUDE par l'introduction de quelques informations qui nous permet de donner la spécialisation d'un cas d'utilisation.

mod GENERAL-USE-CASE is including SPECIAL-USE-CASE .	mod SPECIAL-USE-CASE is rl[1] :<Configuration> => <Configuration>.

crl[1] :<Configuration> => <Configuration>
if (attribute = value)
*** value est une variable qui déclenche l'exécution du SPECIAL-USE-CASE
***l'appel du module SPECIAL-USE-CASE
crl[2] :<Configuration> => <Configuration>
if (attribute =/= value)
*** continuer l'exécution du GENERAL-USE-CASE
Endm
	…***le scénario du cas d'utilisation SPECIAL-USE-CASE.
	rl[m] : <Configuration> => <Configuration> .
	endm

Figure4.23. génération d'une relation de généralisation en Maude

- **La relation de communication**

Comme nous l'avons mentionné, l'aspect sémantique de la relation de communication qui est assimilé à une image partielle de l'inclusion. C-a-d qu'on peut pas terminer le scénario d'un cas d'utilisation sans appeler une partie d'un scénario dans un autre cas d'utilisation.

mod A-INCLUDING is including Partial-B-INCLUDED. *** appel du comportement du cas d'utilisation B. rl [1] : <Configuration> => <Configuration> . … rl [m] : <Configuration> => <Configuration> . endm	mod Partial-B-INCLUDED is rl[1] : <Configuration> => <Configuration> . …***le scénario du cas d'utilisation included. rl [2] : <Configuration> => <Configuration> . endm

Figure4.24. génération d'une relation de communication en Maude

IX. Étude de cas : « système bancaire ATM » :

A. La spécification de domaine :

Dans cette partie, on va appliquer notre approche à un exemple concret. Le système bancaire ATM (Automatic Teller Machine) c'est un bon exemple de validation de notre approche car elle contient toutes les relations identifiées entre les cas d'utilisation et il est clair et facile à comprendre.

Les exigences fonctionnelles du système bancaire sont décrites par le diagramme de cas d'utilisation étendu de la figure 4.25.

Ce diagramme est étendu en utilisant des acteurs internes représentant des agents du système. Nous identifions trois acteurs internes (agents) jouant le rôle de médiateur c'est un agent interface qui est responsable de toute les affichages visible par le système, agent sécurité c'est l'agent responsable du coté sécurité relativement à l'authentification ou à la validité des cartes clients et finalement c'est l'agent gestionnaire des transactions est ce lui qui assure la responsabilité validation, le rejet ou l'annulation des transactions, l'enregistrement, l'impression, la vérification des soldes. Le client est un acteur externe qui interagit avec le système bancaire ATM. Chaque cas d'utilisation d'agent décrit une fonctionnalité fournie par le système.

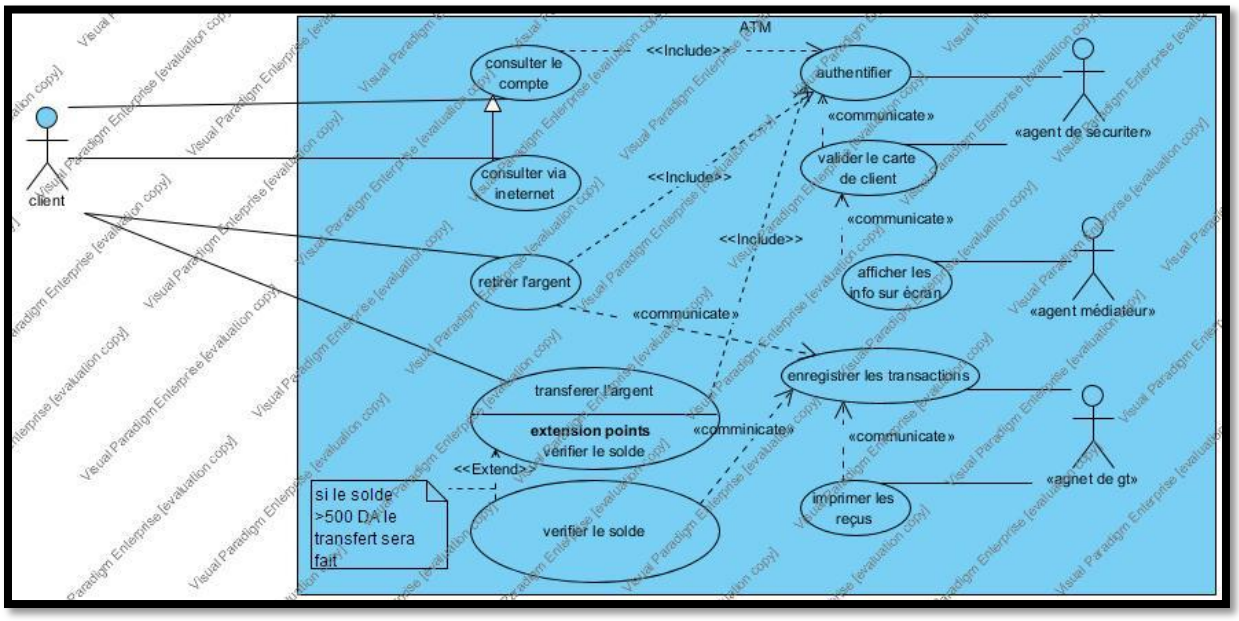


Figure4.25. Le diagramme de description de domaine des ATM

Après avoir présenté le diagramme de description de domaine on va passer à la deuxième étape c'est le diagramme d'identification des agents. Dans cette étape on va regrouper les cas d'utilisation dans un paquetage relative à un agent bien determiné. Les relations entre les cas d'utilisation d'un même agent est présenter par les stéréotypes habituel « extend » et « include » par contre les relations entre les cas d'utilisations de différentes agents sont présenter par le stéréotype « communicate».

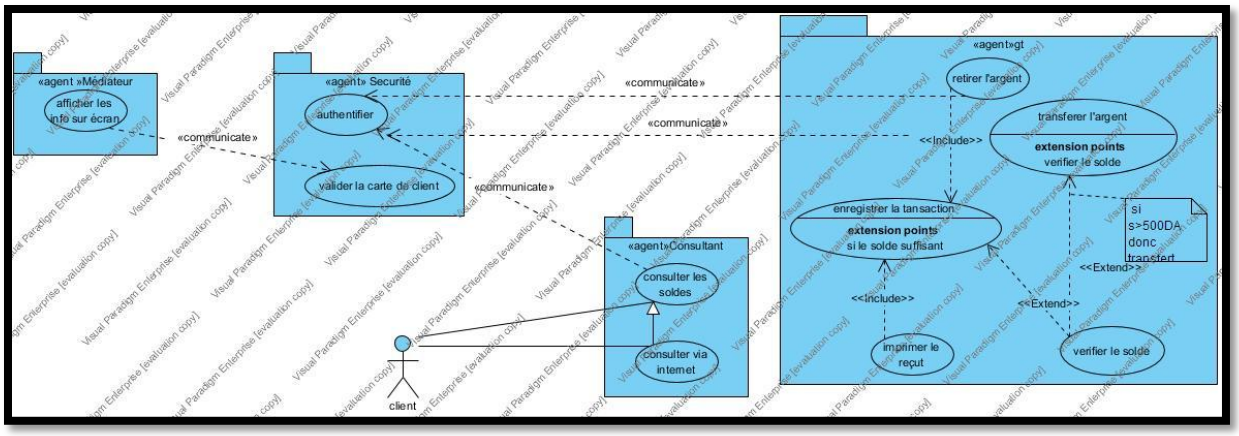


Figure4.26. Le diagramme d'identification des agents des ATM

La troisième étape représente l'identification des rôles, elle est basée sur l'exploration de touts chemins de communications inter-agent dans le diagramme d'identification d'agents. Chaque chemin décrit un scénario dans lequel des agents interagissent pour satisfaire un but du système. Pour ce la on va voir quelque exemple de diagramme d'identification des rôles de ce système.

- **Diagramme d'identification de rôle d'authentification :**

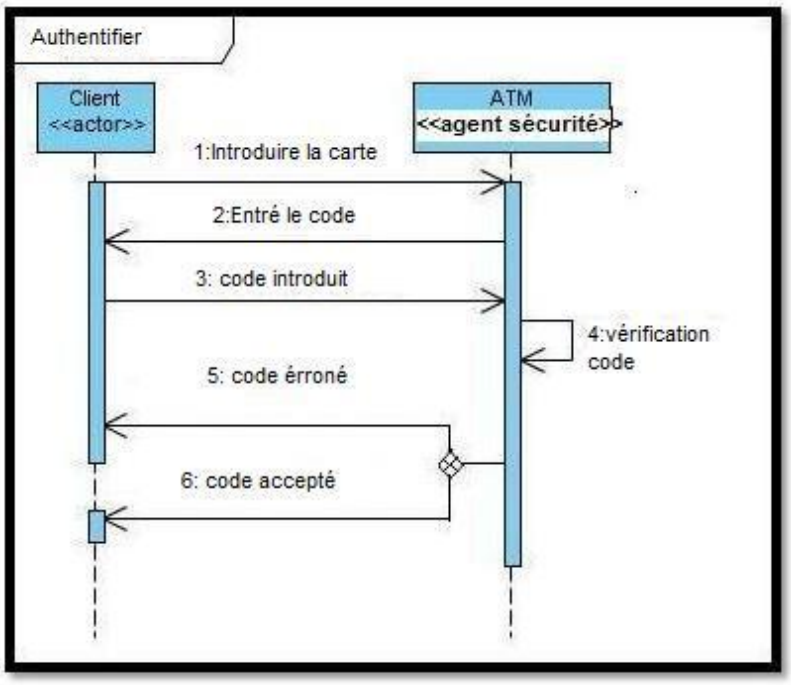


Figure4.27. Le diagramme d'identification de rôle AUML d'authentification

- **Diagramme d'identification de rôle de consultation :**

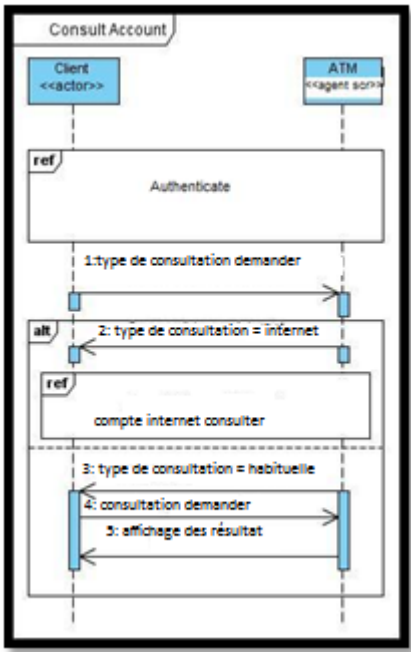


Figure4.28. Le diagramme d'identification de rôle AUML 'consultation de compte'

- **Diagramme d'identification de rôle 'transfert d'argent':**

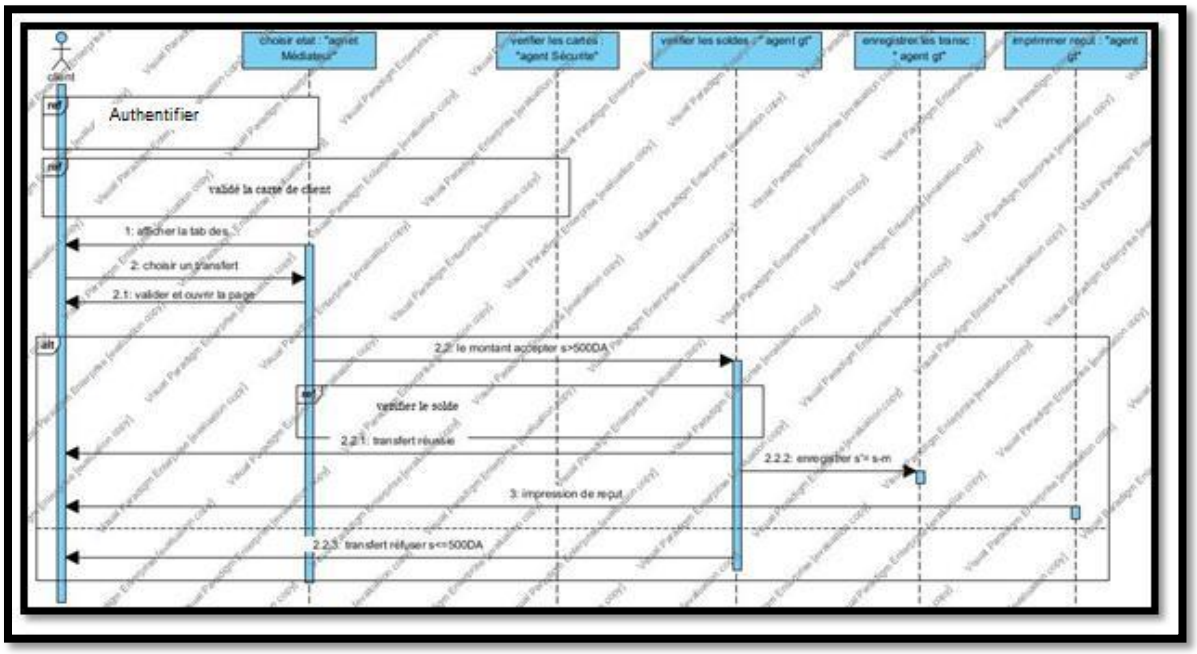


Figure4.29. Le diagramme d'identification de rôle 'transfert d'argent'

On termine notre spécification avec le diagramme d'identification des taches. Chaque agent a un diagramme d'activité composé de deux couloires :

- Coté droit: contient l'ensemble d'activité de l'agent.
- Coté gauche : contient des activités représentant les autres agents en interaction.

On prend par exemple deux diagrammes d'identification des taches :

- **Diagramme des taches de l'agent de gestionnaire de transaction:**

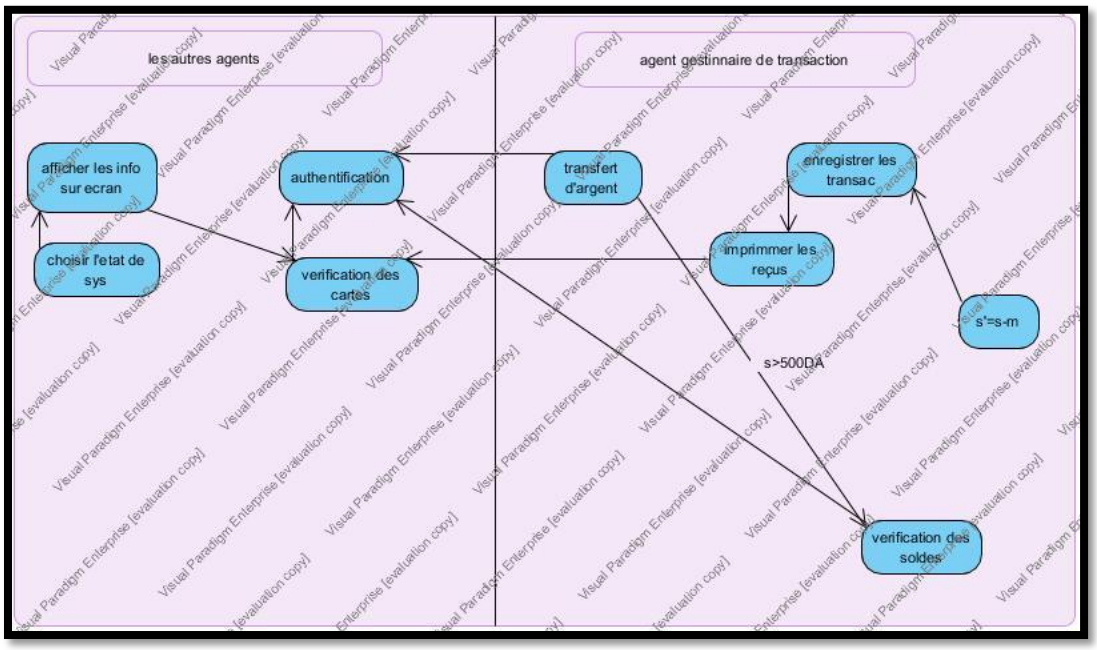


Figure4.30. Le diagramme d'identification des taches de l'agent gestionnaire de transaction

- **Diagramme des taches de l'agent de sécurité:**

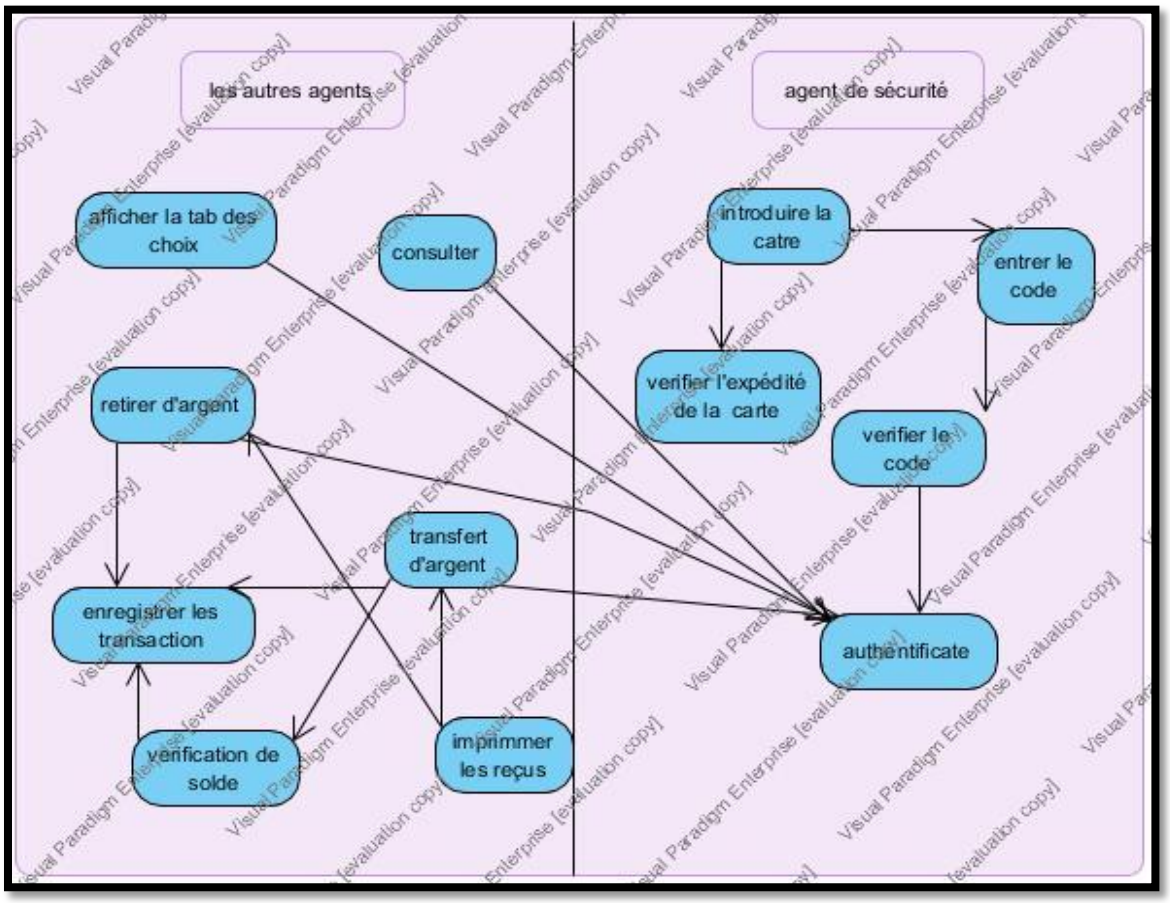


Figure 4.31. Le diagramme d'identification des taches de l'agent de sécurité

B. la validation de la spécification générer :

Dans cette partie on va voir une validation de notre spécification à partir des captures d'écran de quelque exemple :

- **Génération de specification Maude:**

Dans cette partie on va déclarer l'état de notre système

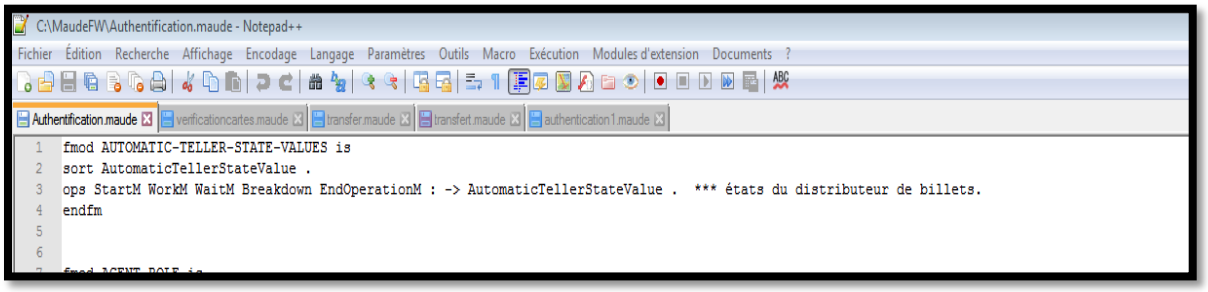


Figure 4.32. Déclaration des ATM en Maude

- **la déclaration des rôles d'agents :**

Cette tache nos permet d'affecter des rôles aux agents

```
6
7  fmod AGENT-ROLE is
8  sort TypeRoleAg .
9  ops mediateur securite-verification-carte GT-enregistrer-transaction GT-impression-reçus  : -> TypeRoleAg .   *** Module qui permet d'identifier quatre types de
10 endfm
11
```

Figure 4.33. Déclaration des rôles d'agents en Maude

- **la déclaration des messages :**

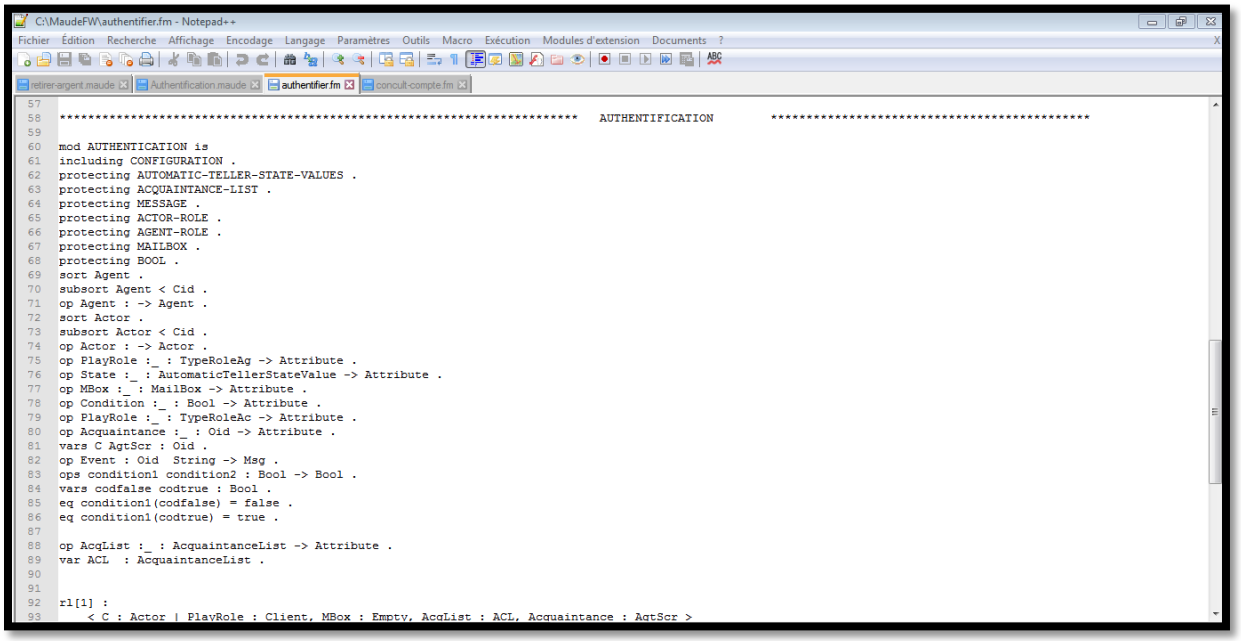
Cette tache nos permet de déclarer les messages entre agents

```
40
41
42 mod MESSAGE is              *** les msg échanger entre agents
43 including CONFIGURATION .
44 protecting STRING .
45 sort content .
46
47 ops introductioncard entercode consultationtyperequest consutatiotypeordinary codehasbeenentered acceptedcode wrongcode
48     refusetransfer websitehasbeenentered websiteresponse consultationrequest displayresult requestamount amountrefused
49     amountaccepted withdrawamount askreloading reloadfinished Distributorreloaded enteramounttotransfer amounthasbeenentered transferfinished : -> content .
50
51 sort Actor .
52 subsort Actor < Oid .
53 op Message : Oid Oid content -> Msg .
54
55 endm
56
```

Figure 4.34. Déclaration des messages en Maude

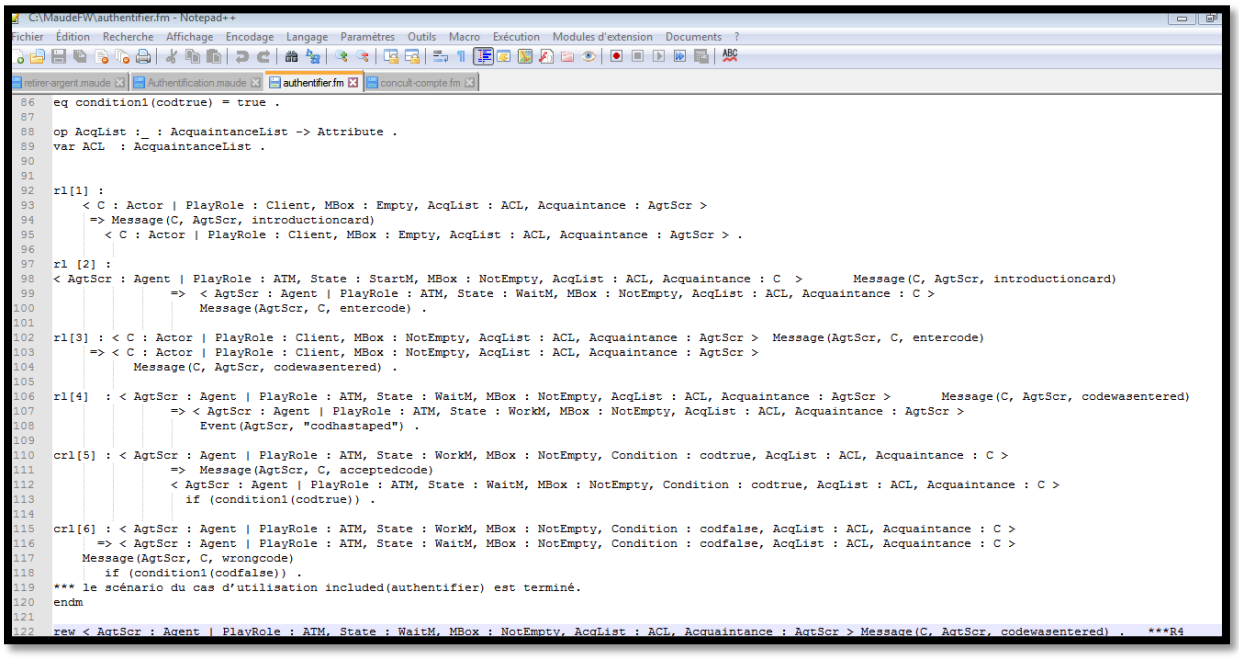
- **l'authentification :**

Dans Cette tache on va déclarer le module d'authentification



```
57
58  ********************************************************************  AUTHENTIFICATION        ******************************************
59
60  mod AUTHENTICATION is
61  including CONFIGURATION .
62  protecting AUTOMATIC-TELLER-STATE-VALUES .
63  protecting ACQUAINTANCE-LIST .
64  protecting MESSAGE .
65  protecting ACTOR-ROLE .
66  protecting AGENT-ROLE .
67  protecting MAILBOX .
68  protecting BOOL .
69  sort Agent .
70  subsort Agent < Cid .
71  op Agent : -> Agent .
72  sort Actor .
73  subsort Actor < Cid .
74  op Actor : -> Actor .
75  op PlayRole :_ : TypeRoleAg -> Attribute .
76  op State :_ : AutomaticTellerStateValue -> Attribute .
77  op MBox :_ : MailBox -> Attribute .
78  op Condition :_ : Bool -> Attribute .
79  op PlayRole :_ : TypeRoleAc -> Attribute .
80  op Acquaintance :_ : Oid -> Attribute .
81  vars C AgtScr : Oid .
82  op Event : Oid  String -> Msg .
83  ops condition1 condition2 : Bool -> Bool .
84  vars codfalse codtrue : Bool .
85  eq condition1(codfalse) = false .
86  eq condition1(codtrue) = true .
87
88  op AcqList :_ : AcquaintanceList -> Attribute .
89  var ACL  : AcquaintanceList .
90
91
92  rl[1] :
93     < C : Actor | PlayRole : Client, MBox : Empty, AcqList : ACL, Acquaintance : AgtScr >
```

Figure4.35a. l'authentification en Maude



```
86  eq condition1(codtrue) = true .
87
88  op AcqList :_ : AcquaintanceList -> Attribute .
89  var ACL  : AcquaintanceList .
90
91
92  rl[1] :
93      < C : Actor | PlayRole : Client, MBox : Empty, AcqList : ACL, Acquaintance : AgtScr >
94       => Message(C, AgtScr, introductioncard)
95         < C : Actor | PlayRole : Client, MBox : Empty, AcqList : ACL, Acquaintance : AgtScr > .
96
97  rl [2] :
98  < AgtScr : Agent | PlayRole : ATM, State : StartM, MBox : NotEmpty, AcqList : ACL, Acquaintance : C  >       Message(C, AgtScr, introductioncard)
99                  =>  < AgtScr : Agent | PlayRole : ATM, State : WaitM, MBox : NotEmpty, AcqList : ACL, Acquaintance : C >
100                    Message(AgtScr, C, entercode) .
101
102 rl[3] : < C : Actor | PlayRole : Client, MBox : NotEmpty, AcqList : ACL, Acquaintance : AgtScr >  Message(AgtScr, C, entercode)
103      => < C : Actor | PlayRole : Client, MBox : NotEmpty, AcqList : ACL, Acquaintance : AgtScr >
104           Message(C, AgtScr, codewasentered) .
105
106 rl[4]  : < AgtScr : Agent | PlayRole : ATM, State : WaitM, MBox : NotEmpty, AcqList : ACL, Acquaintance : AgtScr >       Message(C, AgtScr, codewasentered)
107                 => < AgtScr : Agent | PlayRole : ATM, State : WorkM, MBox : NotEmpty, AcqList : ACL, Acquaintance : AgtScr >
108                    Event(AgtScr, "codhastaped") .
109
110 crl[5] : < AgtScr : Agent | PlayRole : ATM, State : WorkM, MBox : NotEmpty, Condition : codtrue, AcqList : ACL, Acquaintance : C >
111                 =>  Message(AgtScr, C, acceptedcode)
112                 < AgtScr : Agent | PlayRole : ATM, State : WaitM, MBox : NotEmpty, Condition : codtrue, AcqList : ACL, Acquaintance : C >
113                   if (condition1(codtrue)) .
114
115 crl[6] : < AgtScr : Agent | PlayRole : ATM, State : WorkM, MBox : NotEmpty, Condition : codfalse, AcqList : ACL, Acquaintance : C >
116       => < AgtScr : Agent | PlayRole : ATM, State : WaitM, MBox : NotEmpty, Condition : codfalse, AcqList : ACL, Acquaintance : C >
117     Message(AgtScr, C, wrongcode)
118        if (condition1(codfalse)) .
119 *** le scénario du cas d'utilisation included(authentifier) est terminé.
120 endm
121
122 rew < AgtScr : Agent | PlayRole : ATM, State : WaitM, MBox : NotEmpty, AcqList : ACL, Acquaintance : AgtScr > Message(C, AgtScr, codewasentered) .   ***R4
```

Figure4.35b. l'authentification en Maude

Dans la figure 4.35c on va voir l'exécution de module d'authentification

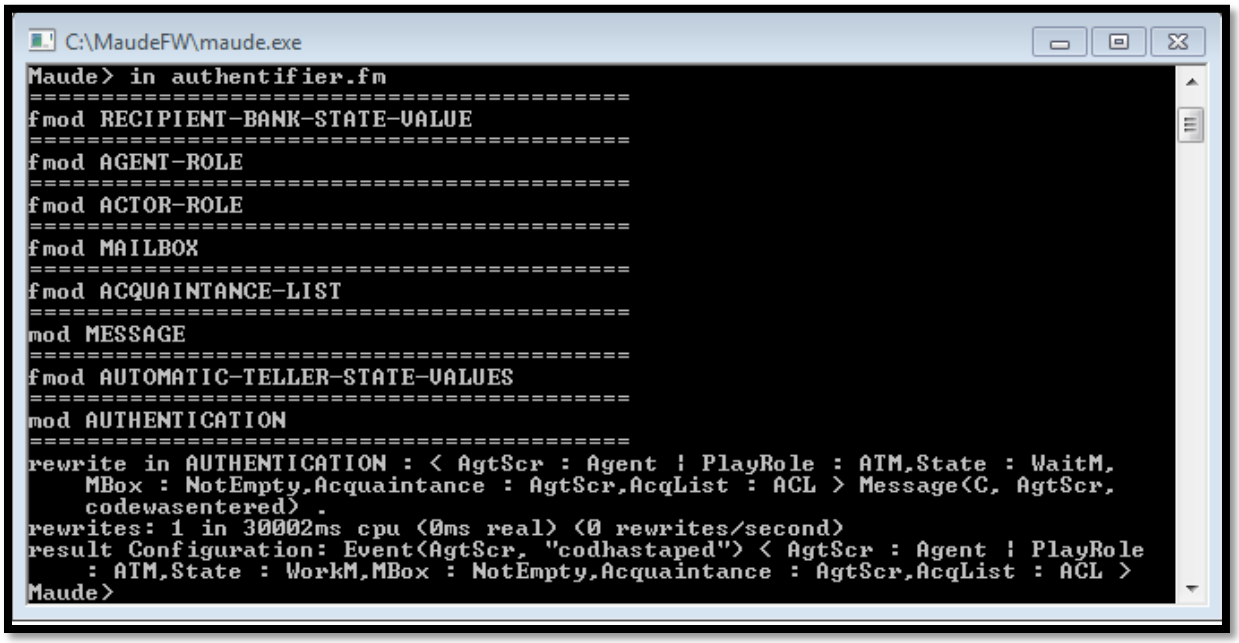


Figure4.35c. l'authentification en Maude

- **le module de consulter le compte via internet:**

  Cette tache nous permet d'effectuer une consultation des comptes via le net

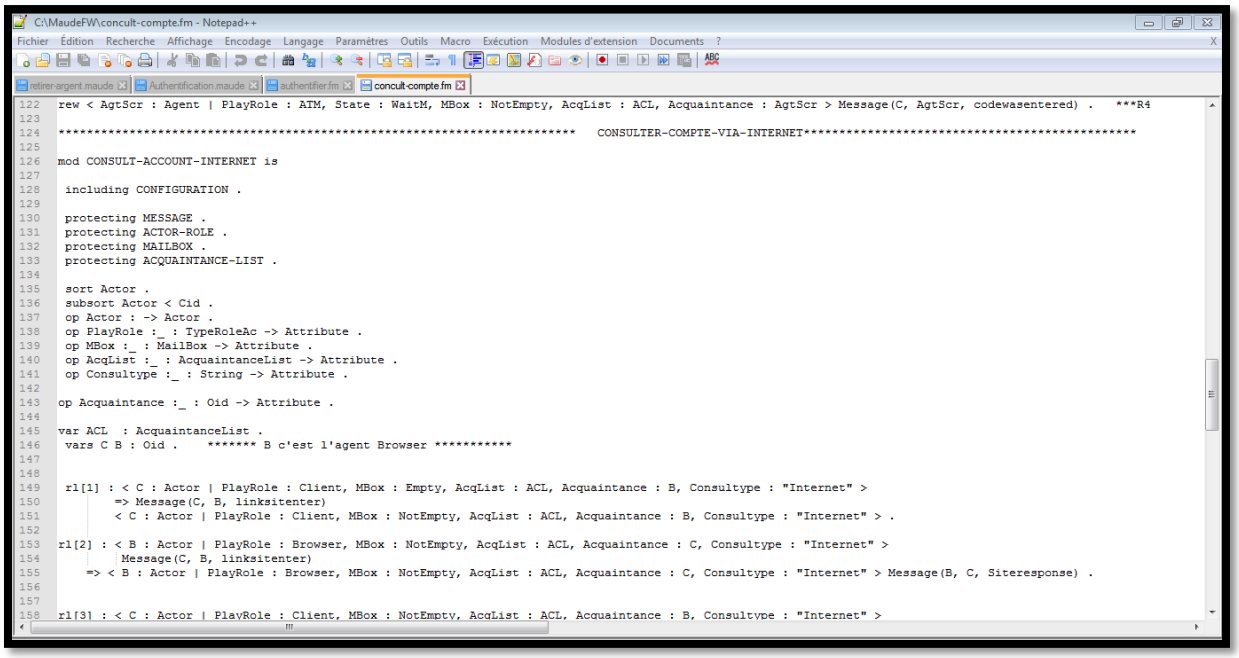
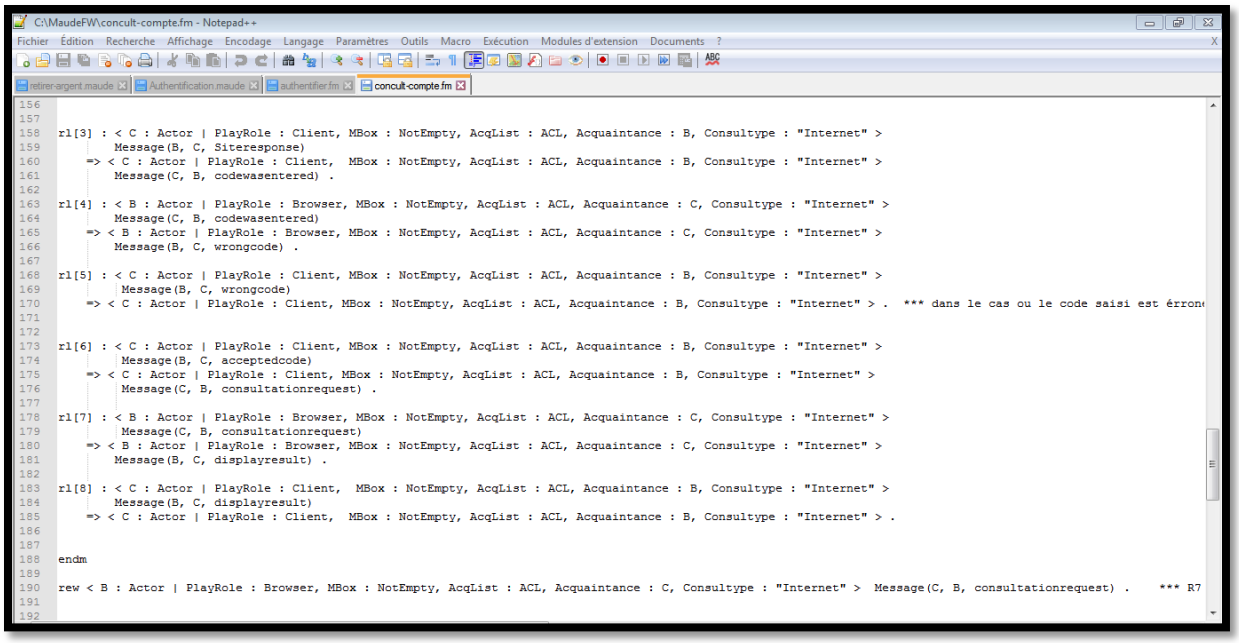


Figure4.36a. la consultation de compte via le net en Maude

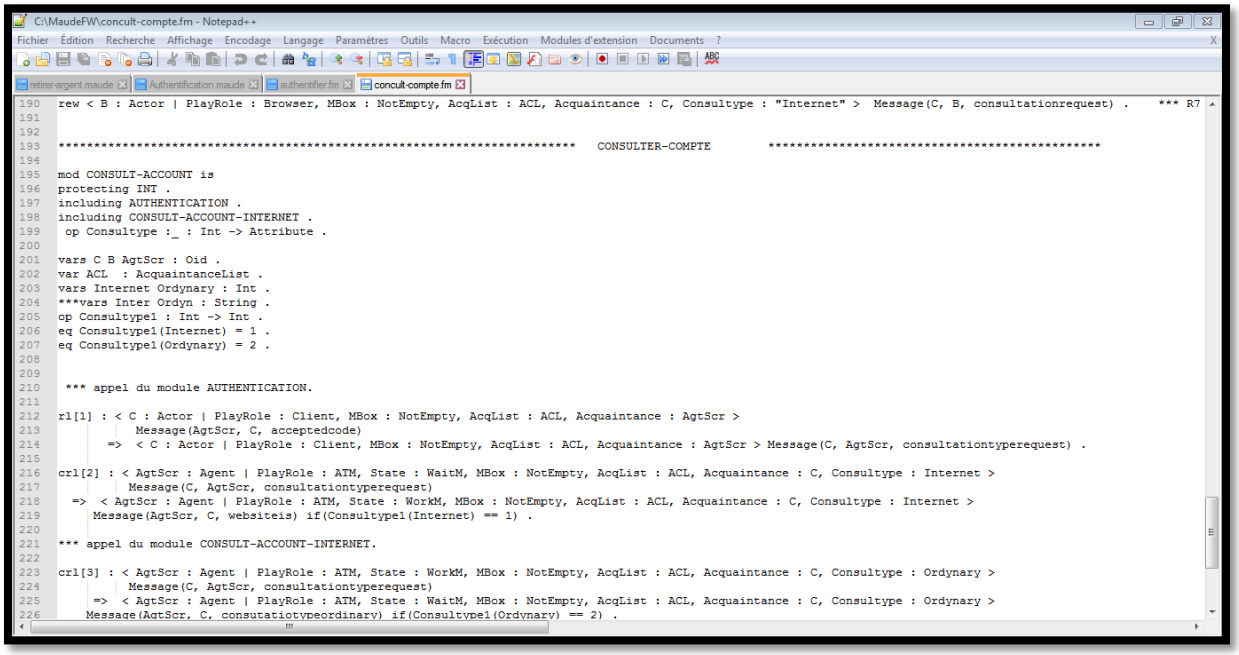


```
rl[3] : < C : Actor | PlayRole : Client, MBox : NotEmpty, AcqList : ACL, Acquaintance : B, Consultype : "Internet" >
        Message(B, C, Siteresponse)
     => < C : Actor | PlayRole : Client,  MBox : NotEmpty, AcqList : ACL, Acquaintance : B, Consultype : "Internet" >
        Message(C, B, codewasentered) .

rl[4] : < B : Actor | PlayRole : Browser, MBox : NotEmpty, AcqList : ACL, Acquaintance : C, Consultype : "Internet" >
        Message(C, B, codewasentered)
     => < B : Actor | PlayRole : Browser, MBox : NotEmpty, AcqList : ACL, Acquaintance : C, Consultype : "Internet" >
        Message(B, C, wrongcode) .

rl[5] : < C : Actor | PlayRole : Client, MBox : NotEmpty, AcqList : ACL, Acquaintance : B, Consultype : "Internet" >
        Message(B, C, wrongcode)
     => < C : Actor | PlayRole : Client, MBox : NotEmpty, AcqList : ACL, Acquaintance : B, Consultype : "Internet" > .  *** dans le cas ou le code saisi est érron

rl[6] : < C : Actor | PlayRole : Client, MBox : NotEmpty, AcqList : ACL, Acquaintance : B, Consultype : "Internet" >
        Message(B, C, acceptedcode)
     => < C : Actor | PlayRole : Client, MBox : NotEmpty, AcqList : ACL, Acquaintance : B, Consultype : "Internet" >
        Message(C, B, consultationrequest) .

rl[7] : < B : Actor | PlayRole : Browser, MBox : NotEmpty, AcqList : ACL, Acquaintance : C, Consultype : "Internet" >
        Message(C, B, consultationrequest)
     => < B : Actor | PlayRole : Browser, MBox : NotEmpty, AcqList : ACL, Acquaintance : C, Consultype : "Internet" >
        Message(B, C, displayresult) .

rl[8] : < C : Actor | PlayRole : Client,  MBox : NotEmpty, AcqList : ACL, Acquaintance : B, Consultype : "Internet" >
        Message(B, C, displayresult)
     => < C : Actor | PlayRole : Client,  MBox : NotEmpty, AcqList : ACL, Acquaintance : B, Consultype : "Internet" > .


endm

rew < B : Actor | PlayRole : Browser, MBox : NotEmpty, AcqList : ACL, Acquaintance : C, Consultype : "Internet" >  Message(C, B, consultationrequest) .    *** R7
```

Figure4.36b. la consultation de compte via le net en Maude

- **le module de consulter des comptes ordinaire**

  Cette tache nos permet d'affecter une consultation ordinaire de comptes



```
rew < B : Actor | PlayRole : Browser, MBox : NotEmpty, AcqList : ACL, Acquaintance : C, Consultype : "Internet" >  Message(C, B, consultationrequest) .    *** R7


*********************************************************************   CONSULTER-COMPTE        ***********************************************

mod CONSULT-ACCOUNT is
protecting INT .
including AUTHENTICATION .
including CONSULT-ACCOUNT-INTERNET .
 op Consultype :_ : Int -> Attribute .

vars C B AgtScr : Oid .
var ACL  : AcquaintanceList .
vars Internet Ordynary : Int .
***vars Inter Ordyn : String .
op Consultype1 : Int -> Int .
eq Consultype1(Internet) = 1 .
eq Consultype1(Ordynary) = 2 .


 *** appel du module AUTHENTICATION.

rl[1] : < C : Actor | PlayRole : Client, MBox : NotEmpty, AcqList : ACL, Acquaintance : AgtScr >
          Message(AgtScr, C, acceptedcode)
       =>  < C : Actor | PlayRole : Client, MBox : NotEmpty, AcqList : ACL, Acquaintance : AgtScr > Message(C, AgtScr, consultationtyperequest) .

crl[2] : < AgtScr : Agent | PlayRole : ATM, State : WaitM, MBox : NotEmpty, AcqList : ACL, Acquaintance : C, Consultype : Internet >
          Message(C, AgtScr, consultationtyperequest)
  =>  < AgtScr : Agent | PlayRole : ATM, State : WorkM, MBox : NotEmpty, AcqList : ACL, Acquaintance : C, Consultype : Internet >
    Message(AgtScr, C, websiteis) if(Consultype1(Internet) == 1) .

*** appel du module CONSULT-ACCOUNT-INTERNET.

crl[3] : < AgtScr : Agent | PlayRole : ATM, State : WorkM, MBox : NotEmpty, AcqList : ACL, Acquaintance : C, Consultype : Ordynary >
          Message(C, AgtScr, consultationtyperequest)
     =>  < AgtScr : Agent | PlayRole : ATM, State : WaitM, MBox : NotEmpty, AcqList : ACL, Acquaintance : C, Consultype : Ordynary >
   Message(AgtScr, C, consutatiotypeordinary) if(Consultype1(Ordynary) == 2) .
```

Figure4.37a. la consultation ordinaire des comptes en Maude

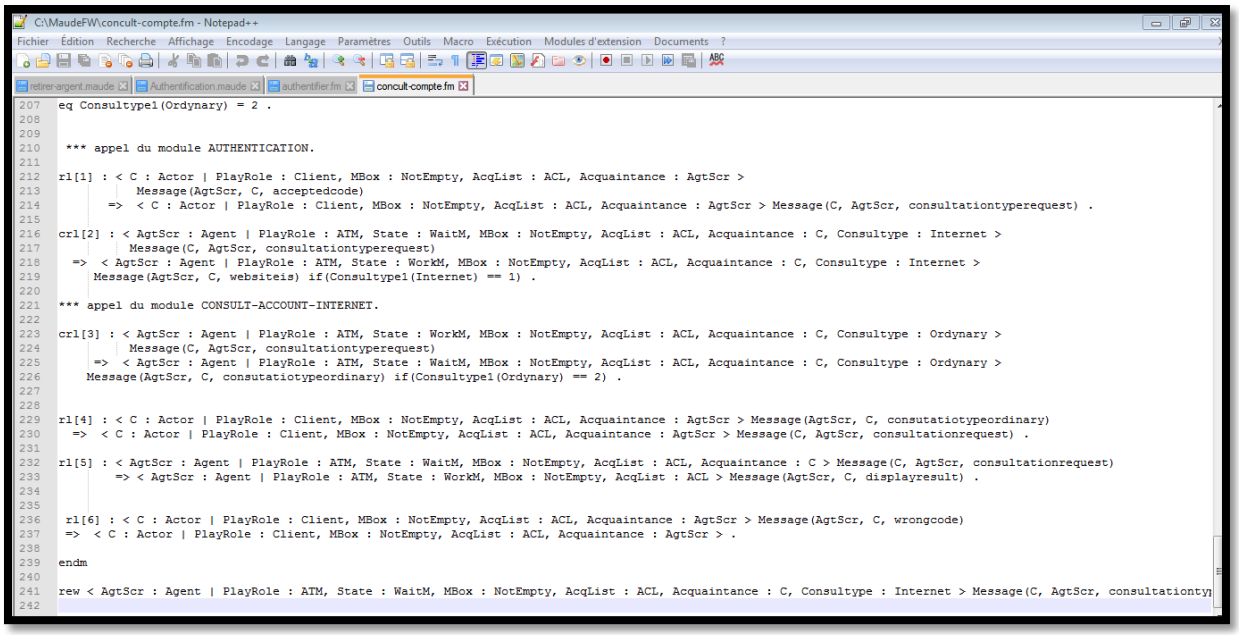


Figure4.37b. la consultation ordinaire des comptes en Maude

Dans la figure 4.37c, 4.37d, 4.37e on va voir l'exécution de cette tache

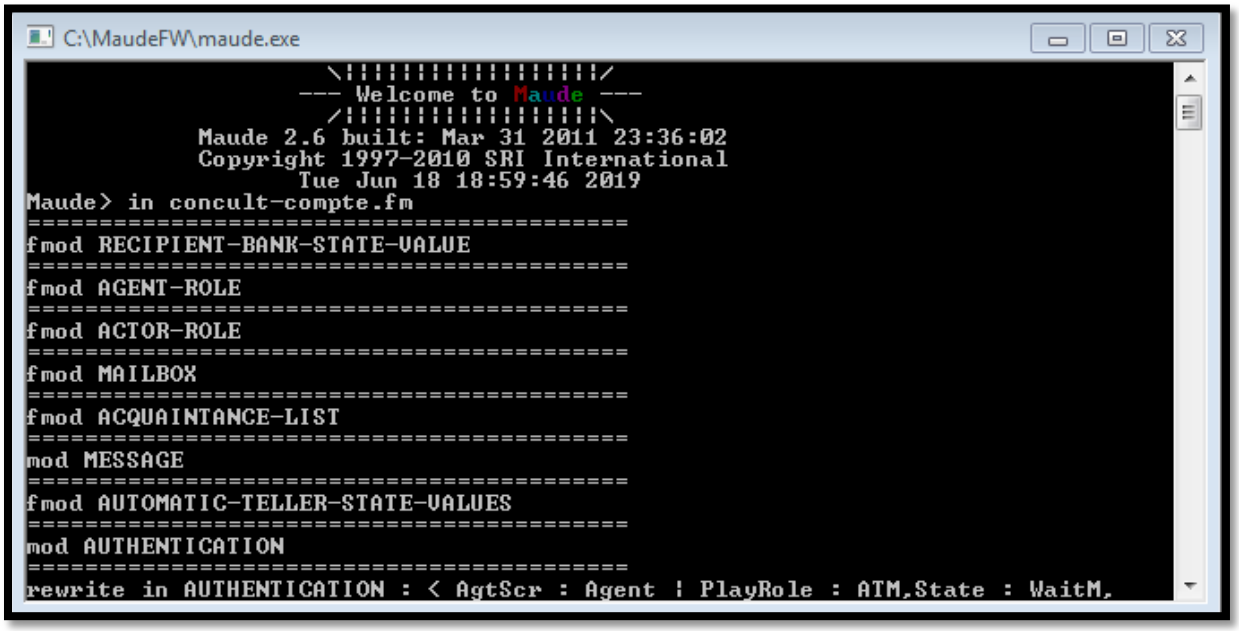


Figure4.37c. la consultation (ordinaire, via net) des comptes en Maude

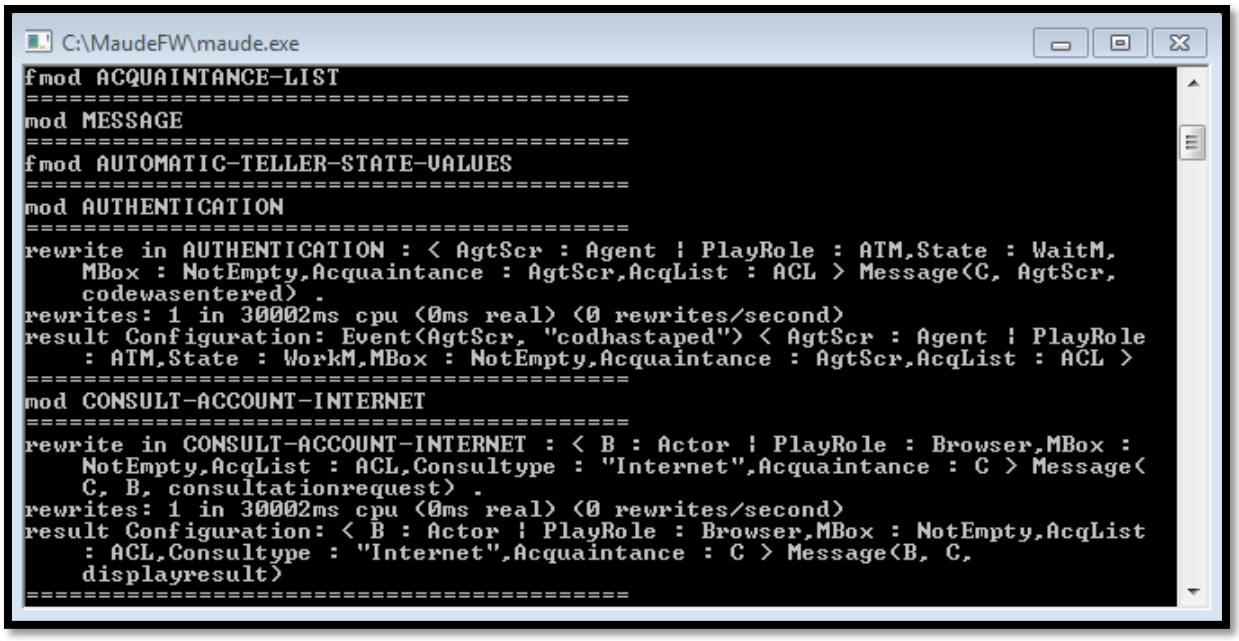


Figure4.37d. la consultation (ordinaire, via net) des comptes en Maude

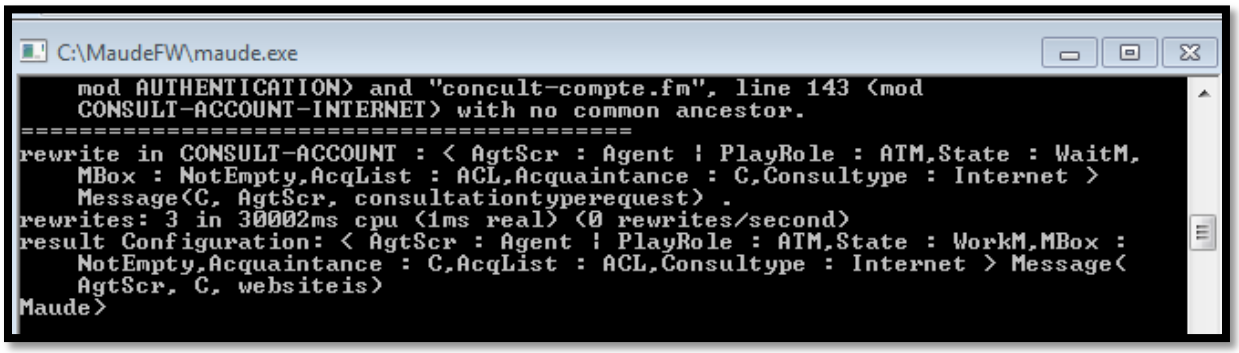


Figure4.37e. la consultation (ordinaire, via net) des comptes en Maude

- **le module de transfert d'argent:**

  Dans Cette tache on va voir le module de transfert d'argent

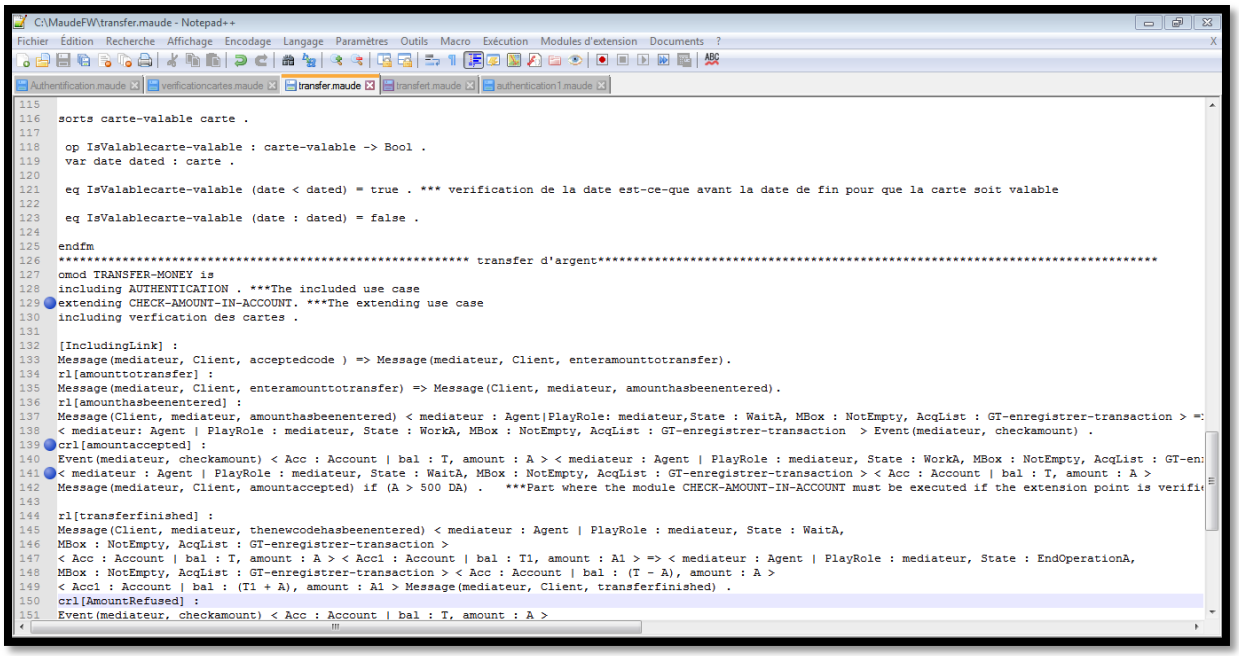


Figure 4.38.le module de transfert en Maude

- **le module de retrait d'argent:**

  Dans cette tache on va voir comment on peut retire l'argent

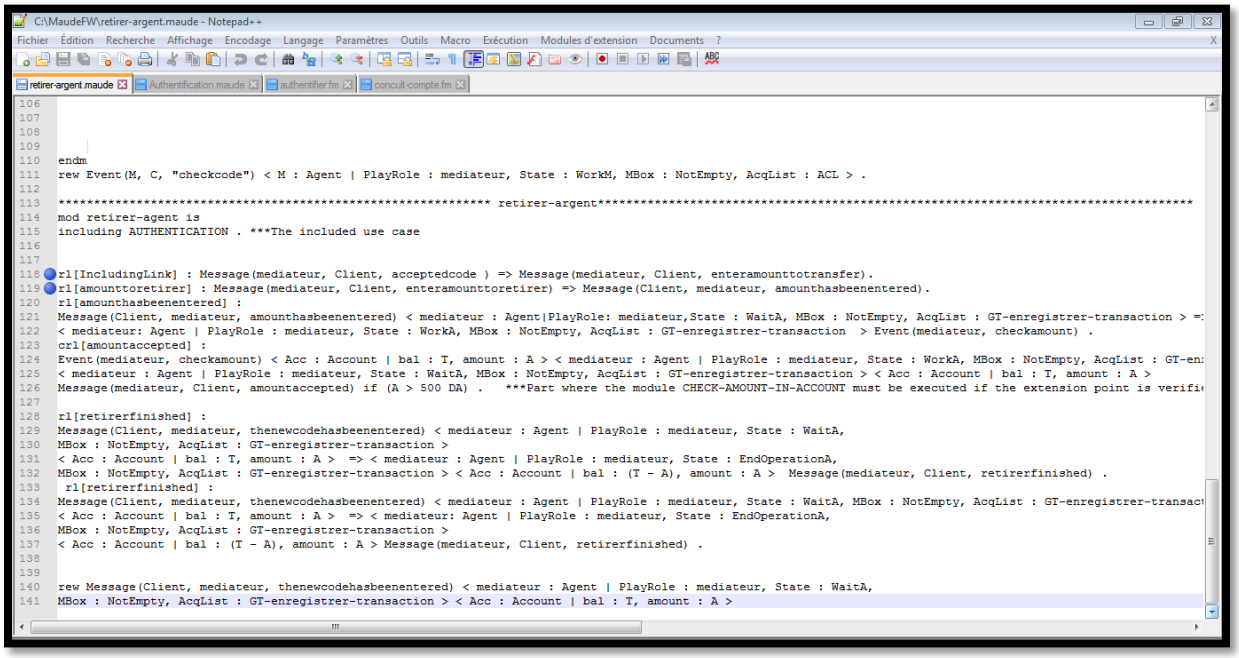


Figure 4.39.le module de retrait d'argent en Maude

X. Conclusion :

La formalisation des exigences fonctionnelles représente une activité importante lors du processus de développement de systèmes multi-agents, c'est une base solide pour la vérification et la validation des systèmes. Dans ce projet, nous avons proposé une approche permettant, de générer une spécification formelle écrite en langage Maude à partir d'un ensemble de diagrammes illustré dans la méthodologie PASSI qui décrivent les besoins fonctionnels des SMA. Cette description nous offre plusieurs avantages comme la compréhension approfondie du système est d'autres avantages assez intéressants. Notre framework formel contient les briques de bases (modules) pour la formalisation de besoins fonctionnels des SMA.

Conclusion et perspective:

Nelson Mandela

«L'éducation est l'arme la plus puissante qu'on puisse utiliser pour changer le monde. »

La phase analyse représente une activité très importante dans le cycle de développement des applications logiciel puisque il à comme but d'éviter de produire des applications qui ne répond pas aux besoins d'utilisateurs. Plusieurs approche ont été proposé pour spécifier les besoins fonctionnels des usagés, parmi elles se sont les spécifications formelle qui nous guidés à prévoir une bonne qualité de 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 permettant la translation d'un ensemble de modèles décrivant les besoins fonctionnels des SMA de la méthodologie basé agent dite PASSI (description graphique) vers une spécification formelle écrite en langage 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 suivantes :

- Tenir en compte les versions récentes de la méthodologie PASSI.
- Étendre notre approche afin de supporter les systèmes multi-agents ouverts ou le nombre des agents n'est pas connu a priori.

Références Bibliographiques

CHAPITRE 1 :

[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.

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

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

[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.

[Sri97]: Computer Science Laboratory. SRI International 1997.

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

[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.

[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.

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

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

[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.

[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.

[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.

[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.

[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.

[Mes98]: Jose 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{7, 1997, Selected Papers, volume 1376 of Lecture Notes in Computer Science, pages 1861. Springer, 1998.

[Cla09]: Manuel Clavel, Francisco Duran, Steven Eker, Patrick Lincoln, Narciso Marti-Oliet, José Meseguer, Carolyn Talcott. " Maude Manual (Version 2.4) " October 2008 (Revised February 2009) .

[Pet05]: Csaba Peter Olveczky. " Formal Modeling and analysis of distributed systems in Maude " Nov 29, 2005.

CHAPITRE 2:

[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.

[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.

[Cos et al 14]: Cossentino, M. and V. Seidita (2014). PASSI: Process for Agent Societies Specification and Implementation. Handbook on Agent-Oriented Design Processes. M. Cossentino, V. Hilaire, A. Molesini and V. Seidita, Springer Berlin Heidelberg: 287-329

[Dem 95]: 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.

[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.

[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.

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

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

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

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

[Sho93]: Y. Shoham. " Agent oriented programming " journal of Artificial Intelligence, Vol 60 1993.

[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.

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

[Lab93]: S. Labidi, W. Lejouad. " De l'Intelligence Artificielle Distribuée aux Systèmes Multi-Agents " Rapport de recherche de l'INRIA N°2001, 1993.

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

[Flo02]: Université polytechnique Bucharest, Agent Intelligent cours web interactif, http://turing.cs.pub.ro/auf2/html/chapters/chapter2/chapter_2_2_1.html, 2002.

[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.

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

[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.

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

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

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

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

[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.

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

[Lab03]: S. Labidi et W. Lejouad. " De l'intelligence Artificielle distribuée aux Systèmes Multi-agents ", Rapport de Recherche de l'INRIA N° 2001, 1993.

[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.

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

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

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

[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.

[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.

[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.

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

[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.

[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.

[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.

[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.

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

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

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

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

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

[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.

[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

[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.

[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.

[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.

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

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

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

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

[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.

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

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

[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.

[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

[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.

[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.

[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 ". these doctorat discipline informatique UNIVERSITÉ PARIS IX-DAUPHINE Page 62 mai 2003.

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

[GLI98]: P. Glize, M.P. Gleizes et Camps Valérie. " Une théorie de l'apprentissage fondée sur l'auto-organisation par coopération ". 1998.

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

CHAPITRE 3:

[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

[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.

[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.

[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.

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

[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.

[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.

[Kin96]: D. Kinny, M. Georgeff, and A. Rao. " A methodology and modeling 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 Autonmous Agents in a Multi-Agent World MAAMAW'96, (LNAI Volume1038). Springer-Verlag : Heidelberg, Germany,1996.

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

[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.

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

[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 de maîtrise, Département de mathématiques et d'informatique, Université du Québec à Trois-Rivières (Canada), décembre 2001.

[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.

[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.

[Mas12]: Massimo Cossentino **"**From Requirements to Code with the PASSI Methodology" **April 2012**

Chapitre 4:

[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.

[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.

[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.

[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.

[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.

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

[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.

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

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

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

[Ham et al 10] : F. Hamidane, F. Mokhati and H. Belleili-Souici. Towards Formalizing Multi-Agent Systeme Functional Requirements In Maude. International Journal of Advanced Research in Computer Science (IJARCS) Volume 1, No. 2, August 2010, pp. 41-48.

[Mas12]: Massimo Cossentino **"**From Requirements to Code with the PASSI Methodology" **April 2012**