

République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique

Université de Batna
Faculté des sciences
Département D'informatique

MEMOIRE
Présenté en vue de l'obtention
du diplôme de Magister en Informatique

Thème

**UNE APPROCHE DE TEST
DES SYSTEMES MULTIAGENTS
BASEE SUR DES MODELES FORMELS**

Présenté par :
AII GUECHI Farida

Devant le jury:

Pr Zidani A., Professeur à l'université Batna, En qualité de président
Pr Bilami A., Professeur à l'université Batna, En qualité d'examineur
Pr Benmohammed Mohamed à l'université Constantine-2, En qualité d'examineur
Dr Maamri R., Maitre de conférences à l'université Constantine-2, En qualité de rapporteur

Dédicace

D'abord je tiens à remercier Allah.

Je dédie ce mémoire :

*A mon père et ma mère qui m'ont beaucoup aidée et encouragée
durant mes études, qu'Allah les protègent.*

A ma belle mère et mon beau père.

A mon mari Sofiane.

A mon frère Toufik et sa femme Fadila.

A mes beau frères: Kheir eldine et Abd elnassar

A mes sœur Jihane, Wissem et Nabila.

*A mes belles sœur: Hanan, Sana et Intisar et son fils Abd
elmouiz et sa fille Ibtihal.*

*A mes nièces Narimene, Meriem elbatoul, Joumana et Abd
elmouemen.*

A mes amies Zahra, Assia, Mouchira, Sana et Amel.

*ET à tous ceux qui m'ont aidé de près ou de loin à accomplir ce
travail.*

KAMAR

Remerciements

Avant tout, merci mon DIEU de m'avoir donné le courage, la volonté et la patience de mener ce travail à terme.

Je tien à remercier vivement mon encadreur

Dr. MAAMRI Ramdane

, pour son encadrement avec patience et pour son aide et ces conseils.

Je tien à remercier

Dr. KISSOUM Yacine

, pour son encouragement et ses remarques pertinentes qui m'ont permis de mieux structurer ce travail et de mieux le décrire.

Je remercie également les membres du jury, Pr Zidani.A, Pr Bilami.A, et Pr Benmohammed.M qui ont accepté d'évaluer ce travail.

Résumé. Le test des systèmes Multi-Agent (SMA) a besoin de techniques appropriées pour évaluer les comportements autonomes de l'agent aussi bien que les propriétés de distribution, sociales et délibératives, qui sont particulières à ces systèmes. Parmi ces techniques, nous trouvons le test basé sur les modèles, il est basé sur un modèle de système afin de produire des cas de test abstraits. Pour que ces derniers puissent être soumis au système sous test, les cas de test abstraits doivent être transformés en des cas de test concrets. L'approche de test basé sur le modèle pour les agents mobiles est une approche qui a besoin d'un algorithme clair de concrétisation, ce dernier est proposée et appliquée dans ce mémoire en utilisant le problème du transport de ressources par des robots comme cas d'étude.

Mots-clés : le test des systèmes Multi-agents, Test basé sur les modèles, concrétisation, cas de test concrets, message de FIPA ACL

Abstract. Testing Multi-Agent Systems (MAS) needs suitable techniques to evaluate agent's autonomous behaviors as well as distribution, social and deliberative properties, which are particular to these systems. Among these techniques, we find "Model based testing" technique; it is based on a system model in order to produce abstract test cases. To run these last ones against systems under test, the abstract test cases have to be transformed to concrete test cases. The model based testing approach for mobile agents is an approach which need a clear concretization algorithm, this last one is proposed in this paper and applied using the problem of resources transportation by robots as case study.

Keywords: Multi-agent system testing, Model-Based Testing, concretization, concrete test cases, FIPA ACL messages

ملخص. اختبار أنظمة متعددة الوكلاء يحتاج تقنيات مناسبة لتقييم سلوكيات الوكيل التي تتمتع بحكم ذاتي وكذلك خصائص التوزيع والاجتماعية والتداولية، والتي هي محددة لهذه النظم. ومن بين هذه التقنيات، نجد الاختبار استنادا إلى النماذج، لأنه يقوم على نموذج نظام لتوليد حالات الاختبار مجردة. بحيث يمكن تقديمها إلى النظام تحت الاختبار، يجب أن تتحول حالات الاختبار المجردة إلى حالات الاختبار ملموسة. المقاربة القائمة على نموذج الاختبار لوكلاء المحمول هو النهج الذي يحتاج إلى أعمال خوارزمية واضحة، مقترحة ومطبقة في هذه المدكرة باستخدام مشكلة نقل الموارد من الروبوتات كحالة الدراسة.

كلمات البحث: اختبار متعدد النظم الوكلاء، استنادا إلى نماذج اختبار، وتنفيذ، واختبار الحالات الملموسة، رسالة FIPA

ACL

Table des matières

Chapitre 1: Introduction générale.....	11
Chapitre 2: Les systèmes multi-agents.....	14
2.1 La notion d'agent	14
2.1.1 Définition	14
2.1.2 Les caractéristiques multidimensionnelles d'un agent.....	16
2.1.3 Différentes catégories d'agents.....	18
2.1.3.1 Selon la granularité	18
2.1.3.2 Selon la fonctionnalité	20
2.1.3.3 Selon la mobilité	21
2.2 Système multi-agents.....	21
2.2.1 Définition.....	21
2.2.2 Domaines d'application.....	22
2.2.3 Exemples d'un SMA.....	23
2.2.4 Les avantages des systèmes multi-agents.....	24
2.2.5 Les défis et les contraintes des SMA.....	24
2.2.6 Différences entre S.M.A et Système orienté objets.....	25
2.2.7 Interaction entre agents.....	25
2.2.7.1 Situation d'indifférence.....	26
2.2.7.2 Situation de coopération.....	27
2.2.7.3 Situations d'antagonisme.....	28
2.2.8 Communication entre agents.....	29
2.2.8.1 Définition.....	29
2.2.8.2 Les modes de communication.....	29
2.2.9 Les langages de communication dans les SMA.....	31
2.2.9.1 Actes de langage.....	31
2.2.9.2 Le langage KQML.....	32
2.2.9.3 Le langage ACL.....	34
2.2.10 Plateformes multi-agents.....	37
2.2.10.1 La plate-forme SWARM.....	37
2.2.10.2 La plate-forme ZEUS.....	38
2.2.10.3 La plate-forme MADKIT.....	38
2.2.10.4 La plate-forme JADE.....	38
2.3 Conclusion.....	40
Chapitre 3: Le test des systèmes multi-agents.....	41
3.1 Le test de logiciel.....	41
3.1.1 Qu'est-ce qu'un logiciel.....	41
3.1.2 Qu'est ce qu'un test de logiciel.....	41
3.1.3 Terminologie liée au test.....	42
3.2 Pourquoi le test des logiciels.....	44
3.3 Objectif du test.....	45
3.4 Le test dans le cycle de développement.....	45
3.4.1 Le test unitaire.....	45
3.4.2 Le test d'intégration.....	46
3.4.3 Le test de système.....	46
3.4.4 Le test d'acceptation.....	48
3.5 Classification des méthodes de test.....	48
3.5.1 Les méthodes de test statiques.....	48

3.5.2	Les méthodes de test dynamiques.....	49
3.5.2.1	Le test structurel (boite blanche).....	50
3.5.2.2	Le test fonctionnel (boite noire).....	52
3.5.3	Autres méthodes de test.....	53
3.5.3.1	Le test aléatoire.....	53
3.5.3.2	Test basé sur les modèles (Model Based Testing).....	54
3.6	Le critère d'arrêt.....	56
3.7	Difficulté du test.....	57
3.8	Le test des systèmes multi-agents.....	58
3.8.1	Difficultés de test des SMA	58
3.8.2	Niveaux de test des SMA	59
3.8.2.1	Test unitaire.....	60
3.8.2.2	Test d'agent.....	60
3.8.2.3	Test d'intégration ou de groupe.....	60
3.8.2.4	Test de système ou de société.....	60
3.8.2.5	Test d'acceptation.....	60
3.8.3	Aperçu sur le test des systèmes multi-agents.....	60
3.8.3.1	niveau unitaire.....	61
3.8.3.2	Niveau d'agent.....	61
3.8.3.3	Niveau d'intégration.....	63
3.8.3.4	Plusieurs niveaux de test.....	63
3.9	Conclusion.....	64
	Chapitre 4: Les réseaux de Pétri.....	65
4.1	Qu'est ce qu'un modèle?.....	65
4.2	Pourquoi modéliser?.....	65
4.3	Les réseaux de Pétri.....	66
4.3.1	Définition informelle.....	66
4.3.2	Définition formelle.....	67
4.3.3	Marquage.....	68
4.3.4	Evolution d'un réseau de Pétri.....	69
4.4	Qualités des réseaux de Pétri.....	70
4.5	Propriétés des réseaux de Pétri.....	71
4.5.1	Les propriétés dynamiques.....	71
4.5.1.1	Bornitude.....	72
4.5.1.2	Vivacité.....	72
4.5.1.3	Blocage.....	72
4.5.1.4	Réinitiability.....	72
4.5.2	Les propriétés structurelles.....	73
4.6	Les méthodes d'analyse des réseaux de Pétri.....	73
4.7	Structures fondamentales pour la modélisation des systèmes.....	74
4.7.1	Parallélisme.....	74
4.7.2	Synchronisation Mutuelle.....	75
4.7.3	Partage de ressources.....	75
4.8	Quelques extensions des réseaux de Pétri.....	76
4.8.1	Réseaux de Pétri colorés.....	76
4.8.2	Réseaux de Pétri temporisés.....	77
4.8.3	Réseaux de Pétri t-temporels	77
4.8.4	Réseaux de Pétri synchronisés	77
4.8.5	Le paradigme (nets-within-nets) et les Réseaux de référence.....	78
4.8.5.1	Modélisation de la mobilité.....	79

4.9 Outils de modélisation des réseaux de Pétri.....	81
4.10 Utilisation des réseaux de Pétri dans le domaine des SMA.....	82
4.10.1 Modélisation des agents mobiles.....	82
4.10.2 Modéliser l'aspect social.....	82
4.10.3 Modélisation des interactions.....	82
4.11 Conclusion.....	82
Chapitre 5: Test basé modèle des agents mobiles utilisant le paradigme des réseaux dans les réseaux.....	84
5.1 Test basé modèle des agents mobiles utilisant le paradigme des réseaux dans les réseaux.....	84
5.1.1 Modélisation	85
5.1.2 Validation du modèle.....	86
5.1.3 Simulation et génération des cas de tests abstraits.....	87
5.2 Concrétisation des cas de test abstraits.....	88
5.2.1 Analyse statique.....	88
5.2.2 Instrumentation.....	89
5.2.3 La construction des cas de test concrets.....	90
5.2.3.1 Génération de la partie prologue.....	90
5.2.3.2 Génération de la partie corps de test et du verdict.....	91
5.2.3.3 Génération de la partie épilogue.....	92
5.2.4 Exécution du système et évaluation des résultats.....	93
5.3 Conclusion.....	94
Chapitre 6: Etude de cas et implémentation.....	95
6.1 le problème de transport de ressources par des robots.....	95
6.2 Modélisation du problème de transport de ressources.....	96
6.3 Validation du modèle.....	103
6.4 Simulation et génération des cas de tests abstraits	103
6.5 Concrétisation des cas de test abstraits.....	105
6.5.1 Analyse statique.....	106
6.5.2 Instrumentation.....	106
6.5.3 La construction des cas de test concrets.....	107
6.6 Présentation de l'application développée.....	110
6.6.1 Choix techniques.....	110
6.6.2 Interfaces.....	110
6.7 Conclusion.....	118
Chapitre 7: Conclusion et perspectives.....	119
Bibliographie.....	122

Liste des figures

Chapitre 2: Les Systèmes multi-agents

Figure 2.1: Structure générale d'un agent réactif	18
Figure 2.2: Structure d'un agent cognitif	19
Figure 2.3: Représentation imagée d'un agent en interaction avec son environnement et les autres agents	22
Figure 2.4: Communication indirecte via l'environnement	30
Figure 2.5: Communication par partage d'informations.....	30
Figure 2.6: Communication par envoi de messages.....	31
Figure 2.7: Un message KQML.....	33
Figure 2.8: Un message FIPA-ACL.....	35
Figure 2.9: Le protocole request du FIPA.....	37
Figure 2.10: La fenêtre de JADE.....	40

Chapitre 3: Le Test des Systèmes multi-agents

Figure 3.1: Concepts de vérification et validation.....	43
Figure 3.2: Relation entre défaillance / faute / erreur	43
Figure 3.3: Coût du test	44
Figure 3.4: Cycle de vie en V	45
Figure 3.5: Les étapes du test dynamique	50
Figure 3.6: Un exemple pour le test structurel	51
Figure 3.7: Le processus de test basé sur les modèles	55
Figure 3.8: Champs d'application du test basé sur les modèles.....	56

Chapitre 4: Les Réseaux de Pétri

Figure 4.1: Les principes de base des réseaux de pétri.....	66
Figure 4.2: Définition formelle d'un Réseaux de Pétri.....	68
Figure 4.3: Exemple de marquage initiale.....	68
Figure 4.4: Evolution d'un réseau de Pétri.....	70
Figure 4.5: Parallélisme.....	74
Figure 4.6: Synchronisation mutuelle.....	75
Figure 4.7: Partage de ressource.....	76
Figure 4.8: Un réseau objet contenu dans un réseau système.....	80
Figure 4.9: Les réseaux système et objet après tirage de la transition.....	80

Chapitre 5: Test basé modèle des agents mobiles utilisant le paradigme des réseaux dans les réseaux

Figure 5.1: Système multi-agents modélisé avec les réseaux dans les réseaux.....	86
Figure 5.2: Concrétisation des cas de test abstrait (travail de l'agent testeur).....	93

Chapitre 6: Etude de cas et implémentation

Figure 6.1: Le problème du transport de ressources par des robots.....	96
Figure 6.2: Le réseau système du problème de transport de ressources.....	97
Figure 6.3: La structure d'une plateforme.....	98
Figure 6.4: La structure d'un agent.....	99
Figure 6.5: La structure de l'implémentation interne.....	100
Figure 6.6: Les protocoles prendre, déposer et décision de direction.....	102
Figure 6.7: Une simulation de réseau système du problème de transport de ressources dans renew.....	104
Figure 6.8: Trace de simulation	105
Figure 6.9: Implémentation sous test.....	105
Figure 6.10: Analyse statique.....	106
Figure 6.11: Instrumentation.....	107
Figure 6.12: La concrétisation des cas de test abstraits.....	108
Figure 6.13: Application de l'agent testeur sur le problème de transport	109
Figure 6.14: Première fenêtre de notre outil de test.....	111
Figure 6.15: Implémentation du problème de transport de ressources.....	111
Figure 6.16: Génération des cas de test abstraits.....	112
Figure 6.17: Analyse statique (mots-clé).....	113
Figure 6.18: Analyse statique (remplir le fichier d'informations d'agents).....	113
Figure 6.19: Instrumentation.....	114
Figure 6.20: Début de la concrétisation (prologue).....	115
Figure 6.21: Nouveau robot.....	115
Figure 6.22: Valeur juste pour la variable "but"	116
Figure 6.23: Changement du code (L'insertion d'une erreur).....	116
Figure 6.24: Erreur détectée par l'agent testeur.....	117
Figure 6.25: Résultats de la concrétisation.....	118

Liste des tableaux

Chapitre 2: Les Systèmes multi-agents

Tableau 2.1: Classification des situations d'interactions	27
Tableau 2.2 : Les paramètres d'un message KQML.....	33
Tableau 2.3: Liste des performatives de FIPA-ACL.....	35

Chapitre 4: Les Réseaux de Pétri

Tableau 4.1: Quelques interprétations typiques de transitions et de places.....	67
Tableau 4.2: Les types de mobilité.....	81

Chapitre 5: Test basé modèle des agents mobiles utilisant le paradigme des réseaux dans les réseaux

Tableau 5.1: La forme d'une étape de simulation (cas de test abstrait).....	87
--	----

Chapitre 1: Introduction générale

La présence de fautes introduites lors du développement d'un dispositif programmé est malheureusement courante. Elle doit être considérée avec beaucoup d'attention car notre vie de tous les jours dépend de plus en plus des systèmes logiciels et de lourdes pertes économiques, voire des issues fatales, peuvent être des conséquences de ces fautes. La validation des systèmes programmés avant leurs mises en service est essentielle et doit tenter d'atteindre un niveau de confiance satisfaisant. Ceci nécessite, de la part des opérateurs économiques, des académiciens et des chercheurs, le développement et l'amélioration de méthodes et d'outils adéquats pour la validation des systèmes logiciels.

Alors qu'il y a quelques décennies, un logiciel représentait quelques milliers de lignes de code, aujourd'hui ces produits comportent couramment plusieurs millions. Naturellement cette complexité entraîne une quantité plus importante d'erreurs lors du développement et en particulier lors de la programmation.

Le test de logiciel est une phase de développement de logiciel, destinée à évaluer et à augmenter la qualité du produit en détectant des erreurs et des anomalies. Le test de logiciel est une activité dans laquelle un système est exécuté sous des conditions spécifiques, les résultats sont observés ou enregistrés et comparés à des caractéristiques ou des résultats prévus.

Un système de multi-agents (SMA) est un contexte informatique dans lequel les agents agissent les uns avec les autres, d'une façon de collaboration ou concurrentielle, et parfois d'une façon autonome essayant d'atteindre leurs différents buts, accédant à des ressources et produisant de temps en temps des résultats. Les agents agissent d'une façon concurrent, asynchrone et décentralisé donc les systèmes multi-agents sont des systèmes complexes. En conséquence, il est difficile de les corriger et les tester.

Le test des SMA est une tâche provocante parce que ces systèmes sont distribués, autonomes, et délibératifs. Ces caractéristiques très particulières des agents logiciels rendent l'application des méthodes de test existantes difficile. Il y a des questions au sujet de communication et d'interopérabilité sémantique, aussi bien que la coordination. Tous ces dispositifs sont difficiles non seulement à modéliser et à programmer, mais à tester aussi.

Les agents sont une technologie prometteuse pour traiter le développement de système de plus en plus complexe. Un agent peut avoir beaucoup de manières de réaliser une tâche

donnée, et il choisit la manière la plus appropriée de traitement. Un agent mobile est une classe particulière d'agent avec la possibilité pendant l'exécution de déplacer d'une place à l'autre où il peut reprendre son exécution. Cette mobilité représente de nouveaux défis pour les systèmes dynamiques en toutes les phases du cycle de vie, comme la modélisation, l'implémentation et le test.

Il existe plusieurs méthodes et techniques pour le test des SMA, Parmi ces techniques, nous trouvons le test basé sur les modèles, il est basé sur un modèle de système afin de produire des cas de test abstraits. Pour que ces derniers puissent être soumis au système sous test, les cas de test abstraits doivent être transformés en des cas de test concrets.

L'approche de test basé modèle des agents mobiles est une approche fondée sur un modèle appelé réseau de référence. Les réseaux de référence sont basés sur le paradigme des réseaux dans les réseaux (en anglais nets within nets) qui généralise la notion de jeton classique à des jetons qui peuvent symboliser une structure de donnée complexe voire un réseau de Pétri. La génération des cas de test abstraits au niveau de cette approche s'appuie fortement sur les résultats de la simulation produits par un outil appelé Renew. Cette approche a besoin d'un algorithme de concrétisation.

La problématique principale de ce mémoire est la concrétisation au niveau de cette approche, plus particulièrement, ce travail s'intéresse à proposer un algorithme précis de concrétisation, et à le valider on l'appliquant sur un cas d'étude, dans ce cas le problème de transport de ressources par des robots est choisi.

Ce mémoire est organisé comme suit :

Le premier chapitre est une introduction générale à travers laquelle on a expliqué les domaines inclus dans notre travail.

Dans le deuxième chapitre, une vue générale est faite sur les systèmes multi-agents, les façons de communication et d'interaction entre les agents ainsi que les différentes plateformes et outils utilisés pour les développer.

Le troisième chapitre présente le test des logiciels en générale, et le test des systèmes multi-agents en particuliers.

Dans le quatrième chapitre, les réseaux de Pétri sont présentés comme méthode de modélisation, le chapitre commence par un ensemble de définitions des différentes notions

liés aux réseaux de Pétri, ensuite les types de réseaux de Pétri en générale et les réseaux de référence en particulier sont présentés.

Dans le cinquième chapitre on a proposé l'algorithme de concrétisation. Le sixième chapitre est une étude de cas à travers laquelle on a validé notre algorithme. On termine avec une conclusion de ce qui est fait, et des perspectives pour de nouveaux travaux.

Chapitre 2: Les systèmes multi-agents

L'Intelligence Artificielle (IA) est apparue au début des années 60. L'objectif de cette nouvelle discipline est d'aborder des problèmes complexes par la modélisation et la simulation de comportements dits intelligents. L'évolution de l'application de l'IA dans des domaines, en particulier celui de l'aide à la décision, montre les limites de son approche classique qui s'appuie sur une centralisation de l'expertise au sein d'un système unique.

Les travaux menés au début des années 70 sur la concurrence et la distribution ont contribué à la naissance d'une nouvelle discipline nommée l'intelligence artificielle distribuée (IAD). Elle a pour but de remédier aux insuffisances de l'approche classique de l'IA en proposant la distribution de l'expertise sur un groupe d'entités devant être capables de travailler en groupe et d'agir dans un environnement commun afin de résoudre les conflits éventuels. Les Systèmes Multi-Agents (SMA) sont une des branches issue de l'IAD, ils traitent le comportement d'un ensemble d'agents autonomes qui essaient de résoudre un problème commun.

Dans ce chapitre, les concepts qui découlent du domaine de recherche que constituent les SMA sont décrits. L'intérêt premier est porté d'abord sur le concept de "l'agent" qui constitue la brique fondamentale du domaine.

2.1 La notion d'agent

2.1.1 Définition

Plusieurs chercheurs, ont donné différentes définitions du terme agent. J Ferber définit un agent comme suit [1]: On appelle agent une entité physique ou virtuelle

- a. Qui est capable d'agir dans un environnement,
- b. Qui peut communiquer directement avec d'autres agents,
- c. Qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),
- d. Qui possède des ressources propres,
- e. Qui est capable de percevoir (mais de manière limitée) son environnement,

- f. Qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
- g. Qui possède des compétences et offre des services,
- h. Qui peut éventuellement se reproduire,
- i. Dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.

Donc un agent est une entité physique agit dans le monde réel. Un robot, un avion une voiture sont des exemples d'entités physiques. En revanche, un composant logiciel, un module informatique sont des entités virtuelles, car elles n'existent pas physiquement.

Les agents sont capables d'agir, et non pas seulement de raisonner comme dans les systèmes d'IA classique. L'action, qui est un concept fondamental pour les systèmes multi-agents, repose sur le fait que les agents accomplissent des actions qui vont modifier l'environnement des agents et donc leurs prises de décision futures. Ils peuvent aussi communiquer entre eux.

Les agents sont doués d'autonomie. Cela signifie qu'ils ne sont pas dirigés par des commandes venant de l'utilisateur (ou d'un autre agent), mais par un ensemble de tendances qui peuvent prendre la forme de buts individuels à satisfaire ou de fonctions de satisfaction ou de survie que l'agent cherche à optimiser. Il a la possibilité de répondre par l'affirmative ou le refus à des requêtes provenant des autres agents. Mais l'autonomie n'est pas seulement comportementale, elle porte aussi sur les ressources. Pour agir, l'agent a besoin d'un certain nombre de ressources: énergie, CPU (Central Processing Unit ou Unité centrale de traitement), quantité de mémoire, accès à certaines sources d'informations, etc. Ces ressources sont à la fois ce qui rend l'agent non seulement dépendant de son environnement, mais aussi, en étant capable de gérer ces ressources, ce qui lui donne une certaine indépendance vis-à-vis de lui. Les agents n'ont qu'une représentation partielle de leur environnement, c'est à dire qu'ils n'ont pas de vision globale de tout ce qui se passe.

L'agent est ainsi une sorte "d'organisme vivant" dont le comportement, qui se résume à communiquer, à agir et, éventuellement, à se reproduire, vise à la satisfaction de ses besoins et de ses objectifs à partir de tous les autres éléments (perceptions, représentations, actions, communications et ressources) dont il dispose.

2.1.2 Les caractéristiques multidimensionnelles d'un agent

Plusieurs propriétés sont reliées aux agents mais un agent ne possède pas forcément toutes ces propriétés. Celles qui sont les plus énoncées dans la littérature sont [1]:

- La nature : agents physiques ou virtuels.
- L'autonomie : l'agent est plus ou moins indépendant de l'utilisateur, des autres agents, et des ressources (U.C, mémoire, etc....)
- L'environnement : c'est l'espace dans lequel l'agent va agir ; celui-ci peut se réduire au réseau constitué par l'ensemble des agents.
- La capacité représentationnelle : l'agent peut avoir une vision très locale de son environnement, mais il peut aussi avoir une représentation plus large de cet environnement et notamment des agents qui l'entourent (accointances).
- L'objectif (dimension téléonomique) : l'agent peut poursuivre le but global de système, peut satisfaire des objectifs propres ou même se comporter dans la perspective de s'absoudre une fonction de survie.
- La perception : La perception est un moyen de recevoir de l'information d'un environnement qui est souvent complexe et évolutif.
- La communication : l'agent aura plus ou moins de capacités à communiquer avec les autres agents.
- Le raisonnement : l'agent peut être lié à un système expert ou à d'autres mécanismes de raisonnements plus ou moins complexes.
- La quantité de ses congénères : le système peut contenir de quelques-uns à plusieurs milliers d'agents.
- Le contrôle : il peut-être totalement distribué entre les agents mais peut être voué à une certaine classe d'agents comme les agents « facilitateurs ».
- L'anticipation : l'agent peut plus ou moins avoir les capacités d'anticiper les événements futurs.
- La granularité ou complexité : l'agent peut être très simple comme un neurone mais aussi plus complexe.

- La contribution : l'agent participe plus ou moins à la résolution du problème ou à l'activité globale du système.
- L'efficacité : l'agent et sa rapidité d'exécution, d'intervention.
- La bienveillance : l'agent a plus ou moins le devoir d'aider ses congénères plutôt que de s'opposer à eux.
- Intentionnalité : Un agent intentionnel est un agent guidé par ses buts. Une intention [2] est la déclaration explicite des buts et des moyens d'y parvenir. Elle exprime donc la volonté d'un agent d'atteindre un but ou d'effectuer une action.
- Rationalité : Un agent rationnel est un agent qui suit le principe suivant (dit principe de rationalité) [3] : « Si un agent sait qu'une de ses actions lui permet d'atteindre un de ses buts, il la sélectionne ». Les agents rationnels disposent de critères d'évaluation de leurs actions, et sélectionnent selon ces critères les meilleures actions qui leur permettent d'atteindre le but. De tels agents sont capables de justifier leurs décisions.
- Adaptabilité : un agent adaptatif est un agent capable de modifier son comportement et ses objectifs en fonction de ses interactions avec son environnement et avec les autres agents [4].
- Engagement : La notion d'engagement [5] est l'une des qualités essentielles des agents coopératifs. Un agent coopératif planifie ses actions par coordination et négociation avec les autres agents. En construisant un plan pour atteindre un but, l'agent se donne les moyens d'y parvenir et donc s'engage à accomplir les actions qui satisfont ce but ; l'agent croit qu'il a élaboré, ce qui le conduit à agir en conséquence.
- Flexibilité : la flexibilité peut être vue comme une forme de l'intelligence. Etre flexible signifie que l'agent est [6] :
 - Réactif : l'agent doit être capable de percevoir son l'environnement et de répondre à temps aux changements qui peuvent affecter cet environnement.
 - Proactif : l'agent n'agit pas seulement d'une manière réactive (en fonction de son environnement) mais il doit avoir un comportement orienté objectif et que l'agent peut prendre des initiatives.
 - Social : l'agent est capable d'interagir avec les autres agents intelligents et humains pour qu'il puisse atteindre ses propres objectifs et aider les autres dans leurs activités.

- Intelligence : il s'agit d'agents ayant la capacité de raisonner, d'apprendre et de s'adapter dans le temps. Un agent est intelligent s'il est capable de réaliser des actions flexibles et autonomes pour atteindre les objectifs qui lui ont été fixés.

2.1.3 Différentes catégories d'agents

De manière générale, il est possible de classer un agent selon les trois principaux points suivants [7]:

- La granularité : agent réactif, agent cognitif et agent hybrides.
- La fonctionnalité : agents collaboratifs, agents d'interface, agents pour la recherche d'informations, agents pour le commerce électronique, agents conversationnels animés et agents guide ou assistants
- La mobilité : agent mobile et agent stationnaire.

2.1.3.1 Selon la granularité

Il existe trois types d'agents : réactifs, cognitifs et hybrides. Ces trois types d'agents se distinguent par la granularité de leurs connaissances. Un agent hybride combine les propriétés d'un agent réactif et d'un agent cognitif. Selon Ferber [1], la distinction cognitif/réactif dépend à la capacité des agents à accomplir individuellement des tâches complexes et à planifier leurs actions.

Agent réactif

Les agents réactifs sont de plus faible granularité. Ils ne disposent pas de modèle des autres agents et leur mécanisme de raisonnement est primitif (réactif), souvent du type stimulus-réponse [1]. Ils réagissent uniquement à leur perception de l'environnement (figure2.1) et agissent en fonction de celle-ci [8].

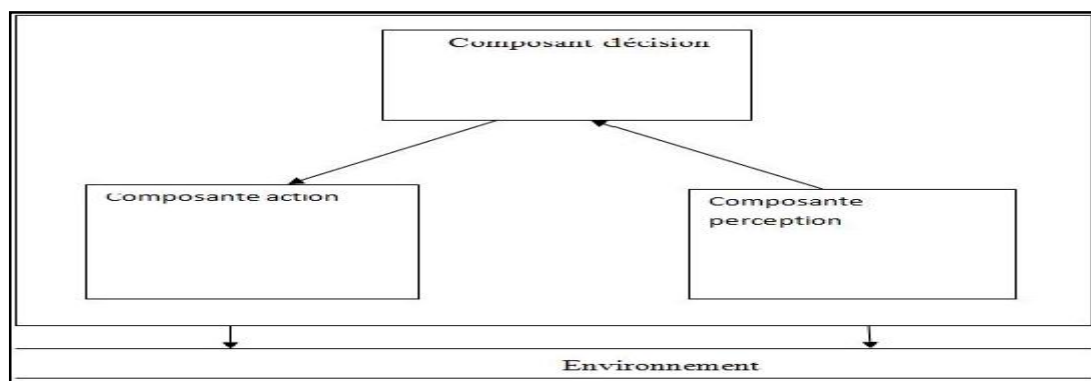


Figure 2.1: Structure générale d'un agent réactif [9]

Les défenseurs de ce type d'agent partent du principe suivant. Dans un système multi-agents, il n'est pas nécessaire que chaque agent soit individuellement "intelligent" pour parvenir à un comportement global intelligent (notion d'émergence). En effet, des mécanismes simples de réactions aux évènements peuvent faire émerger des comportements correspondants aux objectifs poursuivis. Cette approche propose la coopération d'agents de faible granularité (fine grain) mais beaucoup plus nombreux. Les agents réactifs sont de plus bas niveau, ils ne disposent que d'un protocole et d'un langage de communication réduite. Leurs capacités répondent uniquement à la loi stimulus-réponse. Un agent réactif ne dispose pas d'un mécanisme de mémorisation de ses expériences. Il ignore ses expériences car il ne dispose pas d'un processus lui permettant de planifier ou d'apprendre.

Agent cognitif

Les agents cognitifs sont caractérisés par une granularité forte ou moyenne, un modèle de représentation de soi et des autres et une plus grande capacité de raisonnement. De tels agents sont capables de percevoir et d'agir sur leur environnement (figure2.2). Ainsi, ils ont des capacités de cognition leur permettant de raisonner sur les autres et sur la résolution des problèmes [10].

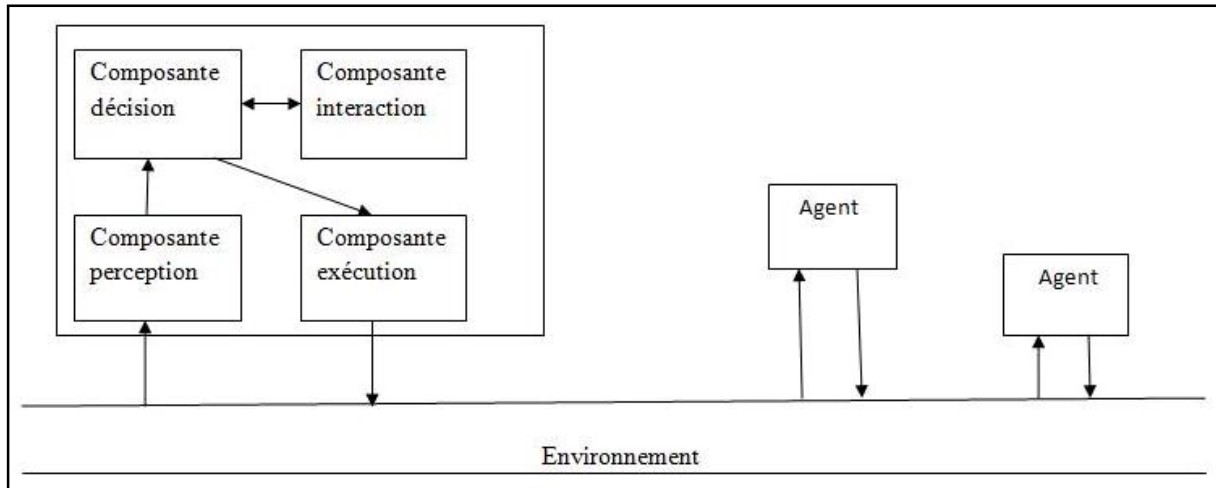


Figure 2.2: Structure d'un agent cognitif [9]

Les systèmes d'agents cognitifs sont fondés sur la coopération d'agents capables à eux seuls d'effectuer des opérations complexes. Ce type de système s'inspire du comportement humain. Un système cognitif comprend, en général, un petit nombre d'agents disposant d'une capacité de raisonnement sur une base de connaissances, d'une aptitude à traiter des informations diverses liées au domaine d'application et d'autres relatives à la gestion des interactions avec les autres agents et l'environnement [11]. Chaque agent est assimilable,

suivant le niveau de ses capacités, à un système expert plus ou moins sophistiqué. On parle d'agent de forte granularité "coarse grain".

Agent hybride

Un agent hybride est un agent combinant les propriétés des agents réactifs et des agents cognitifs. Pour surmonter les faiblesses des systèmes réactifs et des systèmes cognitifs, par rapport à la difficulté de mise en œuvre des systèmes cognitifs dans des environnements complexes et le manque de modèles formels dans les systèmes réactifs, plusieurs chercheurs se sont intéressés aux systèmes hybrides. Ils consistent à maintenir une certaine réactivité d'un agent en le dotant de composants réactifs et de rendre les autres composants cognitifs pour garantir un raisonnement de qualité [12], [13].

2.1.3.2 Selon la fonctionnalité

Agents collaboratifs

Ces agents ont des capacités de coopération. Un regroupement de ces agents permet, entre autres, de réduire un problème complexe en sous problèmes moins complexes [14].

Agents d'interface

Ces agents collaborent avec l'utilisateur pour effectuer certaines tâches telles que les activités de bureautique, les systèmes tuteurs intelligents qui doivent faciliter l'apprentissage et diminuer la surcharge cognitive chez l'apprenant [14].

Agents pour la recherche d'informations

Ces agents effectuent, en premier lieu, une recherche d'informations parmi une collection de données et, en second lieu, procèdent à une analyse des informations utiles trouvées afin de découvrir de nouvelles connaissances [14].

Agents pour le commerce électronique

La montée de l'internet a bien entendu créer de nouvelles nécessités. Les agents issus de cette tendance permettent la promotion, la vente ainsi que l'achat de produits et de services par l'entremise des réseaux informatiques, [14] etc.

Agents conversationnels animés

Ce sont des interfaces de dialogue entre des utilisateurs et des systèmes d'information. Ils se déploient sur des sites Internet, notamment des sites marchands. Ils sont pourvus de bases de dialogues correspondant aux contextes d'interaction dans lesquels ils agissent [14]

Agents guides ou assistants

Ce type d'agents essaye de suggérer des sites susceptibles d'intéresser l'utilisateur, en observant son comportement sur Internet. Le principe est assez simple. Généralement les sites que l'on visite sur Internet reflètent les goûts ou les besoins de l'utilisateur. Ainsi, en analysant ses habitudes de navigation, un « agent assistant » tente d'apprendre des habitudes de son utilisateur et de lui suggérer des sites en relation avec ce qu'il désire [14].

2.1.3.3 Selon la mobilité

La mobilité est une propriété non obligatoire des agents. En effet, tous les agents ne sont pas mobiles.

Agent stationnaire ou statique

L'agent stationnaire s'exécute seulement sur le système où il est créé. Si jamais il a besoin d'informations supplémentaires inexistantes localement ou d'interagir avec d'autres agents dans différents systèmes, alors il utilise typiquement le mécanisme RPC (Remote Procedure Calling) qui est une méthode appliquée dans le modèle client serveur [15].

Agents mobiles

Un agent mobile est un agent capable de se déplacer dans son environnement, qui peut être physique (réel ou simulé) ou structurel (niveaux d'exécution par exemple) [16].

2.2 Système multi-agents**2.2.1 Définition**

Parmi les différentes définitions des systèmes multi-agents, celle de Ferber est la plus détaillée, il le définit comme un système composé des éléments suivants [1] (voir figure 2.3):

- a. Un environnement E , c'est-à-dire un espace disposant généralement d'une métrique.
- b. Un ensemble d'objets 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. Ils peuvent être perçus, créés, détruits et modifiés par les agents.
- c. Un ensemble A d'agents, qui sont des objets particuliers (A inclut dans O), lesquels représentent les entités actives du système.
- d. Un ensemble de relations R qui unissent des objets (et donc des agents) entre eux.

- e. Un ensemble d'opérateurs Op permettant aux agents de A de percevoir, de produire, de consommer, de transformer et de manipuler des objets de O.
- f. Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appelle les lois de l'univers.

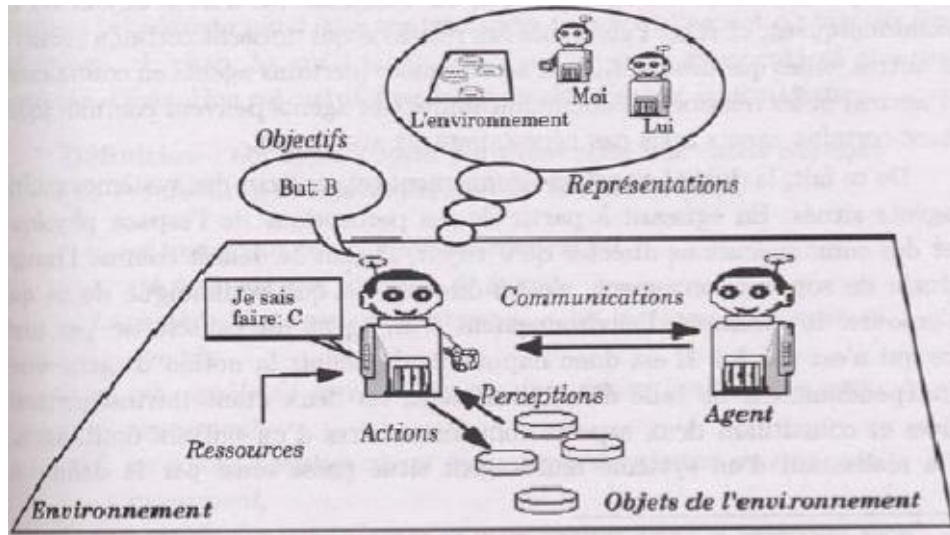


Figure 2.3: Représentation imagée d'un agent en interaction avec son environnement et les autres agents [1]

2.2.2 Domaines d'application

L'approche multi-agents est une véritable révolution dans la conception des systèmes. C'est la raison pour laquelle les domaines d'application sont nombreux. Citons par exemples: La résolution de problèmes au sens large, La robotique distribuée, La simulation multi-agents [1]:

- La résolution de problèmes au sens large: elle concerne en fait toutes les situations dans lesquelles des agents logiciels accomplissent des tâches utiles aux êtres humains. Cette catégorie s'oppose aux applications de robotique distribuée en ce sens que les agents sont purement informatiques et n'ont pas de structure physique réelle.
- La robotique distribuée: La robotique distribuée porte sur la réalisation non pas d'un seul robot, mais d'un ensemble de robots qui coopèrent pour accomplir une mission. la robotique distribuée utilise des agents concrets qui se déplacent dans un environnement réel.
- La simulation multi-agents: elle est fondée sur l'idée qu'il est possible de représenter sous forme informatique le comportement des entités qui agissent dans le monde et

qu'il est ainsi possible de représenter un phénomène comme le fruit des interactions d'un ensemble d'agents disposant de leur propre autonomie opératoire.

2.2.3 Exemples d'un SMA

Les systèmes multi-agents associés à l'intelligence artificielle représentent actuellement un grand domaine d'application et de recherche. Plusieurs systèmes ont été développés, nous présenterons ici quelques uns tels que [17]:

- Le système MANTA [18] : ce système illustre parfaitement l'intérêt de la modélisation multi-agents de type réactif. Il modélise la constitution d'une fourmilière mature à partir d'un ou plusieurs reines, étudie la capacité d'adaptation d'une telle colonie, le mécanisme de polyéthisme (division du travail), et la spécialisation des ouvrières. Cette simulation avait vérifié le fait qu'une société d'agents peut bien survivre et s'organiser en se passant de tout système de contrôle centralisé et d'une quelconque organisation hiérarchique.
- Les systèmes industriels distribués : où les concepteurs partent de problèmes réels existants et ils cherchent à les résoudre en se basant sur les techniques d'interaction et de coopération des systèmes multi-agents. Plusieurs systèmes ont été développés dans les domaines de la télécommunication, et de contrôle du trafic aérien.
- Applications temps réel : Les agents ont été bien évidemment appliqués au domaine des systèmes temps réel; ce dernier maintient des systèmes à contrainte souple. On voit de plus en plus des systèmes temps réels dit Hard utilisant des agents. Exemples: le système GUARDIAN [19], il a pour but de gérer les soins des patients d'une unité de soins intensifs chirurgicale.
- Applications agents pour le commerce électronique : Le E-Commerce signifie des échanges de produits qui se passent via Internet. Les sites pour les ventes aux enchères, pour les négociations entre les utilisateurs (producteurs/consommateurs), etc.
- Applications agents pour la Recherche d'Information : Une grande partie des applications de système multi-agents est dans le domaine de recherche d'information. Parmi ces nombreuses applications dans ce domaine, on peut trouver "NetSA" : une architecture de système multi-agents pour la recherche d'information dans des sources hétérogènes et réparties.

2.2.4 Les avantages des systèmes multi-agents

Les systèmes multi-agents sont des systèmes idéaux pour représenter des problèmes possédant de multiples méthodes de résolution, de multiples perspectives [20]. Ces systèmes possèdent les avantages traditionnels de la résolution distribuée et concurrente de problèmes comme la modularité, la vitesse (avec le parallélisme), et la fiabilité (due à la redondance). Ils héritent aussi des bénéfices envisageables de l'intelligence artificielle comme le traitement symbolique (au niveau des connaissances), la facilité de maintenance, la réutilisation et la portabilité mais surtout, ils ont l'avantage de faire intervenir des schémas d'interaction sophistiqués. Les types courants d'interaction incluent la coopération (travailler ensemble à la résolution d'un but commun) ; la coordination (organiser la résolution d'un problème de telle sorte que les interactions nuisibles soient évitées ou que les interactions bénéfiques soient exploitées) ; et la négociation (parvenir à un accord acceptable pour toutes les parties concernées) [17].

2.2.5 Les défis et les contraintes des SMA

Bien que les S.M.A offrent de nombreux avantages potentiels, ils doivent aussi relever beaucoup de défis [21]:

- Comment formuler, décrire, décomposer, et allouer les problèmes et synthétiser les résultats?
- Comment permettre aux agents de communiquer et d'interagir? Quoi et quand communiquer?
- Comment assurer que les agents agissent de manière cohérente : en prenant leurs décisions ou actions, en gérant les effets non locaux de leurs décisions locales ou en évitant les interactions nuisibles?
- Comment permettre aux agents individuels de représenter et raisonner sur les actions, plans et connaissances des autres agents afin de se coordonner avec eux? Comment raisonner sur l'état de leurs processus coordonnés (comme l'initialisation ou la terminaison)?
- Comment reconnaître et réconcilier les points de vue disparates et les intentions conflictuelles dans un ensemble d'agents essayant de coordonner leurs actions?
- Comment trouver le meilleur compromis entre le traitement local au niveau d'un seul agent et le traitement distribué entre plusieurs agents (traitement distribué qui induit la

communication)? Plus généralement, comment gérer la répartition des ressources limitées?

- Comment éviter ou amoindrir un comportement nuisible du système global, comme les comportements chaotiques ou oscillatoires?
- Comment concevoir les plates-formes technologiques et les méthodologies de développement pour les S.M.A?

2.2.6 Différences entre S.M.A et Système orienté objet

Un S.M.A offre un puissant répertoire d'outils, de techniques, et de métaphores qui y ont le potentiel d'améliorer considérablement les systèmes logiciels. La technologie agent représente un nouveau paradigme de programmation similaire à la programmation orientée objet et qui d'ailleurs pourrait s'intituler programmation orientée agent. Il convient de ne pas confondre "agent" et "objet". Tout comme les agents, les objets encapsulent leur état interne (leurs données). Ils peuvent également poser des actions sur cet état par le biais de leurs méthodes et ils communiquent en s'envoyant des messages. À ce niveau, ils diffèrent des agents par leur degré d'autonomie. En effet, une méthode doit être invoquée par un autre objet pour pouvoir accomplir ses effets. Un agent, quant à lui, recevra une requête et décidera de son propre gré s'il doit poser ou non une action. Une seconde différence provient du caractère flexible du comportement d'un agent. Bien que certains disent qu'il est possible de bâtir un programme orienté-objet qui intègre ces caractéristiques, on doit également voir que le modèle standard d'un objet ne dit rien à propos de ces types de comportements. La troisième et dernière différence provient du fait qu'on considère un agent comme étant lui-même une source de contrôle au sein du système tandis que dans un système orienté objet, on n'a qu'une seule source de contrôle [21].

2.2.7 Interaction entre agents

D'après Ferber [1], un agent sans interaction avec d'autres agents n'est plus qu'un corps isolé, dépourvu de caractéristiques d'adaptation. Dans une telle éventualité, les SMA (pour peu qu'on puisse appeler ainsi un tel regroupement d'agents) présenteraient les mêmes lacunes que les systèmes experts sans toutefois en avoir les avantages.

Les interactions jouent donc un rôle très important dans les SMA. La capacité des agents à interagir entre eux signifie que les agents peuvent être affectés par d'autres agents pour atteindre leurs objectifs et exécuter leurs tâches [22]. Ferber définit l'interaction Comme "

une mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques"[1].

Un agent peut agir sur le monde qui l'environne, c'est-à-dire sur les agents présents dans son univers et sur l'environnement lui-même (réel ou simulé). Cette intervention peut prendre la forme d'une modification de l'état des autres agents qu'il côtoie, que ce soit au niveau de leurs connaissances (ex : envoi d'information) ou au niveau de leur activité (ex : demande d'engagement).

Plusieurs situations imposent l'interaction entre agents ; l'échange de données, la mise en commun des compétences pour résoudre un problème global. Le partage de ressources limitées, l'aide d'un agent par un autre,...etc. Ferber [1] identifie trois critères pour classer les principales situations d'interaction :

- Les buts compatibles/ contradictoires des agents: La première classification repose sur cette distinction des buts, d'une manière générale. On dira que des buts sont incompatibles si la satisfaction de l'un entraîne l'insatisfaction de l'autre, ou l'affecte négativement, la réciproque étant automatiquement vraie.
- Les compétences : Les compétences des agents ou leurs capacités par rapport aux tâches est une composante fondamentale des interactions entre agents, un agent est-il suffisamment compétent pour réaliser seul une tâche, ou a-t-il besoin des autres ?
- Les ressources utilisées : les ressources sont tous les éléments environnementaux et matériels utiles à la réalisation d'une action. Quand les ressources sont limitées et nécessaires pour par plus d'un agent en même temps une situation de conflit se produira.

Selon les trois critères mentionnés ci-dessus les situations possibles d'interaction entre agents sont illustrées dans le tableau 2.1 Les agents sont dans une situation d'indifférence ou de coopération si leurs buts sont compatibles et dans une situation d'antagonisme dans le cas contraire.

2.2.7.1 Situation d'indifférence

La situation 'indifférence ou d'indépendance ne pose aucun problème d'interaction de points de vue multi-agents et se résume à la simple juxtaposition des actions des agents pris indépendamment sans qu'il y ait effectivement d'interaction du fait qu'il n'existe aucun point de rencontre entre eux : aucun agent n'a besoin des autres agents pour atteindre son

objectif [1], les ressources sont suffisantes et leurs buts sont indépendants, dans telle situation chaque agent poursuit son propre agenda indépendamment des autres mais d'après Franklin [23] il peut aussi interagir avec les autres pour atteindre ces propres objectifs. Dans une situation d'indépendance, les agents peuvent même avoir les mêmes buts mais chacun d'entre eux travaille indépendamment des autres. Dans ces situations la satisfaction de l'un des buts des agents n'est pas affectée ni négativement ni positivement par celle de l'autre [1].

Buts	Ressources	Compétences	Types de situation	Remarques
Compatibles	Suffisantes	Suffisantes	<i>Indépendance</i>	Situation d'indifférence
Compatibles	Suffisantes	Insuffisantes	<i>Collaboration simple</i>	Situations de coopérations
Compatibles	Insuffisantes	Suffisantes	<i>Encombrement</i>	
Compatibles	Insuffisantes	Insuffisantes	<i>Collaboration coordonnée</i>	
Incompatibles	Suffisantes	Suffisantes	<i>Compétition individuelle pure</i>	Situations d'antagonismes
Incompatibles	Suffisantes	Insuffisantes	<i>Compétition collective pure</i>	
Incompatibles	Insuffisantes	Suffisantes	<i>Conflits individuels pour des ressources</i>	
Incompatibles	Insuffisantes	Insuffisantes	<i>Conflits collectifs pour des ressources</i>	

Tableau 2.1: Classification des situations d'interactions [1]

2.2.7.2 Situation de coopération

Contrairement à la situation d'indifférence, dans les situations coopératives la satisfaction de l'un des buts des agents est renforcée par la satisfaction de celle de l'autre [1]. On distingue trois situations possibles vis-à-vis les compétences des agents et les ressources utilisées : Collaboration simple, encombrement et collaboration coordonnée.

- Collaboration simple (ressources suffisantes, compétences insuffisantes): La collaboration simple consiste en une simple addition des compétences pour réaliser les tâches jugées irréalisables individuellement. Puisque les ressources sont suffisantes, cette situation ne nécessite pas d'actions supplémentaires de coordination entre les intervenants.
- Encombrement (ressources insuffisantes, compétences suffisantes): Les agents se gênent mutuellement dans l'accomplissement de leurs tâches, du fait de l'insuffisance des ressources. Puisque leurs buts sont compatibles, ils essaient de coordonner leurs actions pour vaincre l'encombrement.

- Collaboration coordonnée (ressources insuffisantes, compétences insuffisantes): Représente une situation où les agents doivent collaborer pour combler leur manque de compétences. En plus de la collaboration, les agents doivent coordonner leurs actions à cause de l'insuffisance des ressources. C'est la plus complexe des situations de collaboration.

2.2.7.3 Situations d'antagonisme

Dans les situations d'antagonisme, la satisfaction de l'un des buts des agents est affectée négativement par celle de l'autre [1], quatre situations sont possibles : compétition individuelle pure, conflit individuelle sur les ressources, compétition collective pure et conflit collective sur les ressources.

- Compétition individuelle pure (ressources suffisantes, compétences suffisantes): Les ressources ne sont pas limitées. L'accès aux ressources ne constitue pas l'enjeu du conflit. Les agents luttent pour atteindre leurs objectifs individuels.
- Conflit individuel pour les ressources (ressources insuffisantes, compétences suffisantes): Dans ce cas les agents luttent pour atteindre leurs objectifs individuels et luttent aussi sur les ressources limitées. Chaque agent essaye d'acquérir les ressources pour pouvoir atteindre ces buts. Cette situation pousse les agents à négocier l'accès aux ressources.
- Compétition collective pure (ressources suffisantes, compétences insuffisantes): Les agents n'ont pas des compétences suffisantes. Ils vont se regrouper au sein de coalitions ou d'associations pour parvenir à atteindre leurs objectifs, l'interaction entre les agents d'un même groupe est collaboration simple. Les collections d'agents (les groupes) entrent en compétitions pour atteindre leurs objectifs.
- Conflits collectifs pour des ressources (ressources insuffisantes, compétences insuffisantes): Dans ce cas les groupes d'agents formés à cause de l'insuffisance de compétences, en plus de leurs compétitions dues à l'incompatibilité de leurs buts, ils entrent en compétition sur les ressources vues qu'elles sont limitées.

2.2.8 Communication entre agents

2.2.8.1 Définition

La communication est l'un des modes principaux d'interaction existant entre les agents. [1]. En communiquant, les agents peuvent échanger des informations et coordonner leurs activités.

La communication est l'ensemble des processus physiques et psychologiques par lesquels s'effectue l'opération de mise en relation d'un ou plusieurs agents (l'émetteur) avec un ou plusieurs agents (le récepteur), en vue d'atteindre certains objectifs. [24], cette définition de la communication donnée par Didier Anzieu et Jacques-Yves Martin dans leur étude sur la dynamique des groupes restreints. Elle s'applique bien au contexte des systèmes multi-agents parce qu'elle attribue à la communication à la fois une dimension physique, une dimension psychologique et une dimension relationnelle (ou sociale). Dans le cadre des systèmes multi-agents, la dimension psychologique se réfère aux structures de connaissance qui représentent les constructions mentales de l'agent telles que l'action, l'intention et la croyance.[25]

2.2.8.2 Les modes de communication

Il existe plusieurs modes de communication : la communication indirecte via l'environnement, la communication par partage d'informations et la communication par envoi de messages.[14]

Communication indirecte via l'environnement

Dans ce mode de communication les agents se communiquent par signaux via l'environnement. Ces signaux, une fois interprétée, vont produire des effets sur les agents. Ce type de communication est spécifique aux agents réactifs.

Prenons l'exemple des fourmis, où chacune se déplace dans un espace à la recherche de nourriture. Chaque fourmi dépose sur son parcours des phéromones qui servent de marqueurs à son itinéraire. Alors il est possible aux fourmis d'optimiser la collecte de nourriture en analysant la concentration en phéromone et en choisissant le chemin le plus parcouru.

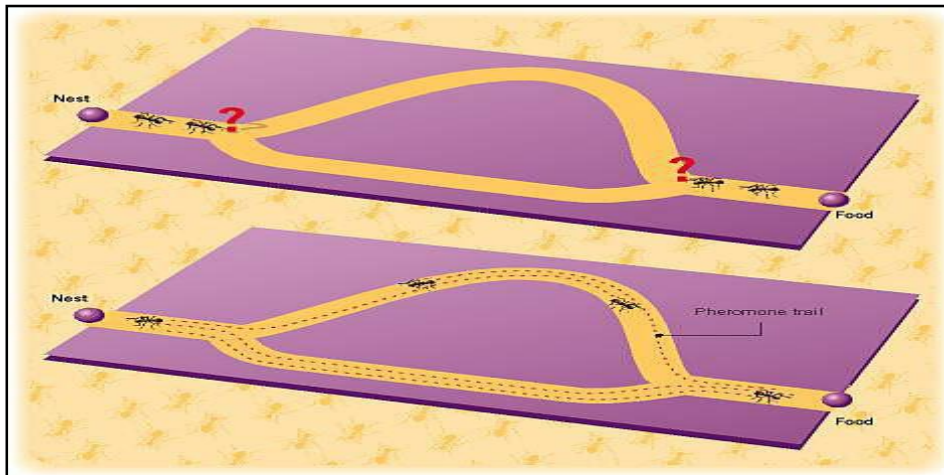


Figure 2.4: Communication indirecte via l'environnement [26]

Communication par partage d'informations

Les agents ne sont pas en liaison directe mais communiquent via une structure de données partagée, où on trouve les connaissances relatives à la résolution qui évolue durant le processus d'exécution. Cette manière de communiquer est l'une des plus utilisées dans la conception des systèmes multi-experts. L'exemple parfait d'utilisation de ce mode de communication est l'architecture de blackboard, on parle plutôt de sources de connaissances que d'agents. La communication par partage d'informations suppose l'existence d'une base partagée sur laquelle les composants viennent lire et écrire (voir figure 2.5)

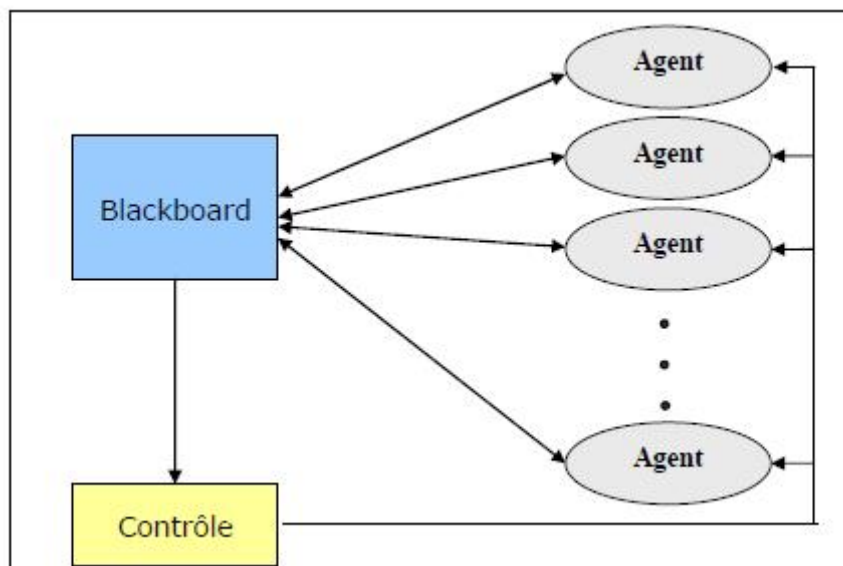


Figure 2.5: Communication par partage d'informations [17]

Communication par envoi de messages:

Les agents envoient leurs messages directement et explicitement au destinataire.

- Mode point à point : l'agent émetteur du message connaît et précise l'adresse de ou des agent(s) destinataire(s). Ce type de communication est généralement le plus employé par les agents cognitif.
- Mode par diffusion : le message est envoyé à tous les agents du système. Ce type de transmission est très utilisé dans les systèmes dynamiques ainsi que les systèmes d'agent réactif. En fait, ceci suppose en général une messagerie : un agent spécialisé gère autant de files d'attente que de destinataires, chaque agent peut traiter le premier message de sa file.

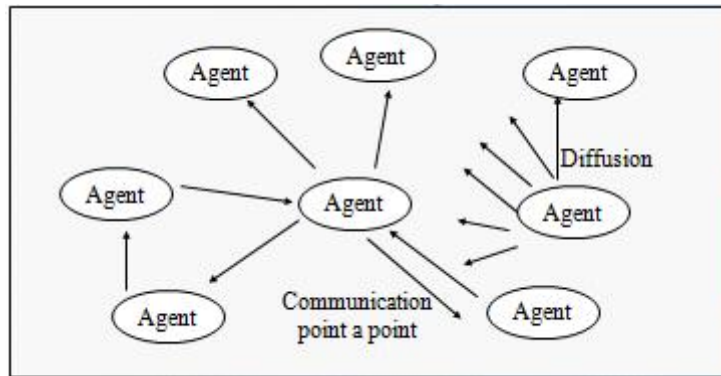


Figure 2.6: Communication par envoi de messages

2.2.9 Les langages de communication dans les SMA

Une communication peut correspondre à une information, une requête ou une interrogation ; elle doit avoir une sémantique reconnue par l'agent récepteur [11]. Pour réaliser la communication, un langage de communication est nécessaire à travers lequel les agents arrivent à échanger des messages.

2.2.9.1 Actes de langage

Un langage de communication est basé sur la théorie des actes de langages. Cette théorie a été introduite par le philosophe J-L. Austin [27]. Elle constitue un fondement théorique de la communication basé sur la constatation suivante : en parlant, on effectue des actes. Cela paraît clairement dans le cas des verbes appelés performatifs (comme promettre, ordonner, demander, etc.) où l'acte est effectué par l'énoncé même. Mais, on peut la généraliser aux autres verbes appelés constatifs (par exemple : il pleut devient je déclare qu'il pleut). J-L. Austin distingue trois types d'actes dans un énoncé [11] :

- Actes locutoires : activité mentale et physique de formulation et d'articulation de l'énoncé. L'acte réussit si l'énoncé est formulé et articulé correctement.
- Actes illocutoires : actes effectués par l'interlocuteur lors de la formulation de l'énoncé. En énonçant : il est dix heures? On peut comprendre s'il s'agit d'une affirmation ou d'une interrogation. Si on arrive à comprendre qu'il s'agit bien d'une phrase interrogative, alors l'acte illocutoire est réussi.
- Actes perlocutoires : c'est la conséquence indirecte mais pouvant être intentionnelle des actes locutoires et illocutoires sur l'auditeur. Ces actes dépendent du contexte, et contrairement aux actes illocutoires, ils ne sont pas codés dans l'énoncé. Par exemple à l'énoncé il fait froid est associé l'acte perlocutoire de mise en marche du chauffage.

Un acte de langage met en relation deux agents [28] : L'émetteur de l'acte qui doit savoir comment le générer et l'émettre et le récepteur de l'acte qui doit savoir comment le recevoir et l'interpréter.

Il existe un certain nombre de langages qui permettent la communication entre les agents dont on va parler sur deux qui sont le KQML et FIPA ACL.

2.2.9.2 Le langage KQML

Premier apparu parmi les ACLs (Agent communication Language), le langage KQML (Knowledge Query Manipulation Language), il a été développé pour échanger des informations et des connaissances entre des systèmes à base de connaissances. Il a été ensuite repris dans [29] pour décrire les messages échangés entre les agents.

KQML est à la fois un langage et un protocole permettant de structurer des messages afin de partager de l'information entre agents [30]. KQML est basé sur la théorie des actes de langage. Il propose une encapsulation des messages dans une performative qui définit l'acte illocutoire. Il est constitué d'un important nombre de performatifs qui sont en quelque sorte les opérations permises entre les agents. Les 36 performatifs de KQML peuvent être classifiés en trois grandes catégories :

- Les 18 performatives de discours : servent à échanger des connaissances et des informations présentes dans la base de connaissance de l'agent. Exemples: ask-if, ask-one, tell, describe, stream-all ...
- Les 11 performatives d'interconnexion : aide à la mise en relation des agents entre eux. Exemples: register, unregister, broadcast ...

- Les 7 performatives d'exception : servent à changer le déroulement normal des échanges. Exemples: error, sorry, standby ...

Le tableau 2.2 représente l'ensemble des paramètres que l'on peut retrouver à l'intérieur d'un message KQML ainsi que la signification de chacun d'entre eux. Selon le performatif considéré, tous ou seulement certains des attributs ci-dessus sont utilisés dans le message[31].

Mot-clé	Signification
:sender	L'expéditeur du message.
:receiver	Le récepteur du message.
:from	Indique l'origine du performatif se trouvant dans l'étiquette :content lorsque forward est utilisé.
:to	La destination finale du performatif se trouvant dans l'étiquette :content lorsque forward est utilisé.
:content	L'information que le performatif exprime.
:in-reply-to	L'étiquette de réponse à laquelle se réfère ce message.
:language	Le nom du langage utilisé pour représenter le contenu.
:ontology	L' (Les) ontologie(s) utilisée(s) qui donne(nt) la signification aux éléments du contenu.
:reply-with	L'étiquette qui doit être utilisée pour les réponses si l'expéditeur en désire une.

Tableau 2.2: Les paramètres d'un message KQML [30]

La figure 2.7 présente un exemple du message KQML, où l'agent A1 veut informer l'agent A2 que le bloc A est sur le bloc.

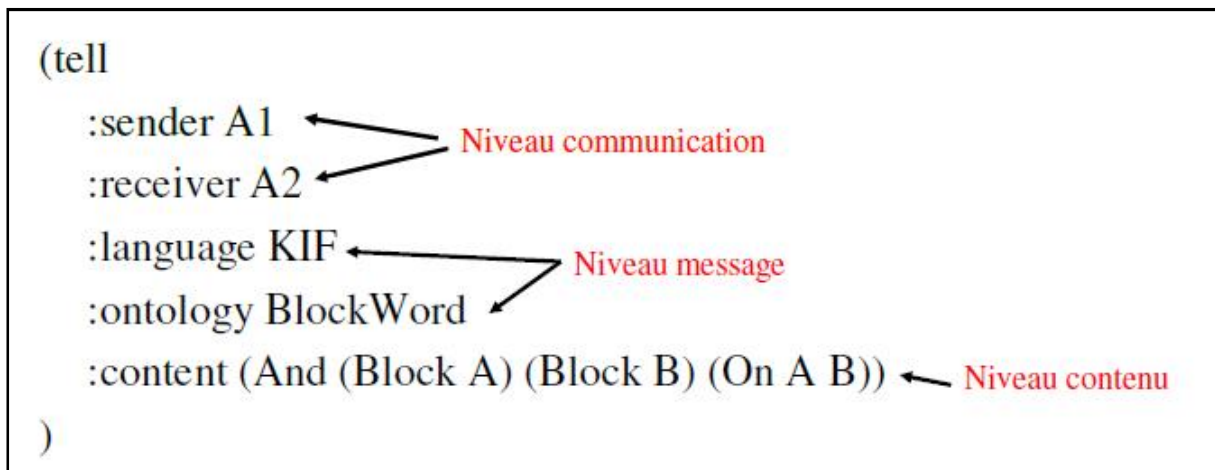


Figure 2.7: Un message KQML [32]

KQML du fait qu'il est le premier standard des ACLs présente certains avantages, ce qui faisait de lui, un langage fort utilisé et Beaucoup d'applications le supportent [30]. Toutefois, KQML présente également certains inconvénients. Parmi lesquels, on identifie l'Ambiguïté

dans certains performatifs qui ne sont pas clairs ou des performatifs manquants : Il s'agit sans doute de la critique la plus importante. [30].

2.2.9.3 Le langage ACL

La fondation pour les agents intelligents physiques (FIPA pour "Foundation for Physical Intelligent Agents") est un organisme international à but non lucratif regroupant diverses compagnies dont l'objectif est de fournir des standards pour faciliter l'interopérabilité entre les systèmes à base d'agents. Le Langage qui a résulté de ces efforts est FIPA-ACL [30]. Comme KQML, ce dernier est basé sur la théorie des actes de langage : les messages sont vus comme des actes communicatifs [33]. FIPA-ACL est très proche de KQML: leur syntaxe est similaire sauf pour un petit nombre de performatives dont le nom diffère [33].

Ce langage est fondé sur vingt actes communicatifs (tableau2.3), exprimés par des performatifs, qui peuvent être groupés selon leurs fonctionnalités de la façon suivante[32] :

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

performative	description
accept-proposal	<i>S</i> accepte une proposition d'action envoyée précédemment.
agree	<i>S</i> accepte de réaliser une action demandée.
cancel	<i>S</i> informe qu'il n'a plus besoin qu'un autre agent réalise une action demandée précédemment.
call-for-proposal	<i>S</i> fait un appel d'offre pour une action.
confirm	<i>S</i> informe qu'une proposition est vraie (ce type de message est envoyé quand <i>S</i> pense que son interlocuteur est incertain à propos de la valeur de la proposition).
disconfirm	<i>S</i> informe qu'une proposition est fausse.
failure	<i>S</i> informe qu'il a essayé de réaliser un action mais que cette tentative a échoué.
inform	<i>S</i> informe de la valeur d'un proposition.
inform-if	<i>S</i> informe qu'une proposition est vraie.
inform-ref	<i>S</i> informe <i>R</i> de la valeur d'un objet.
not-understood	<i>S</i> informe <i>R</i> qu'il n'a pas compris une des actions (éventuellement communicative) de <i>R</i> .
propose	<i>S</i> propose de réaliser une certaine action.
query-if	<i>S</i> demande si une proposition est vraie ou non.
query-ref	<i>S</i> demande la valeur d'un objet.
refuse	<i>S</i> refuse de réaliser une action.
reject-proposal	<i>S</i> refuse une proposition d'action.
request	<i>S</i> demande à <i>R</i> de réaliser une action.
request-when	<i>S</i> demande à <i>R</i> de réaliser une action quand une certaine proposition deviendra vraie.
request-whenever	<i>S</i> demande à <i>R</i> de réaliser une action à chaque fois qu'une certaine proposition deviendra vraie.
subscribe	<i>S</i> demande à <i>R</i> de le prévenir des changements de valeur d'un objet.

Tableau 2.3: Liste des performatives de FIPA-ACL [33]

Par exemple, dans la figure 2.8, l'agent A informe l'agent B du temps qu'il fera demain, selon ses prévisions à l'aide d'un message FIPA-ACL.

```
(inform
  :sender A
  :receiver B
  :content temps (demain, pleuvoir)
  :language Prolog
)
```

Figure 2.8: Un message FIPA-ACL [32]

Une sémantique pour des actes de langage individuels (ou performatif) n'est pas suffisante pour pleinement supporter l'interaction entre agents : le contenu de l'acte de langage est important, tout comme la capacité à formuler des dialogues (ou de protocoles) qui guident l'interaction entre agents. Traditionnellement, un protocole est une spécification de ces interactions. Il précise qui peut dire quoi à qui et les réactions possibles à ce qui est dit par l'autre partie. La fonction de base d'un protocole d'interaction est de fournir aux agents une façon de communiquer utilement sans avoir à planifier explicitement chaque acte de langage en se basant sur ses effets attendus. Quand nous utilisons un protocole d'interaction, nous supposons que le concepteur du protocole a effectué une analyse hors ligne suffisante et garantit ainsi que suivre le protocole permette d'atteindre les buts associés aux états finaux identifiés du protocole. Plus un protocole efficace, moins d'informations ont besoin d'être transmises et moins de temps est passé dans la communication [34]. Dans FIPA, il existe plusieurs protocoles : fipa-request, fipa-query, Contract Net Protocol (CNP)... , dans la suite en va présenter le protocole request puisque il est utilisé dans notre travail.

Avec FIPA-request (figure2.9), un agent sollicite un autre agent pour exécuter des actions et l'agent récepteur retourne soit une réponse favorable à l'exécution d'actions, soit une réponse défavorable expliquée par telle ou telle raison. Supposons que l'agent i aie besoin de l'agent j pour exécuter l'action "action".L'agent i envoie "request" à l'agent j:

- Si l'agent j accepte la requête, il retourne "agree". Ensuite, quand j a fini d'exécuter « action », il en informe i en utilisant "inform".
- Si l'agent j accepte mais rencontre un problème durant le traitement de « action », il retourne « failure » et les raisons de l'échec.
- Si l'agent j n'accepte pas la requête de l'agent i, j retourne "refuse" et les raisons de ce refus.

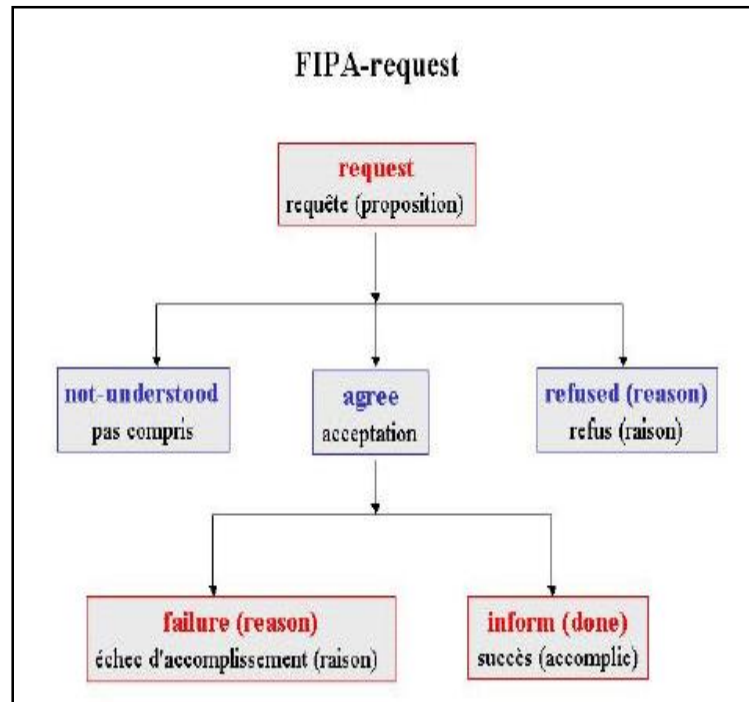


Figure 2.9: Le protocole request du FIPA [32]

2.2.10 Plateformes multi-agents

Afin de réaliser une opérationnalisation plus accessible des systèmes multi-agents, des travaux ont tenté de réutiliser des architectures et des langages existants pour construire des environnements de développement de ces systèmes. Les environnements de développement ou les plateformes multi-agents sont nécessaires pour renforcer le succès de la technologie multi-agents. Les plates-formes multi-agents permettent aux développeurs de concevoir et de réaliser leurs applications sans perdre de temps à réaliser des fonctions de base pour la création et l'interaction entre agents et éliminent, dans la plupart des cas, la nécessité d'être familier avec les différents concepts théoriques des systèmes multi-agents. Parmi les plateformes les plus connues on peut citer Swarm [35], Zeus [36], MadKit [37] et JADE [38], etc.

2.2.10.1 La plate-forme SWARM

SWARM (Minar e.a., 1996) est une plate-forme multi-agents avec agents réactifs. L'inspiration du modèle d'agent utilisé vient de la vie artificielle. SWARM est l'outil privilégié de la communauté américaine et des chercheurs en vie artificielle. L'environnement offre un ensemble de bibliothèques qui permettent l'implémentation des systèmes multi-agents avec un grand nombre d'agents simples qui interagissent dans le

même environnement. De nombreuses applications ont été développées à partir de SWARM qui existe aujourd'hui implémenter en plusieurs langages (Java, Objective-C).

2.2.10.2 La plate-forme ZEUS

ZEUS (Nwama e.a., 1999) est une plate-forme de développement de SMA générique, personnalisable et pourrait être augmentée par l'adjonction de nouveaux composants. Elle est présente sous forme d'un ensemble de classes implémentées dans le langage Java. En cela, elle est complètement portable et peut être utilisée sur tout système disposant d'une machine virtuelle Java. Un agent est composé de plusieurs objets et threads. Ces classes sont catégorisées dans trois groupes fonctionnels: une librairie de composants, un outil de développement et un outil de visualisation. De par sa genericité, elle peut être utilisée dans la conception d'un grand nombre d'applications mais en retour nécessitera des augmentations assez nombreuses et spécifiques à chaque type d'application.

2.2.10.3 La plate-forme MADKIT

MADKIT (Madkit, 2003) est une plate-forme développée par le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) de l'Université Montpellier II. MADKIT est libre pour l'utilisation dans l'éducation. Il est écrit en Java et est fondé sur le modèle organisationnel Alaadin. Il utilise un moteur d'exécution où chaque agent est construit en partant d'un micro-noyau. Chaque agent a un rôle et peut appartenir à un groupe. Il y a un environnement de développement graphique qui permet facilement la construction des applications.

2.2.10.4 La plate-forme JADE

JADE (Java Agent Development Framework - Bellifemine, Poggi, Rimassa, 1999) est une plate-forme multi-agents développée en Java par CSELT (Groupe de recherche de Gruppo Telecom, Italie) qui a comme but la construction des systèmes multi-agents et la réalisation d'applications conformes à la norme FIPA (FIPA, 1997). JADE comprend deux composantes de base : une plate-forme agents compatible FIPA et un paquet logiciel pour le développement des agents Java. Le but de JADE est de simplifier le développement des systèmes multi-agents en conformité avec la norme FIPA pour réaliser des systèmes multi-agents interopérables. Pour atteindre ce but, JADE offre la liste suivante de caractéristiques au programmeur d'agents :

- La plate-forme multi-agents compatible FIPA: qui inclut le Système de Gestion d'Agents (AMS), le Facilitateur d'Annuaire (DF), et le Canal de Communication entre

Agents (ACC). Ces trois agents sont automatiquement créés et activés quand la plate-forme est activée.

- La plate-forme d'agents distribuée: La plate-forme d'agents peut être distribuée sur plusieurs hôtes, à condition qu'il n'y ait pas de pare-feu entre ces hôtes. Une seule application Java, et donc une seule Machine Virtuelle Java, est exécutée sur chaque hôte. Les agents sont implémentés comme des threads d'exécution Java et les événements Java sont utilisés pour la communication efficace et légère entre agents sur un même hôte. Un agent peut exécuter des tâches parallèles et JADE planifie ces tâches d'une manière plus efficace (et même plus simple pour le programmeur) que la planification faite par la Machine Virtuelle Java pour les threads d'exécution.
- Un certain nombre de DF (Facilitateurs d'Annuaire) compatibles FIPA qui peuvent être activés quand on lance la plate-forme pour exécuter les applications multi-domaines, où la notion de domaine est la notion logique décrite par le document FIPA97 dans sa Partie 1.
- Une interface de programmation pour simplifier l'enregistrement de services d'agents avec un ou plusieurs domaines de type DF.
- Le mécanisme de transport et l'interface pour l'envoi et la réception des messages.
- Le protocole IIOP compatible avec le document FIPA97 pour connecter des plates-formes multi-agents différentes.
- Le transport léger de messages ACL sur la même plate-forme d'agents. Dans le but de simplifier la transmission, les messages internes (sur la même plateforme) sont transférés codés comme des objets Java et non comme des chaînes de caractères. Quand l'expéditeur ou le récepteur n'appartient pas à la même plate-forme, le message est automatiquement converti à/du format de chaîne de caractères spécifiés par la FIPA. De cette façon, la conversion est cachée au programmeur d'agents, qui a seulement besoin de traiter la classe d'objets Java.
- Une bibliothèque de protocoles d'interaction compatibles FIPA.
- L'enregistrement automatique d'agents dans le Système de Gestion d'Agents (AMS).
- Un service d'attribution de noms compatible FIPA : quand on lance la plate-forme, un agent obtient un identificateur unique (Globally Unique Identifier - GUID).

- Une interface graphique utilisateur pour gérer plusieurs agents et plates formes multi-agents en partant d'un agent unique. L'activité de chaque plateforme peut être supervisée et enregistrée.

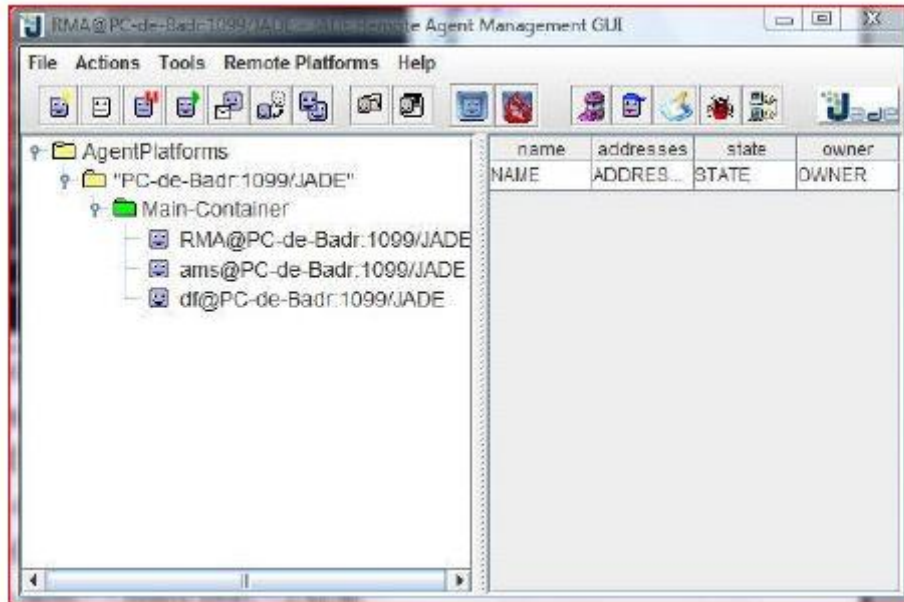


Figure 2.10: La fenêtre de JADE

2.3 Conclusion

Dans ce chapitre, nous avons représenté les concepts agent et le système multi-agents, Les SMA sont largement répandus et développés autour du monde, ils sont parmi les domaines les plus porteurs en Intelligence Artificielle. J. Ferber, précise que les SMA touchent à des domaines différents, variés et pas évidemment reliés directement aux SMA comme les sciences cognitives, la philosophie, la biologie, les théories d'organisation, la gestion des ressources humaines etc.

Les agents agissent d'une façon concourant, asynchrone et décentralisé donc les systèmes multi-agents sont des systèmes complexes. En conséquence, il est difficile de les corriger et les tester. Dans le chapitre suivant nous aborderons le test de logiciel en générale et en particulier le test des systèmes multi-agents.

Chapitre 3: Le test des systèmes multi-agents

Afin de savoir si un logiciel ne comporte pas d'anomalies et répond aux attentes des utilisateurs, les développeurs doivent le soumettre à des tests. Procéder aux tests de logiciels n'est pas un processus visant seulement à détecter les éventuelles anomalies, mais aussi à acquérir la confiance nécessaire avant l'utilisation opérationnelle du produit logiciel en vue d'atteindre la qualité voulue.

Les systèmes multi-agents ont besoin de techniques appropriées pour évaluer les comportements autonomes de l'agent aussi bien que les propriétés de distribution, sociales et délibératives, qui sont particulières à ces systèmes. Dans ce chapitre, nous présentons brièvement le test en général et le test des systèmes multi-agents en particulier.

3.1 Le test de logiciel

3.1.1 Qu'est-ce qu'un logiciel

Le terme logiciel est une traduction du terme anglais Software. Il constitue l'ensemble des programmes et des procédures nécessaires au fonctionnement d'un système informatique. Dans la famille des logiciels, on trouve par exemple des logiciels d'application qui sont spécifiques à la résolution des problèmes de l'utilisateur (progiciel, tableur, traitement de texte, grapheur, etc.), mais aussi des logiciels d'enseignement ou didacticiels, des logiciels de jeu ou ludiciel, etc. [39].

3.1.2 Qu'est ce qu'un test de logiciel

Il existe plusieurs définitions pour le test logiciel, nous citons les suivantes qui donnent une idée complète sur la notion de test.

- Le test est un processus manuel ou automatique, qui vise à établir qu'un système vérifie les propriétés exigées par sa spécification, ou à détecter des différences entre les résultats engendrés par le système et ceux qui sont attendus par la spécification [40].
- Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts [41].

- Le test est l'exécution ou l'évaluation d'un système ou d'un composant, par des moyens automatiques ou manuels, pour vérifier qu'il répond aux spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus [42].
- Le test est une technique de contrôle consistant à s'assurer, au moyen de son exécution que le comportement d'un programme est conforme à des données préétablies [43].
- Le test de logiciel est une activité de vérification. Son objectif est de détecter des fautes ou les inadéquations d'un logiciel. Il est nécessaire de préciser par rapport à quelles références on tente de dégager ces inadéquations les spécifications du logiciel, les normes ou règles portant sur son code ou les documents le concernant [44].
- Le test d'un logiciel est une activité qui fait partie du processus de développement. Il est mené selon les règles de l'assurance de la qualité et débute une fois que l'activité de programmation est terminée. Il s'intéresse aussi bien au code source qu'au comportement du logiciel. Son objectif consiste à minimiser les chances d'apparition d'une anomalie avec des moyens automatiques ou manuels qui visent à détecter aussi bien les diverses anomalies possibles que les éventuels défauts qui les provoqueraient » [45].

3.1.3 Terminologie liée au test

Pour cerner la notion de test, il convient de comprendre des notions que nous présentons ci-après:

- Spécification: En informatique, la spécification est un modèle d'un logiciel. C'est aussi l'étape en génie logiciel qui consiste à décrire ce que le logiciel doit faire. On distingue: Les spécifications informelles, par exemple un texte en français, Les spécifications semi-formelles avec une syntaxe plus précise ou comportant des diagrammes plus ou moins standardisés et Les spécifications formelles qui présentent une syntaxe et une sémantique. [46].
- Satisfaction: Un programme satisfait sa spécification lorsqu'il est en tout point conforme aux exigences de celle-ci [46].
- La vérification est « le processus d'évaluation d'un produit issu d'une des activités du développement logiciel, pour déterminer la correction et la cohérence avec les produits ou les normes fournies comme entrée de cette activité » [47].

La vérification sert à répondre à la question : Est-ce que le logiciel fonctionne correctement ? Le test est un cas particulier de vérification. [48]

- La validation est « le processus d'évaluation d'un logiciel pour déterminer sa conformité avec les besoins spécifiés » [47].

La validation nous tentons de répondre à la question : est-ce le bon système, fonctionne selon les attentes de l'utilisateur?

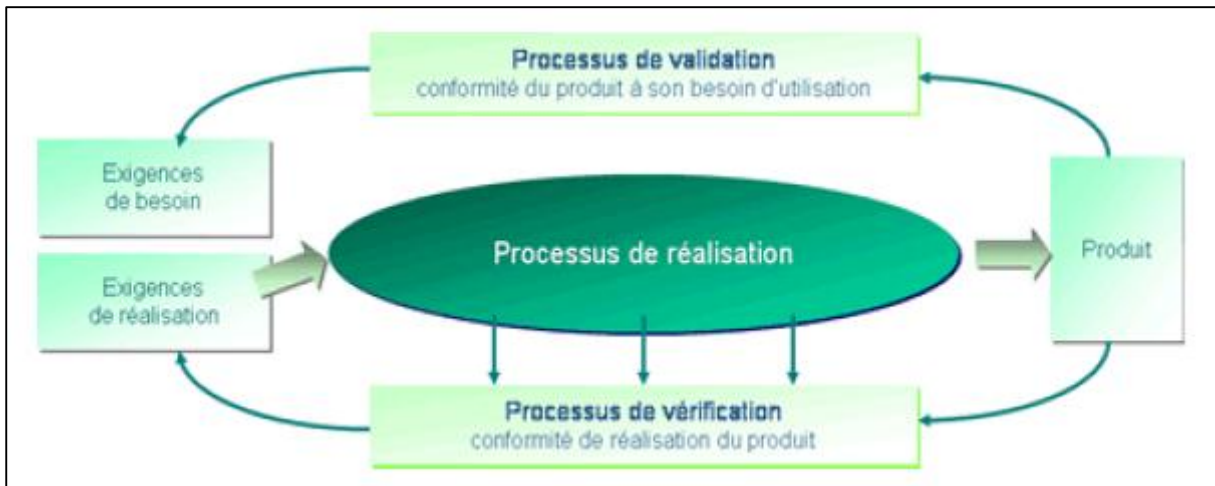


Figure 3.1: Concepts de vérification et validation [49]

- Une erreur (en anglais: error) est une différence entre une valeur ou une condition calculée, observée ou mesurée et la valeur ou condition spécifiée, qui est vraie ou théoriquement correcte [50].
- Un défaut(en anglais: Bug, defect) une imperfection dans un composant ou un système qui peut conduire à ce qu'un composant ou un système n'exécute pas les fonctions requises, par exemple une définition de données incorrecte. Un défaut peut causer la défaillance d'un composant ou d'un système.. [50]
- Une panne ou défaillance (en anglais: failure) est l'incapacité d'un système ou d'un de ses composants d'effectuer les fonctions demandées dans les conditions de performance spécifiées [50].

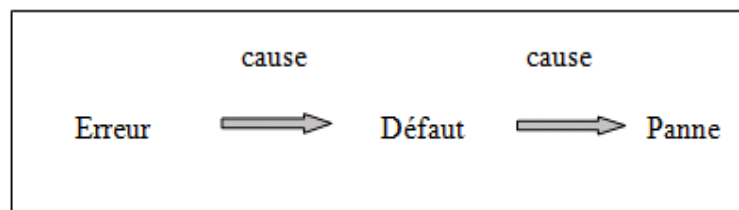


Figure 3.2: Relation entre défaillance / faute / erreur [adapté de 48]

- Cas de test: C'est un ensemble de valeurs d'entrée, de pré-conditions d'exécution, de résultats attendus et de post-conditions d'exécution, développés pour un objectif particulier, tel qu'exécuter un chemin particulier d'un programme ou vérifier le respect d'une exigence spécifique [50].

Cas de test: Chemin fonctionnel à mettre en œuvre pour atteindre un objectif de test. Un cas de test se définit par le jeu de test à mettre en œuvre, le scénario de test à exécuter et les résultats attendus [51].

- Jeu de test: Données en entrée d'un cas de test : valeurs à saisir, données réelles (base de données existante ou de test), génération automatique (aléatoire ou à partir de spécifications). Le même jeu d'essai peut servir à plusieurs cas de test [52].

3.2 Pourquoi le test des logiciels

L'importance du test dans le processus de développement d'un logiciel n'est plus à démontrer. D'une part, en raison de son importance économique croissante (60 % du coût total du développement) et d'autre part, parce que le test constitue une tâche essentielle dans l'élaboration de la qualité d'un logiciel [53]. Considéré longuement comme une activité de «second rang» dans le processus de développement d'un logiciel, le test connaît actuellement une véritable révolution fondée sur une industrialisation de ses processus, une professionnalisation des métiers du test, l'arrivée à maturité d'une chaîne outillée allant des exigences au référentiel de tests, et enfin la mise en place de centres de service, internes ou externes, dédiés aux activités de test. Son objectif est de mettre en œuvre le logiciel en utilisant des données similaires aux données réelles, pour observer les résultats, détecter les anomalies et en déduire l'existence d'erreurs.

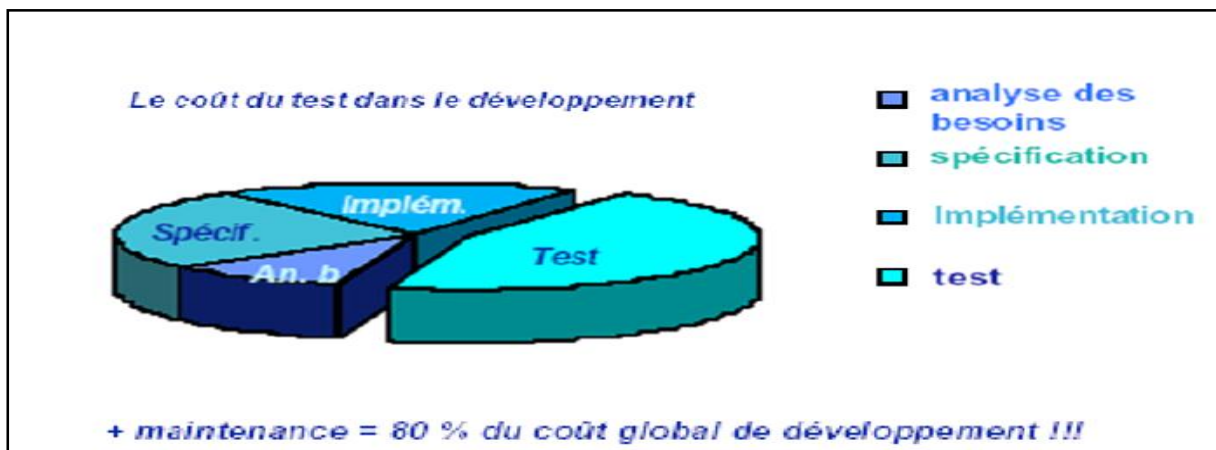


Figure 3.3: Coût du test [48]

3.3 Objectif du test

Le test n'a pas pour objectif de diagnostiquer la cause des erreurs, de corriger les fautes ou encore de prouver la correction d'un programme. Le test vise à mettre en évidence les erreurs d'un logiciel, il peut prouver la présence de l'erreur, mais il ne peut pas prouver leur absence. Le test vise à détecter les défauts avant qu'elles ne causent une panne du système produit et amener le logiciel testé à un niveau acceptable de qualité (après la correction des défauts identifiés) [48].

3.4 Le test dans le cycle de développement

Les activités liées au test se déroulent tout au long du cycle de vie (ou cycle de développement) et interagissent fortement avec les activités de développement. Le modèle cycle de vie en V constitue l'état de l'art, dans le domaine de test du logiciel. C'est le modèle le plus utilisé et traité dans la littérature de test [54];[55]. Ce modèle tel qu'illustré à la figure 3.4, divise le processus de test en plusieurs niveaux, effectués conjointement avec l'implantation du logiciel. Il commence par le test de petits morceaux et avance vers des plus grands, reflétant différents points de vue de test à différents niveaux de détail.

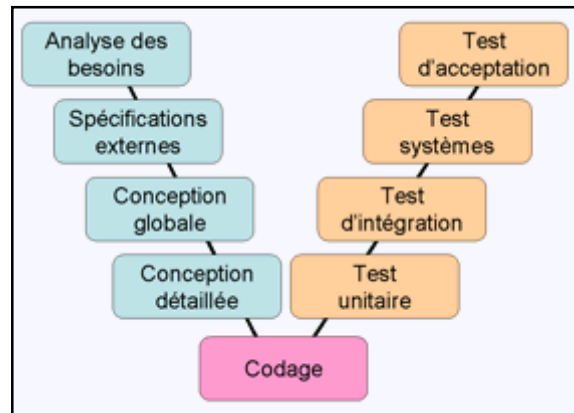


Figure 3.4: Cycle de vie en V [56]

3.4.1 Le test unitaire

Une unité est vue comme une fonction ou une procédure implémentée dans un langage procédural. Dans un système orienté-objet, les méthodes et les classes/objets ont été suggérées par les chercheurs comme un choix d'unité [57].

Les tests unitaires représentent un niveau de test vital. Plus les unités sont petites et simples, plus il sera facile de les tester. Lorsqu'une défaillance est révélée par les tests, il est

plus facile de la localiser et de la réparer (débogage) du moment que l'on ne considère que l'unité sous test.

Le but principal des tests unitaires est de comparer les fonctions d'une unité par rapport à certaines spécifications. En fait, on ne cherche pas à montrer qu'une unité satisfait ses spécifications mais à montrer le contraire, c'est-à-dire qu'elle contredise ses spécifications. Les tests unitaires sont menés en se basant sur le plan de tests unitaires conçus à l'étape de «conception détaillée» du cycle de vie du logiciel. Ce plan met en évidence les spécifications à tester pour chaque unité.

3.4.2 Le test d'intégration

Dans les tests unitaires, le testeur tente de détecter les fautes relatives aux fonctionnalités et la structure de l'unité. Quelques tests d'interface simples sont réalisés lorsque les unités interagissent avec des pilotes (drivers) et des bouchons (stubs). Par ailleurs, les interfaces sont plus adéquatement testées lors des tests d'intégration lorsque chaque unité est connectée aux unités avec lesquelles elle interagit.

L'assemblage ou le processus d'intégration permet de constituer un système complet prêt à être testé lors de la phase suivante "test système". Les tests d'intégration devraient être exécutés sur des unités qui ont été corrigées et qui ont passé les tests unitaires avec succès. Pour effectuer les tests d'intégration, il faudra adopter une stratégie d'intégration. Il existe plusieurs stratégies parmi les plus connues : la stratégie top-down, la stratégie bottom-up, et la stratégie big-bang. Les tests d'intégration se basent sur le graphe d'appel entre les unités. En se basant sur le graphe d'appel, il sera plus simple d'appliquer la stratégie choisie. Le test d'intégration « Top-Down » consiste à intégrer les unités en commençant par la racine du graphe (sommet). L'intégration « Bottom-Up » consiste à intégrer les unités en commençant par les unités qui sont à la base du graphe (feuilles). Le test d'intégration « Big-Bang » consiste à assembler toutes les unités en même temps et à exécuter l'ensemble [41], [58].

3.4.3 Le test de système

Les objectifs des tests système sont de détecter les fautes relatives au comportement du système global, plutôt que le comportement de chacun des éléments, et de tester le logiciel dans son fonctionnement global. Ce niveau d'évaluation implique le système dans sa globalité, et pas seulement les interactions entre unités. Le but des tests système est de s'assurer que le système est en accord avec les besoins.

Les tests système permettent d'évaluer les caractéristiques de la qualité requises telles que la fiabilité, l'utilisabilité, la performance et la sécurité. A ce stade, il sera spécialement utile de pouvoir détecter des défaillances dans les interfaces logicielles et matérielles externes, par exemple, celles qui causent des conditions extrêmes, des verrous, des problèmes d'interruption et de traitement des exceptions, et l'utilisation inefficace de la mémoire.

Afin de tester les différentes caractéristiques de la qualité, il existe plusieurs types de tests système parmi eux [41], [59]:

Les tests de fonctionnalité

Consistent à tester si le système accomplit les fonctions pour lesquelles il a été conçu.

Les tests de volume

Consistent à tester si le système est capable de manipuler un gros volume de données spécifiées dans les besoins.

Les tests d'usage (ou d'utilisabilité)

Ce type de tests s'intéresse au facteur humain. Il s'agira de sélectionner un sous-ensemble représentatif d'utilisateurs potentiels afin de tester le système.

Les tests de sécurité

Consistent à tester les propriétés de sécurité que le système doit assurer telles que la disponibilité, l'intégrité, la confidentialité des données et des services. Le but de ces tests est de révéler les failles de sécurité.

Les tests de performance

Consistent à tester la performance du système en révélant ses défaillances en termes de temps de réponse ou encore de capacité de traitement sous certaines conditions.

Les tests de robustesses

Visent à valider le fait que le système réagit correctement à une utilisation non-conforme (par exemple à la saisie de données invalides).

Les tests de configuration

Consistent à tester le système de façon à vérifier s'il fonctionne dans les différentes configurations (réseaux, systèmes d'exploitation,...) pour lesquelles il a été conçu.

Les tests de fiabilité

Consistent à tester les propriétés de fiabilité du système spécifiées dans les besoins.

3.4.4 Le test d'acceptation

Une fois les tests système achevés, on passe à l'étape des tests d'acceptation. Les tests d'acceptation consistent à mettre le logiciel à la disposition du client afin que les utilisateurs potentiels puissent l'évaluer dans les conditions réelles. Un plan de test est conçu en collaboration avec les clients afin de déterminer les cas de test à considérer lors des tests d'acceptation.

À côté de ces étapes, il y a encore un autre type de test, c'est le test de non-régression. Ce type de test est demandé lorsqu'on modifie un module de logiciel (ajout de traitements, optimisation du code...) et qu'on le réintègre dans le système. Ce type de test est très semblable au test d'intégration. La seule différence réside dans le fait que pour le test d'intégration, on doit tester tôt ou tard, toutes les parties du logiciel sous test. Dans le test de non-régression, on ne s'intéresse qu'aux parties ayant subi l'impact de la modification [58]. Son but est d'assurer que les corrections et les évolutions du code n'ont pas introduit de nouveaux défauts.

3.5 Classification des méthodes de test

Il existe différentes classes de méthodes de test selon que le programme à valider (sous test) est exécuté ou non pendant le test. Les méthodes de test existantes sont divisées en deux catégories : les tests statiques et les tests dynamiques. Les méthodes de tests statiques sont utilisées pour examiner le logiciel sans l'exécuter ; les méthodes de tests dynamiques requièrent la génération de cas de tests pour l'exécution du logiciel [60].

3.5.1 Les méthodes de test statiques

Les méthodes de test statiques consistent en l'analyse textuelle du code logiciel afin d'y détecter des erreurs, sans l'exécution du programme. L'idéal, lors de ce test, est de réunir quatre intervenants : un Modérateur, le Programmeur, le Concepteur et un Inspecteur. [48]

Le test statique est appliqué sur une description du programme ou directement sur le texte du programme mais sans exécuter ce dernier. Il consiste en une analyse du texte dans le but d'en déduire des propriétés : on peut par exemple, détecter des variables utilisées sans être au préalable initialisées, des morceaux du programme qui ne seront jamais exécutés

(c'est ce que l'on appelle du code mort), des boucles sans fin, etc. Il existe différents procédés de test statique dont les plus connus sont l'inspection, l'analyse d'anomalies, l'évaluation symbolique [61].

- Revue ou inspection: Examen détaillé d'une spécification, d'une conception ou d'une implémentation par une personne ou un groupe de personnes, afin de détecter des fautes, des violations de normes de développement ou d'autres problèmes [40].
- L'analyse d'anomalies: Un certain nombre d'anomalies peuvent être facilement détectées dans les logiciels via les analyses faites à la compilation. Ces anomalies peuvent être par exemple la violation de certaines règles raisonnables de typage, des utilisations impropres de pointeurs, la présence d'expressions ayant clairement une valeur constante, des défauts de portabilité, des sorties ou entrées de boucle hors des points standard, la déclaration d'arguments ou de bibliothèques jamais utilisées, etc. [44]. Certaines de ces anomalies n'impliquent pas nécessairement une faute logique, mais elles peuvent en révéler avec une assez forte probabilité. On peut remarquer également que l'utilisation d'un langage de programmation fortement typée éliminera, de fait, un grand nombre de ces fautes [62].
- L'évaluation symbolique: Elle consiste à simuler l'exécution du programme sur des données symboliques. Pour tout chemin sélectionné du graphe de contrôle, les données d'entrée sont représentées par des symboles. On obtient ainsi des expressions correspondant au texte du programme. L'évaluation symbolique produit, pour chaque variable de sortie du programme, une expression retraçant les calculs effectués pour l'obtenir (le long de ce chemin) et les conditions associées (sur le chemin) [44]. Il ne s'agit pas d'un test dynamique car l'évaluation symbolique ne requiert pas l'exécution du logiciel [62].

3.5.2 Les méthodes de test dynamiques

Les méthodes de test dynamiques consistent en l'exécution du programme à valider (sous test) à l'aide d'un jeu de test. Elles visent à détecter des erreurs en confrontant les résultats obtenus par l'exécution du programme à ceux attendus par la spécification de l'application. Le test dynamique suppose l'exécution des quatre étapes suivantes [61]:

- Sélection de jeux de tests. Cette étape consiste à trouver des données d'entrée, des scénarios pour exécuter le test ainsi que le résultat attendu.

- Soumission du jeu de tests. Afin d'exécuter les tests retenus lors de l'étape précédente, un programme de test est exécuté avec les données de test. Ce programme peut être le logiciel qu'on veut tester dans le cas du test système ou du test d'acceptation. Pour le test unitaire et le test d'intégration, le programme de test est un pilote de test qui doit mettre en exécution la partie de logiciel qu'on souhaite tester.
- Analyse des résultats. Il consiste à décider du succès ou échec du jeu de test. Il se pose alors le problème de l'oracle car la décision n'est pas toujours simple, ni même possible. Un oracle est une fonction qui doit permettre de distinguer si une sortie du logiciel est correcte. Si l'oracle a détecté une défaillance, il faut la corriger et ré-exécuter le même cas de test pour vérifier que la correction a effectivement éliminé la faute. Si l'oracle n'a pas détecté de défaillance, on a deux directions soit le critère d'arrêt est satisfait et la phase de test du logiciel est terminée ; soit un nouveau cas de test sera considéré.
- Evaluation de la qualité des tests effectués (critères d'arrêt). On peut rarement effectuer un test exhaustif. Dans la plupart des cas, on doit se contenter d'un test moins coûteux et il faut donc avoir un critère pour décider quand le test est satisfaisant. La figure 3.5 donne l'algorithme de test dynamique.

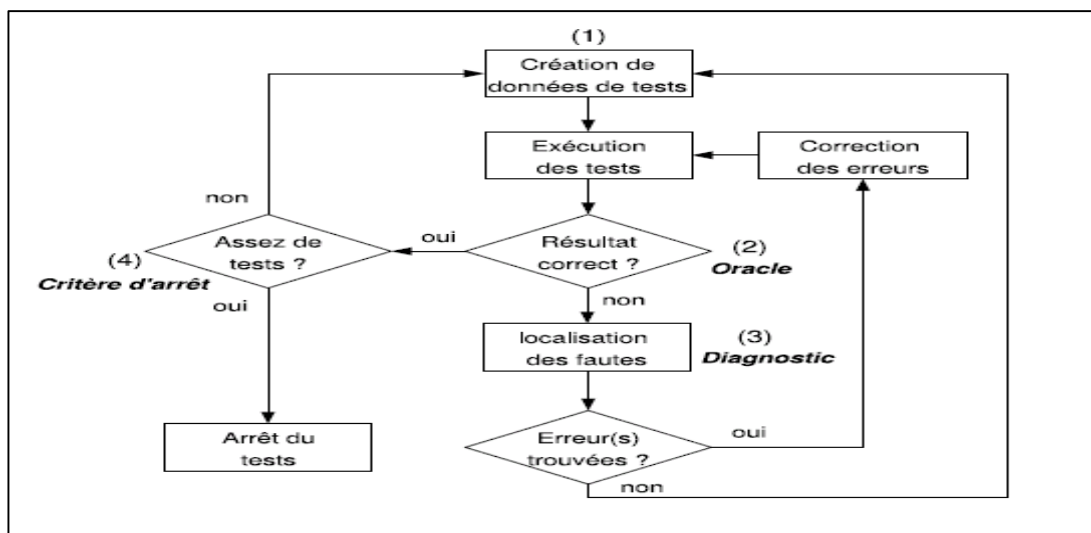


Figure 3.5: Les étapes du test dynamique [63]

3.5.2.1 Le test structurel (boîte blanche)

Le test structurel est basé sur le détail du système. Il est effectué sur des produits dont la structure interne est accessible. Il est dérivé en examinant l'implémentation du système (langage source, conception de base de données, etc.). Il s'intéresse principalement aux

structures de contrôle et aux détails procéduraux. Il permet de vérifier si les aspects intérieurs de l'implémentation ne contiennent pas d'erreurs de logique. Il vérifie si toutes les instructions de l'implémentation sont exécutables. Donc, le test structurel est caractérisé par une sélection des jeux de tests reposant sur la description de la structure du logiciel à tester. Deux types principaux de description sont utilisés le graphe de contrôle et le flot de données. Ce test repose sur la sélection de données d'entrée qui déclenchent l'exécution de certains de ces chemins (figure 3.6). Il n'est pas nécessaire et généralement pas possible de parcourir tous les chemins. En pratique, on se donne des critères de couvertures plus restreints. En effet, la présence de boucles peut facilement conduire à un trop grand nombre de chemins, pouvant être infini, et l'existence de chemins impraticables empêche de couvrir tous les chemins. Les critères de couverture de la structure peuvent se baser sur le graphe de contrôle ou sur le flot de données [61]:

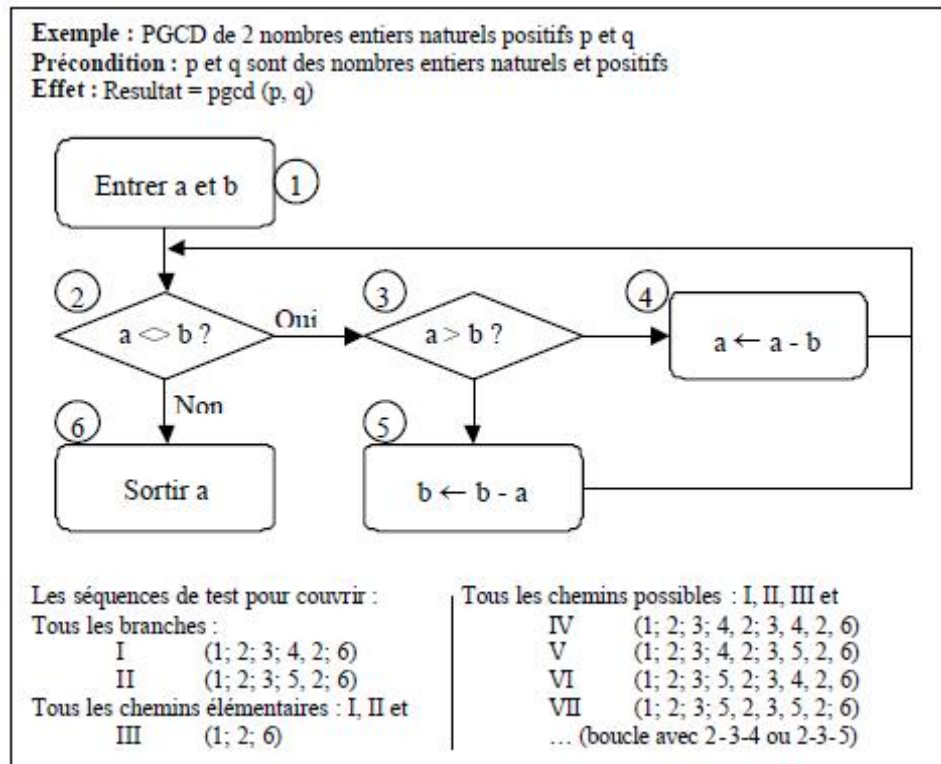


Figure 3.6: Un exemple pour le test structurel [58]

- Critères de couvertures basées sur le graphe de contrôle: un graphe de contrôle est un graphe orienté, composé d'un ensemble de nœuds reliés par un ensemble d'arcs orientés. Un arc (branche) décrit une séquence linéaire contiguë d'opérations (instructions) entre deux nœuds. Un nœud dénote un point d'entrée du programme (nœud d'entrée), un point de sortie (nœud de sortie) ou un transfert de contrôle

(branchement, nœud interne). Les critères de couvertures de graphe de contrôle les plus connus sont:

- Couverture de chaque instruction: la couverture d'instructions consiste à choisir un cas de test parmi tous ceux permettant d'exécuter chaque instruction (accessible) du code. Il s'agit de sélectionner des données de test jusqu'à ce qu'un outil de couverture indique que toutes les instructions du code ont été exécutées.
 - Couverture de tous les chemins: selon le critère de couverture de tous les chemins, il faut emprunter tous les chemins possibles dans le graphe. Ce critère est peu adapté dans la pratique, car le nombre de chemins possibles étant souvent infini et l'existence de chemins infaisables empêche de couvrir tous les chemins. .
- Critères de couvertures basées sur le flot de données: plus utilisés, ils reposent sur la notion suivante : Une utilisation d'une variable est atteinte par une assignation (définition) de cette variable s'il existe un sous chemin du graphe de contrôle qui va de cette assignation à cette utilisation et qui ne comporte pas d'autre assignation de la variable. Dans ce cas les critères de couverture pourront être :
- Couverture de toutes les utilisations: où chaque définition doit être exécutée au moins une fois pour toutes les utilisations qu'elle atteint. De plus, tous les arcs issus de ces utilisations doivent être couverts.
 - Couverture de toutes les définitions: où chaque définition d'une variable doit être exécutée au moins une fois par un chemin qui atteint ensuite une utilisation de la variable.

3.5.2.2 Le test fonctionnel (boîte noire)

L'objectif du test fonctionnel consiste à tester ce qu'est censé faire le programme et non la façon dont il le fait. La structure du programme est donc inconnue ou ignorée et les données d'entrées sont sélectionnées en fonction des spécifications du programme [61]. Parmi les techniques du test fonctionnel on retrouve les suivantes :

- Analyse partitionnelle: la méthode de l'analyse partitionnelle [64] fondée sur les spécifications d'un programme établit des classes équivalentes (partitions). Les données de test se baseront sur le choix d'un représentant quelconque de chaque

classe. De manière générale, pour construire les différentes classes d'équivalence, nous utilisons les spécifications pour en extraire deux types de matières premières: Les domaines de valeurs d'entrée (par exemple : telle variable est une date, telle autre est une personne, un fichier, etc.) et l'ensemble de fonctions qui devront être réalisées par le programme en question (par exemple : on ouvre un fichier, on calcule une date, on estime une moyenne, etc.). On choisit un représentant de chaque classe issue du partage des domaines des définitions des données et/ou un représentant de chaque classe concernée par chacune des fonctions réalisées. On dit alors que les classes créées sont fonctionnellement équivalentes. L'ensemble des classes doit constituer un recouvrement de l'espace total des valeurs.

- Test aux limites: le test aux limites [65] est considéré comme l'une des méthodes fonctionnelles les plus efficaces. En effet, la plupart des programmeurs ont probablement constaté que, souvent, les erreurs sont dues au fait qu'ils n'avaient pas prévu le comportement d'un programme pour des valeurs limites (valeurs très élevées, valeurs nulles d'un indice, valeurs négatives). Les tests aux limites couvrent une large gamme d'erreurs, car les limites forment l'hôte naturel préféré de la plupart des défauts. Leur principale incommodité est surtout la difficulté (et peut-être l'impossibilité) de formalisation de la notion de limite et de marginalité, ce qui explique sans doute leur caractère souvent intuitif et heuristique.

Terminons cette section en insistant sur le fait que les tests fonctionnels et les tests structurels sont complémentaires et permettent de mettre en évidence différents défauts dans un programme. Les tests fonctionnels permettent principalement de détecter des défauts dus à une mauvaise compréhension des spécifications du logiciel. Les tests structurels permettent de détecter des défauts liés à la programmation. Contrairement à ce que peut laisser croire cette terminologie, le test « boîte grise » n'est pas une méthode de test intermédiaire entre le test « boîte blanche » et le test « boîte noire », mais plutôt une union des deux méthodes. L'intérêt du test mixte consiste à utiliser le test fonctionnel pour pallier aux inconvénients du test structurel et vice-versa.

3.5.3 Autres méthodes de test

3.5.3.1 Le test aléatoire

Par définition, la sélection des différents jeux de test est effectuée en choisissant au hasard un jeu de tests parmi les entrées acceptables du logiciel. Les méthodes de sélection de

jeux de tests aléatoires sont donc à priori très simples, sous réserve de disposer d'une caractérisation de l'ensemble des entrées acceptable du logiciel testé. Lorsque la loi de probabilité des entrées lors de l'utilisation effective du produit logiciel est connue, elle peut être utilisée pour guider la sélection. De plus, il est possible de définir un modèle de croissance de fiabilité reposant sur cette loi, ce dernier permet alors de fournir un critère d'arrêt du test. Que la loi de probabilité utilisée soit uniforme ou dépendante des utilisations futures, on parle indifféremment de méthodes de sélection de jeux de tests aléatoires ou statiques [66].

3.5.3.2 Test basé sur les modèles (Model Based Testing)

Une solution de Model Based Testing consiste à produire des cas de test à partir d'un modèle du système à tester. Le principe du test basé sur les modèles(en anglais Model Based Testing ou MBT) est résumé dans la figure 3.7. Cette figure présente les phases clés du processus de validation d'un système par test à partir de modèles. Les flèches pointillées représentent les étapes manuelles, et celles pleines représentent les étapes automatiques ou automatisables. La partie droite de la figure schématise un processus classique de développement. Le cadre pointillé à gauche représente les moyens complémentaires mis spécifiquement en œuvre dans une approche MBT [67] :

- Un modèle formel du système est rédigé à partir des spécifications.
- Un ou plusieurs critères de sélection de tests sont définis. Ils permettent de définir comment les tests sont dérivés du modèle.
- Des cas de test abstraits sont générés en appliquant les critères de sélection de tests au modèle.
- Une relation entre le modèle et le système doit être définie à fin de pouvoir transformer les cas de test abstraits, exprimés au niveau d'abstraction du modèle, en cas de test concrets, exécutables sur le système sous test (SUT).

Et en fin, la partie en aval du schéma représente le cas général de l'exécution des tests (lors d'une approche de validation MBT). Quel que soit la démarche mise en œuvre pour produire les tests, ceux-ci sont exécutés sur le système sous test et un verdict est calculé en comparant les résultats obtenus à partir de l'implémentation à ceux prévus par le modèle.

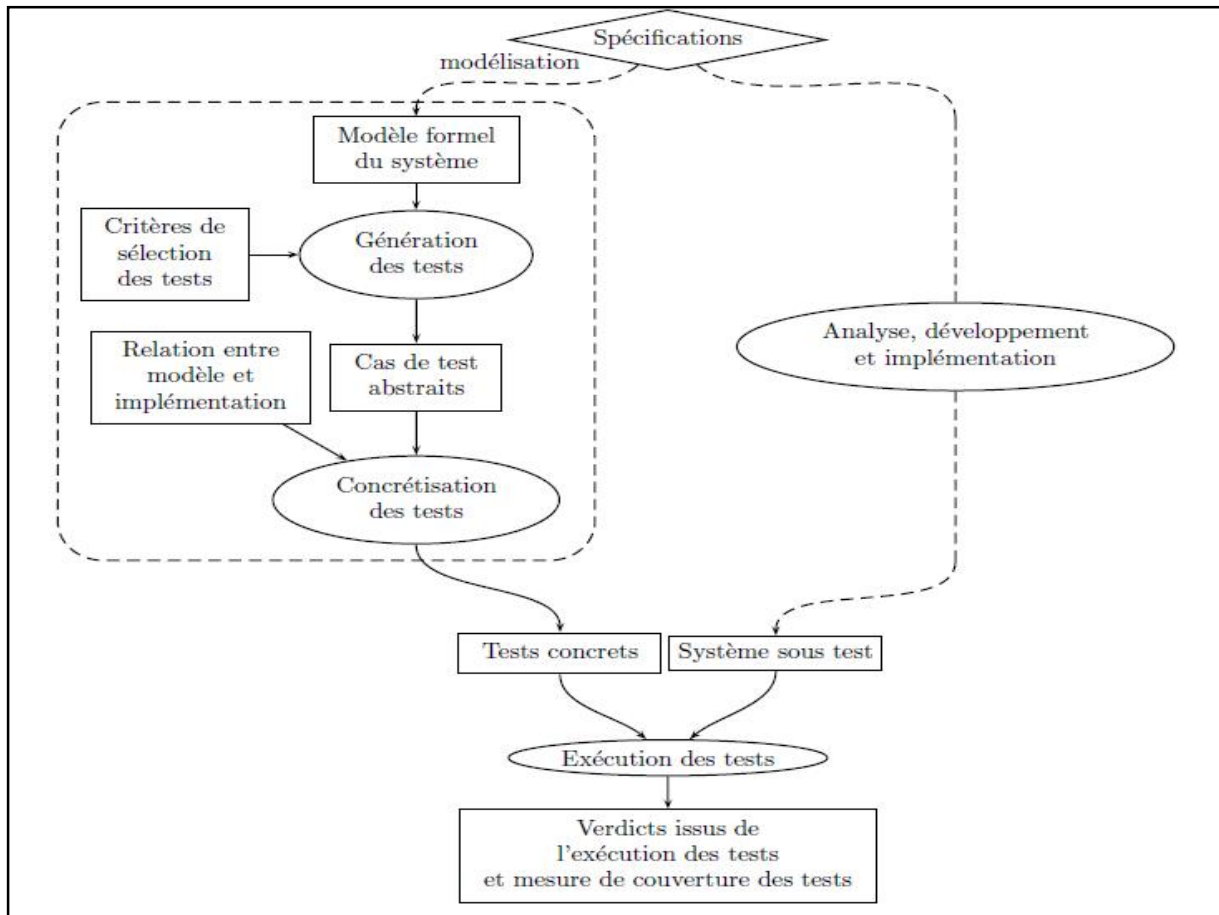


Figure 3.7: Le processus de test basé sur les modèles [67]

Donc un processus MBT inclut les étapes suivantes [68], [69]:

- La modélisation d'un modèle abstrait dédié au test d'un système donné.
- La validation du modèle. Ceci est typiquement fait en retirant les cas de test abstraits du modèle et en les analysant. Cette étape est exécutée pour détecter des erreurs importantes dans le modèle qui peut même gêner la génération des cas de test.
- La génération ou la production des cas de test abstraits à partir de ce modèle. Cette production de tests est la plupart du temps automatisé par des outils de génération automatique de tests.
- La concrétisation des cas de test abstraits en des cas de test concrets qui sont exécutables sur le système sous test,
- L'exécution des tests sur le système et la constitution de leur verdict,
- L'analyse des résultats ainsi obtenus.

Les solutions MBT existantes sont nombreuses et se distinguent notamment par le type de modèle, la méthode de génération des cas de test, ou encore le type d'exécution de ces tests.

Le test basé sur les modèles peut être appliqué à tous les niveaux de test à partir du test unitaire jusqu'au test de système. Le test d'acceptation n'est pas habituellement couvert parce que l'acceptation par les utilisateurs dépend également de beaucoup d'espérances imprécises. Il y a seulement quelques approches pour employer le test basé sur les modèles pour les tests non fonctionnels (structurels). Le schéma 3.8 montre les genres de test appliqué par le test basé sur les modèles. [70]

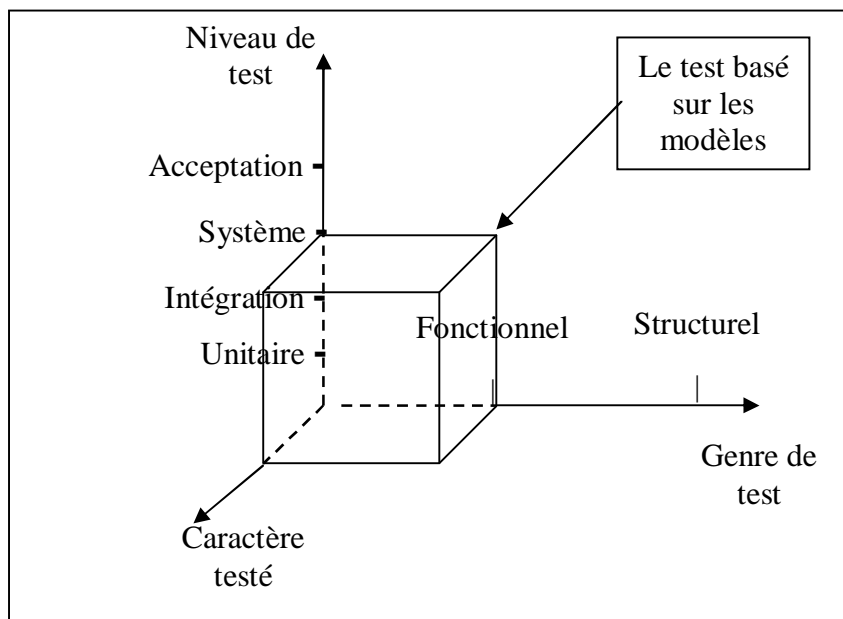


Figure 3.8: Champs d'application du test basé sur les modèles [70]

Il y a plusieurs avantages du test basé sur les modèles: D'abord, le modèle de test est habituellement tout à fait petit, facile à comprendre, et facile à maintenir. Deuxièmement, le test basé sur les modèles aide à réduire les coûts. En outre, l'expérience prouve que la création tôt d'un modèle formel de test aide à trouver des défauts et des conflits dans les besoins. Finalement, le modèle de test peut être employé pour produire automatiquement des petits ou énormes cas de test abstraits qui répondent à un critère de couverture correspondant[70].

3.6 Le critère d'arrêt

Le test a pour but de trouver des erreurs dans une implantation mais ne permet pas, dans le cas général, de s'assurer de la correction d'un programme. En effet, le seul moyen de

prouver par le test la correction d'un programme est de le tester sur tout son domaine d'entrée (test exhaustif). C'est généralement impossible car les domaines d'entrée des programmes usuels sont soit bien trop grands, soit infinis. Il est nécessaire de choisir judicieusement un sous-ensemble du domaine d'entrée du programme pour le test. Le rôle du critère de test (ou critère d'arrêt) est de définir comment choisir l'ensemble des données de test.

Un critère de test est un critère qui doit être vérifié par l'ensemble des cas de test d'un programme à tester. L'utilisation d'un critère de test permet d'assurer le pouvoir de détection d'erreur des tests utilisés. En pratique, les critères de tests peuvent être structurels ou fonctionnels [63].

Dans le cadre du test structurel (ou boîte blanche), c'est-à-dire lorsque que l'on s'appuie sur le code du programme à tester, des critères de test classiques sont la couverture du code ou la couverture du graphe de flot de contrôle. L'objectif est de trouver un ensemble de données en entrée du programme qui permette soit d'exécuter au moins une fois chaque instruction du programme soit de passer au moins une fois par un ensemble de chemins extrait du graphe de flot de contrôle. Dans le cadre du test fonctionnel (ou boîte noire) l'idée est de s'appuyer sur la spécification du programme à tester pour la sélection des données de tests. Par exemple, pour les programmes dont le comportement est spécifié par des machines à états, il existe des critères de test spécifiques qui consistent à sélectionner des données qui permettent de passer au moins une fois par chaque état ou par chaque transition de la machine à état [71], [72].

D'autres approches sont également possibles comme l'analyse de mutation [73] par exemple. Le test par mutation consiste à créer une grande quantité de versions (ou mutants) du programme sous test en introduisant dans chacune une erreur élémentaire. L'hypothèse faite est que si une suite de test est capable de rejeter (ou tuer) tous les mutants alors cette suite de test est capable de détecter des erreurs plus complexes dans le programme original. Le critère d'arrêt s'exprime ici comme le pourcentage de mutants rejetés par la suite de test, appelé score de mutation.

3.7 Difficulté du test

Un test est dit exhaustif, pour une application donnée, si toutes les exécutions possibles de cette application ont été vérifiées [74]. Le test exhaustif est en général impossible à réaliser. En test fonctionnel l'ensemble des données est en général infini ou de très grandes

tailles. Par exemple, un logiciel avec 5 entrées sur 8 bits admet 1280 valeurs différentes en entrée. En test structurel, le parcours du graphe de flot de contrôle conduit à une forte explosion combinatoire. En d'autres termes : Le test ne peut-être qu'une méthode de vérification partielle de logiciels et la qualité du test dépend de la pertinence du choix de données de test. Il existe aussi des difficultés d'ordre psychologique ou «culturel». En effet, le test est un processus destructif : un bon test est un test qui trouve une erreur alors que l'activité de programmation est un processus constructif où on cherche à établir des résultats corrects. Les erreurs peuvent être dues à des incompréhensions des spécifications ou de mauvais choix d'implantation. En d'autres termes : L'activité de test s'inscrit dans le contrôle de qualité et est indépendante du développement. L'exécution des tests peut-être automatisée on peut écrire des programmes qui permettent d'enchaîner l'exécution de plusieurs tests, et qui vérifient que le résultat est correct. On peut ainsi facilement faire des tests de non régression, c'est-à-dire réexécuter un ensemble de tests et ainsi vérifier que les fonctionnalités déjà testées continuent à produire les résultats attendus [75].

Dans la pratique industrielle du test, l'automatisation est trop souvent limitée à l'exécution des tests, parfois avec une mesure de la couverture, et à la comparaison des résultats obtenus avec ceux attendus. C'est à l'utilisateur d'élaborer les cas à tester et les données de test (entrées et sorties attendues), éventuellement à l'aide d'outils de préparation de "scripts de test". Par conséquent, le nombre de tests effectués est limité par le coût et la disponibilité des ingénieurs compétents et la couverture des tests peuvent être difficiles à évaluer. Or, la génération automatique des données de test et d'un oracle permet une sélection des cas de tests basés sur des critères justifiés par des hypothèses explicite [76].

3.8 Le test des systèmes multi-agents

Le test des SMA est une tâche provocante. En particulier, les caractéristiques très particulières des agents de logiciel rendent l'application des méthodes de test existantes difficile. Il y a des questions au sujet de communication aussi bien que la coordination. Tous ces dispositifs sont difficiles non seulement a modélisé et a programmé, mais aussi a testé. Le reste de cette section examine tous les points qui concernent le test des systèmes multi-agents.

3.8.1 Difficultés de test des SMA

Il y a plusieurs raisons de l'augmentation du degré de difficulté de test des SMA : [77]:

- Une complexité croissante due au caractère distribué des applications constituées plusieurs agents s'exécutant de manière autonome et concurrente,
- Une grande quantité de données manipulées par ces agents chacun ayant ses propres objectifs et ses propres buts,
- L'effet non reproductible, qui signifie qu'il n'est pas garanti que deux exécutions du système (voire avec les mêmes données en entrée) aboutissent à un état identique. Comme conséquence directe, la localisation d'une erreur peut s'avérer très difficile, des fois impossible, du fait qu'il n'est pas probable de reproduire l'exécution ayant révélé une erreur [78].
- Ils sont également non déterministes, puisqu'il n'est pas possible de déterminer a priori toutes les interactions d'un agent pendant son exécution.
- Les agents communiquent principalement par le message passant au lieu de l'invocation de méthode, ainsi les approches de test orientées Objet existantes ne s'appliquent pas directement.
- Les agents sont autonomes et coopèrent avec d'autres agents, ainsi ils peuvent fonctionner correctement seuls mais inexactement dans une communauté.
- Un agent mobile est une classe particulière d'agent avec la capacité pendant l'exécution d'émigrer d'une place à l'autre où il peut reprendre son exécution. Cette mobilité représente de nouveaux défis.

En conséquence, le test des SMA demande de nouvelles méthodes de test traitant leur nature spécifique. Les méthodes doivent être efficaces et adéquates pour évaluer les comportements autonomes de l'agent

3.8.2 Niveaux de test des SMA

Il n'est pas possible d'appliquer directement les techniques de test traditionnel pour un système multi-agents. De plus, les spécifications de ces systèmes autonomes et concurrents sont généralement incomplètes, incohérentes voire même incorrectes. Un testeur apercevrait un système multi-agents comme un nombre de différents niveaux d'abstraction.

Le test des SMA se compose de cinq niveaux, comme proposé dans [79] et [80] : unité, agent, intégration, système, et acceptation. Les objectifs de test et les activités de chaque niveau sont décrits comme suit :

3.8.2.1 Test unitaire

Test toutes les unités qui composent un agent, y compris les blocs de code, l'implémentation des unités d'agent comme les buts, les plans, base de connaissance, moteur de raisonnement, spécification des règles, et ainsi de suite ; et s'assurer qu'elles fonctionnent comme conçu.

3.8.2.2 Test d'agent

Teste l'intégration des différents modules à l'intérieur d'un agent ; test les possibilités d'agents pour atteindre leurs buts et pour sentir et effectuer l'environnement.

3.8.2.3 Test d'intégration ou de groupe

Teste l'interaction des agents, le protocole de communication et la sémantique, l'interaction des agents avec l'environnement, l'intégration des agents avec les ressources partagées, application de règlements ; observer les propriétés émergentes, comportements collectifs ; s'assurer qu'un groupe d'agents et les ressources environnementales travaillent correctement ensemble.

3.8.2.4 Test de système ou de société

Teste le SMA comme système qui fonctionnent dans l'environnement ; teste les propriétés émergentes et macroscopiques prévues du système dans son ensemble ; test les propriétés de qualité qui doit être atteindre par le système, comme l'adaptation, la franchise, la tolérance aux pannes et la performance.

3.8.2.5 Test d'acceptation

Test le SMA dans l'environnement de l'exécution du client et vérifier qu'il atteint Les buts de dépositaire, avec la participation des dépositaires.

3.8.3 Aperçu sur le test des systèmes multi-agents

Il y a peu de travaux écrit qui décrit le test des systèmes multi-agents. Le reste de cette section examine le travail récent et actif sur le test des SMA, en respectant les catégories précédentes. Cette classification est prévue pour faciliter seulement la compréhension des travaux de recherches dans ce domaine. Nous récapitulons d'abord les contributions des travaux qui se concentrent principalement sur un niveau particulier de test, puis les travaux qui concernent plus d'un niveau sont présentés.

3.8.3.1 Niveau unitaire**Zhang et al [81, 82, 83]**

Ce corps de travail présente un cadre pour le test automatisé d'unité des systèmes d'agent. L'approche est une approche de test basée sur les modèles où les modèles utilisés sont les objets façonnés de conception de la méthodologie de conception d'agent. La méthodologie choisie (bien qu'elle soit extensible à d'autres méthodologies avec les concepts semblables) est PROMETHEUS [84] et l'outil correspondant qui incorpore les méthodes de test est l'outil de conception de PROMETHEUS (PDT) [82].

Ekinci et al. [85]

Ce travail considère les buts d'agent comme les plus petites unités testables dans le SMA et propose pour tester ces unités un moyen de buts de test. Chaque but de test est conceptuellement décomposé en trois sous-but : installation, but sous test, et affirmer. Les premiers et derniers buts préparent des conditions préalables et vérifient des post conditions respectivement, tout en testant le but sous test.

3.8.3.2 Niveau d'agent**Lam et Barber [86]**

Ce travail propose un processus semi-automatisé pour comprendre les comportements d'agent logiciel. L'approche imite ce qu'un utilisateur humain, peut être un essayeur, fait dans la compréhension de logiciel : construction et raffinement d'une base de connaissance au sujet des comportements des agents, et de l'employer pour vérifier et expliquer les comportements des agents au temps d'exécution.

Nunez et al. [87]

Ce document présente un cadre formel pour spécifier le comportement des agents autonomes de commerce électronique. Les comportements désirés des agents sous test sont présentés au moyen d'un nouveau formalisme, appelé la machine d'état de service qui incarne les préférences des utilisateurs dans ses états. Deux méthodologies de test sont proposées pour vérifier si une exécution d'agent spécifique se comporte comme prévu (c.-à-d., test de conformité) ; un actif, l'autre passif. Dans leur approche de test active, ils emploient pour chaque agent sous test un test (un agent spécial) qui prend les spécifications formelles de l'agent pour le faciliter pour atteindre un état spécifique. La trace opérationnelle de l'agent est alors comparée aux spécifications afin de détecter des défauts. D'une autre part,

les auteurs proposent également d'employer le test passif, dans lequel les agents sous test sont observés seulement, non simulés comme dans le test actif. Les traces invalides, s'ils existent, sont alors identifiées grâce aux spécifications formelles des agents.

Coelho et al. [88]

Inspiré par JUnit [89], ce document propose un cadre pour le test d'unité du SMA basé sur l'utilisation des agents Mock. Leur travail se concentre sur le test des rôles des agents. Les agents Mock qui simulent des vrais agents dans la communication avec l'agent sous test ont été implémentés manuellement ; chacun correspond à un rôle d'agent.

Nguyen et al [90]

Ce travail tire profit des ontologies d'interaction d'agent qui définissent la sémantique des interactions d'agent : (i) produisent des entrées de test ; (ii) guident l'exploration de l'espace d'entrée pendant la génération ; et, (iii) vérifient les messages échangés entre les agents en ce qui concerne l'ontologie définie d'interaction. Un ensemble de règles de génération a été défini et les employant un cadre de test automatisé peut tester un agent particulier intensivement par un grand et divers nombre de cas de test.

Nguyen et al [91]

L'autonomie d'agent rend le test plus dur : les agents autonomes peuvent réagir dans différentes manières aux mêmes entrées dans le temps, parce que, ils ont des buts et des connaissances qui changent. Le test des agents autonomes exige un procédé qui couvre à un large éventail de contextes de cas de test, et cela peut rechercher le plus exigeant de ces cas de test. Nguyen et autres [91] présentent et évaluent une approche pour tester les agents autonomes qui emploie l'optimisation évolutionnaire pour produire des cas de test exigeants. Dans ce travail, les auteurs ont proposé une manière systématique d'évaluer la qualité des agents autonomes. D'abord, des exigences de dépositaire sont représentées comme qualité des mesures, et les seuils correspondants sont employés comme critères de test. Les agents autonomes doivent répondre à ces critères afin d'être fiables. Des fonctions de forme physique qui représentent des objectifs de test sont définies en conséquence, et guident cette technique de génération de test évolutionnaire pour produire des cas de test automatiquement.

3.8.3.3 Niveau d'intégration**Serrano et Botia [92], Serrano et al. [93]**

Une des issues quand on test les SMA est leur évolutivité, en raison du nombre d'agents et plus spécifiquement le nombre important d'interactions qui peuvent surgir. Ceci le rend très difficile pour appliquer des méthodes de test classiques. ACLAnalyser aborde ces issues en appliquant des techniques d'exploitation de données (Data mining) sur les notations des exécutions de SMA (au commencement, sur Plate-forme d'agent de JADE). Ceci laisse découvrir des modèles d'apparition au système niveau (de social). Par exemple, la création des communautés des agents avec fort liens d'interaction peuvent être détectés avec les équipements de groupement d'ACLAnalyser.

3.8.3.4 Plusieurs niveaux de test**Tiryaki et al. [94]**

Ce travail propose une approche test-conduite de développement de SMA qui soutient la construction itérative et par accroissement de SMA. Un cadre de test a appelé SUnit, qui a été construit sur JUnit [89] et Seagent [95], a été développé pour soutenir l'approche. Le cadre permet des écritures de tests pour des comportements d'agent et des interactions entre les agents.

Gomez-Sanz et al. [96]

Ce travail se concentre sur l'agent et le niveau d'interaction de test. Le kit de développement d'INGENIAS fournit également des équipements pour le test de SMA. Il se fonde sur l'approche modèle-conduite d'INGENIAS, qui permet les spécifications des suites de test en modelant le SMA. Il est possible de spécifier des déploiements de test, des tests d'interaction, et l'inspection d'état mental d'agent. Des modèles de SMA sont alors transformés en code sur le cadre d'agent d'INGENIAS (exécuter sur la plate-forme d'agent de JADE), où les tests peuvent être exécutés et le développeur peut obtenir l'information sur la progression des déroulements des opérations d'organisation, les interactions, et les états mentaux d'agents. Des suites de test peuvent être automatisées, et ceci facilite le développement agile modèle-conduite, car il est facile d'effectuer des itérations, chacun qui passe une suite de test.

Nguyen et al. [97]

Le papier propose une méthodologie de test complète, appelée le test de logiciel orienté-but (Goal-Oriented Software Testing GOST), qui complète et exploite analyse orienté-but et conception pour dériver des suites de test à tous les niveaux de test, à partir de l'unité à l'acceptation. Le GOST fournit un modèle de processus de test qui apporte les raccordements entre les buts et les cas de test explicites et une systématique manière de dériver des cas de test de l'analyse de but. Ceci peut aider à découvrir des problèmes tôt, évitant d'implémenter des spécifications incorrectes. En outre, le GOST vient avec un outil qui soutient la génération de cas de test, les la spécification, et l'exécution.

3.9 Conclusion

D'après ce qu'on a vu dans ce chapitre, on peut conclure que le test est très important pour la vérification et la correction d'un logiciel. L'application des méthodes de test existantes sur les systèmes multi-agents est difficile à cause de la nature spécifique des agents. La majeure partie du travail existant de recherches sur le test des SMA se concentre principalement sur le niveau d'agent et d'intégration.

L'approche de test qui va être discuté dans ce mémoire est fondée sur le réseau de référence qui est un réseau de pétri spécial. Dans le prochain chapitre, nous allons présenter le formalisme de réseaux de Pétri, les réseaux de références et le paradigme des réseaux dans les réseaux.

Chapitre 4: Les réseaux de Pétri

Les Réseaux de Pétri (RdP) sont des outils à la fois graphiques et mathématiques permettant de modéliser le comportement dynamique des systèmes à évènements discrets. Leur représentation graphique permet de visualiser d'une manière naturelle le parallélisme, la synchronisation, le partage de ressources...etc. Leur représentation mathématique permet d'analyser le modèle pour étudier ses propriétés et de les comparer avec le comportement du système réel.

Dans le présent chapitre, nous nous intéressons au formalisme RdP. Nous présentons dans un premier lieu les RdP généralisés. Ensuite, nous rappelons certaines de leurs extensions. À la fin de ce chapitre les RdP utilisés pour modéliser les systèmes multi-agents mobiles et en particulier les réseaux de référence sont explorés.

4.1 Qu'est ce qu'un modèle?

La modélisation est le processus de produire un modèle [98]. Un modèle est une abstraction d'un système construite dans un but précis. On dit alors qu'un modèle est une représentation du système. [99] Un modèle est une abstraction dans la mesure où il contient un ensemble restreint d'informations sur un système. Il est construit dans un but précis et les informations qu'il contient sont choisies pour être pertinentes vis-à-vis de l'utilisation qui sera faite du modèle.

4.2 Pourquoi modéliser?

Le but principal d'une modélisation est en fait souvent de rendre clair et explicite un ensemble d'informations largement implicites. Lors de la construction du modèle, un grand nombre d'ambiguïtés doivent être levées [100].

Modéliser un système avant sa réalisation permet de mieux comprendre le fonctionnement du système. C'est également un bon moyen de maîtriser sa complexité et d'assurer sa cohérence [101]. Dans le domaine de l'ingénierie du logiciel, le modèle permet de mieux répartir les tâches et d'automatiser certaines d'entre elles. C'est également un facteur de réduction des coûts et des délais. Le modèle est aussi indispensable pour assurer un bon niveau de qualité et une maintenance efficace.

4.3 Les réseaux de Pétri

Historiquement [102], le réseau est présenté par Carl Adam Pétri dans sa thèse "Communication avec des Automates" en Allemagne à Bonn en 1962 [103]. Ce travail a continu à être développé par Anatol W. Holt, F. Commoner, M. Hack et leurs collègues dans le groupe de recherche de Massachusetts Institute Of Technology (MIT) dans des années 70s. En 1975 la première conférence de réseau de Pétri et des méthodes relationnels ont été organisés à MIT. En 1981 le premier livre du réseau de Pétri a été publié en Anglais par J. Peterson. Aujourd'hui, suivre Pétri-Net Newsletter, chaque année il y a des 600 aux 800 d'œuvres des réseaux de Pétri sont publiés.

4.3.1 Définition informelle

Les Réseaux de Pétri permettant de modéliser les systèmes dynamiques, c'est-à-dire des systèmes évoluant d'un état à un autre à la suite des événements externes ou internes [104].

Informellement un RdP est un graphe orienté, comprenant deux sortes de nœuds: des places et des transitions. Ce graphe est constitué de telle sorte que les arcs du graphe ne peuvent relier que des places aux transitions ou des transitions aux places. On représente graphiquement les places par des cercles et les transitions par des barres ou des traits. Chaque place contient un nombre (ou non) de marques ou jetons (voir figure 4.1).

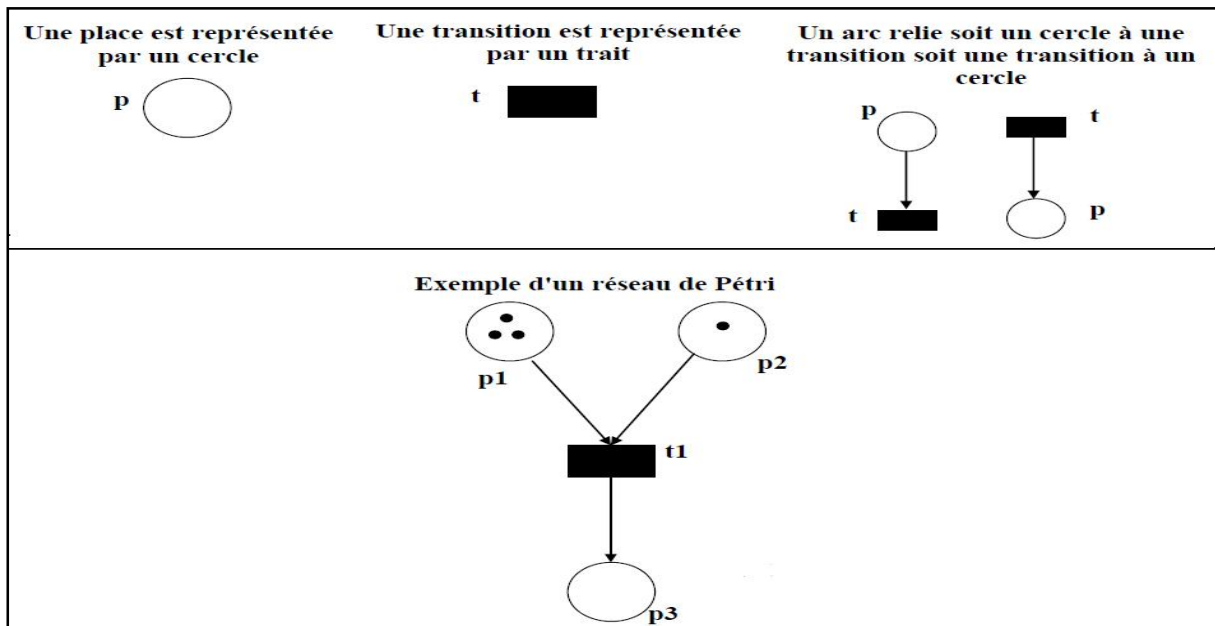


Figure 4.1: Les principes de base des réseaux de pétri

Dans la modélisation des systèmes, Les places servent à représenter les états du système modélisé, tandis que les transitions représentent les changements d'état ou les événements [104]. Quelques interprétations typiques des transitions et leurs places d'entrées et de sorties sont présentées dans Le tableau 4.1 [105].

Places d'entrées	transition	Places de sortie
Pré-conditions	Evènement	Post-conditions
Données d'entrée	Traitement	Données de sortie
Signaux d'entrée	processeur	Signaux de sorties
Ressources demandées	Tâche	Ressources libérées
Conditions	Clauses en logique	Conclusions
Buffers	Processeur	Buffers

Tableau 4.1: Quelques interprétations typiques de transitions et de places [105]

4.3.2 Définition formelle

Formellement un RdP [105] est un quintuple, $R = (P, T, F, W, M_0)$ tel que :

- $P = \{p_1, p_2, \dots, p_m\}$ ensemble fini de m places ;
 - $T = \{T_1, T_2, \dots, T_n\}$ ensemble fini de n transitions ;
 - $F \subseteq (P \times T) \cup (T \times P)$ ensemble d'arcs;
 - $W : F \rightarrow \{1, 2, 3, \dots\}$ fonction de poids ;
 - $M_0 : P \rightarrow \{0, 1, 2, 3, \dots\}$ marquage initial ;
- } $P \cap T = \emptyset$ et $P \cup T = \emptyset$

Le RdP dont tous ses arcs sont de poids "1" est appelé RdP ordinaire. Dans le cas où les arcs peuvent avoir des poids supérieurs à "1", il s'agit de RdP généralisé.

La structure de RdP est donnée par le quadruplet $Q = (P, T, F, W)$, Q représente le RdP sans aucune spécification de marquage initial M_0 . Un RdP marqué est noté par $R = (Q, M_0)$. Soit $R = (P, T, F, W, M_0)$ un RdP. On a les notations suivantes:

- ${}^{\circ}t$: l'ensemble des places d'entrée de la transition t.
- t° : l'ensemble des places de sortie de la transition t.
- ${}^{\circ}p$: l'ensemble des transitions d'entrée de la place p.
- P° : l'ensemble de transitions de sortie de la place p.

La figure 4.2 donne une idée sur toutes les notions définit dans cette section.

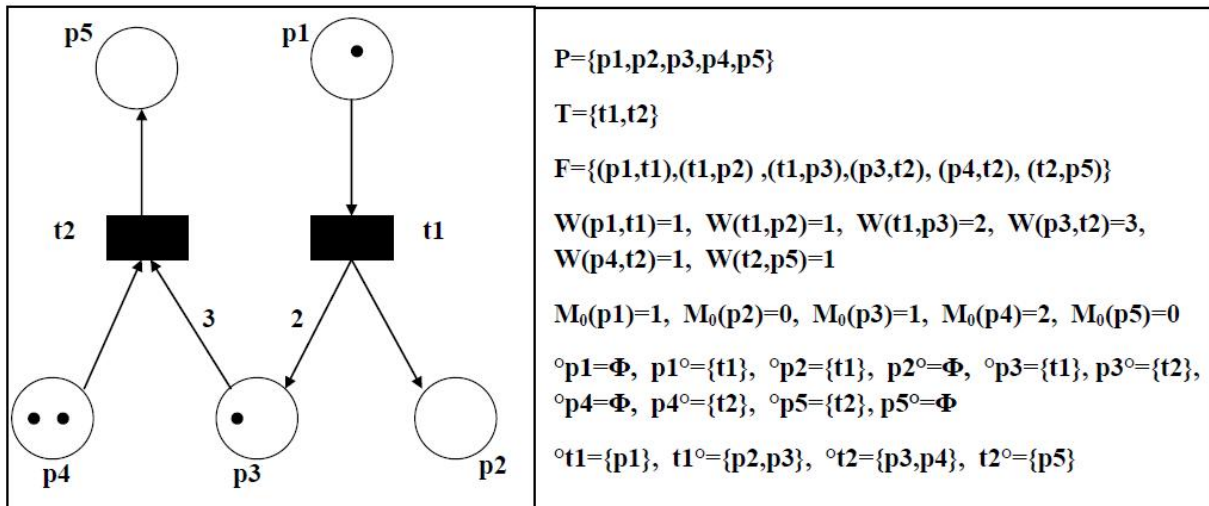


Figure 4.2: Définition formelle d'un Réseau de Pétri

4.3.3 Marquage

On appelle marquage une distribution de jetons sur les places. Le marquage initial noté M_0 est la distribution initiale de jetons dans le réseau à l'instant initial. Le marquage d'un RdP est une application $M : P \rightarrow N$ donnant pour chaque place le nombre de jetons qu'elle contient. Le marquage de la place p_i est noté par $M(p_i)$ qui est un nombre entier. Un marquage est représenté par un vecteur [106] (voir figure 4.3).

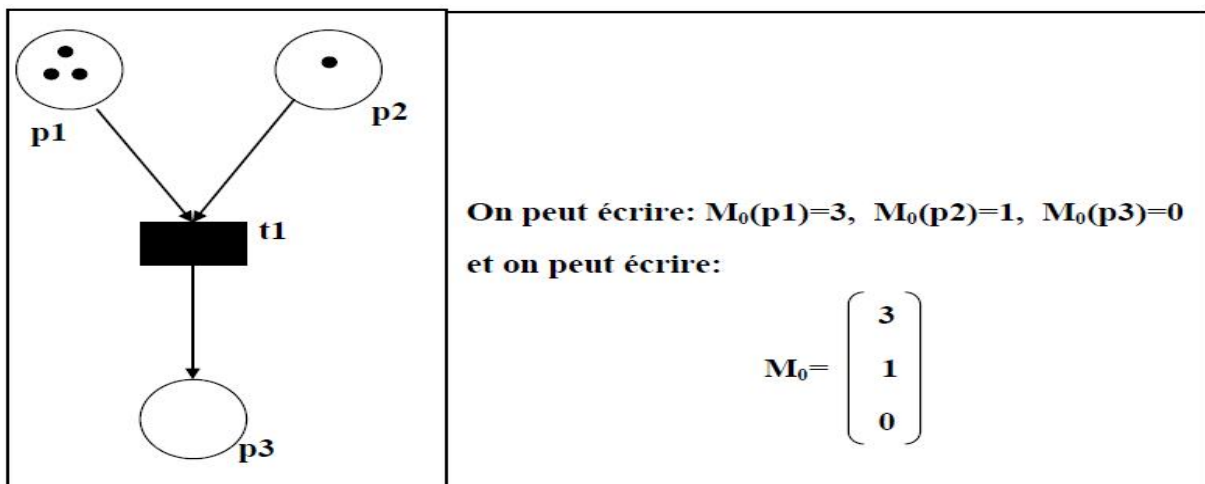


Figure 4.3: Exemple de marquage initiale

Le marquage M définit l'état du système décrit par le réseau à un instant donné. C'est un vecteur colonne de dimension égale au nombre de places dans le réseau. Le i ème élément du vecteur correspond au nombre de jetons contenus dans la place P_i [107].

4.3.4 Evolution d'un réseau de Pétri

L'évolution d'un RdP correspond à l'évolution de son marquage au cours du temps, cette évolution simule le comportement dynamique du système modélisé, il se traduit par un déplacement de marques. Déterminer l'évolution d'un RdP correspond en fait à le simuler, terme plus généralement utilisé en modélisation.

Rappelant qu'à chaque arc ((p,t) ou (t,p)) est associé un nombre entier strictement positif appelé poids de l'arc ($W(p,t)$ ou $W(t,p)$) et notant que lorsque ce poids n'est pas signalé dans le réseau de pétri, il est égal à "1" par défaut , on peut alors donner les règles de changement de marquage du réseau suivantes [108]:

- Une transition t est validée (ou sensibilisée, franchissable ou tirable) si toutes les places p en amont de celle-ci ($p \in {}^{\circ}t$) possèdent au moins $W(p,t)$ jetons.

$$t \text{ est validée} \leftrightarrow \forall p \in {}^{\circ}t : M(p) \geq W(p,t)$$

- Un franchissement d'une transition validée t consiste à supprimer $W(p,t)$ jetons de chaque place d'entrée, et ajouter $W(t,p)$ jetons à chacune de places de sortie de t.

$$\text{le franchissement de t validée} \leftrightarrow \forall p \in {}^{\circ}t : M(p) = M(p) - W(p,t)$$

$$\text{et } \forall p \in t^{\circ} : M(p) = M(p) + W(t,p)$$

Alors le franchissement d'une transition dans un marquage M donne un nouveau marquage.

D'une autre façon on peut dire qu'une transition est franchissable (validée) lorsque chaque place qui lui est en amont (ou toutes les places d'entrée de la transition) contient un nombre de jeton plus grand ou égale au poids de l'arc qui la relie à cette transition. Le franchissement consiste à retirer un nombre de jeton égal au poids de l'arc de chacune des places d'entrée et à rajouter un nombre de jeton égal au poids de l'arc à chacune des places de sortie de la même transition voir figure 4.4.

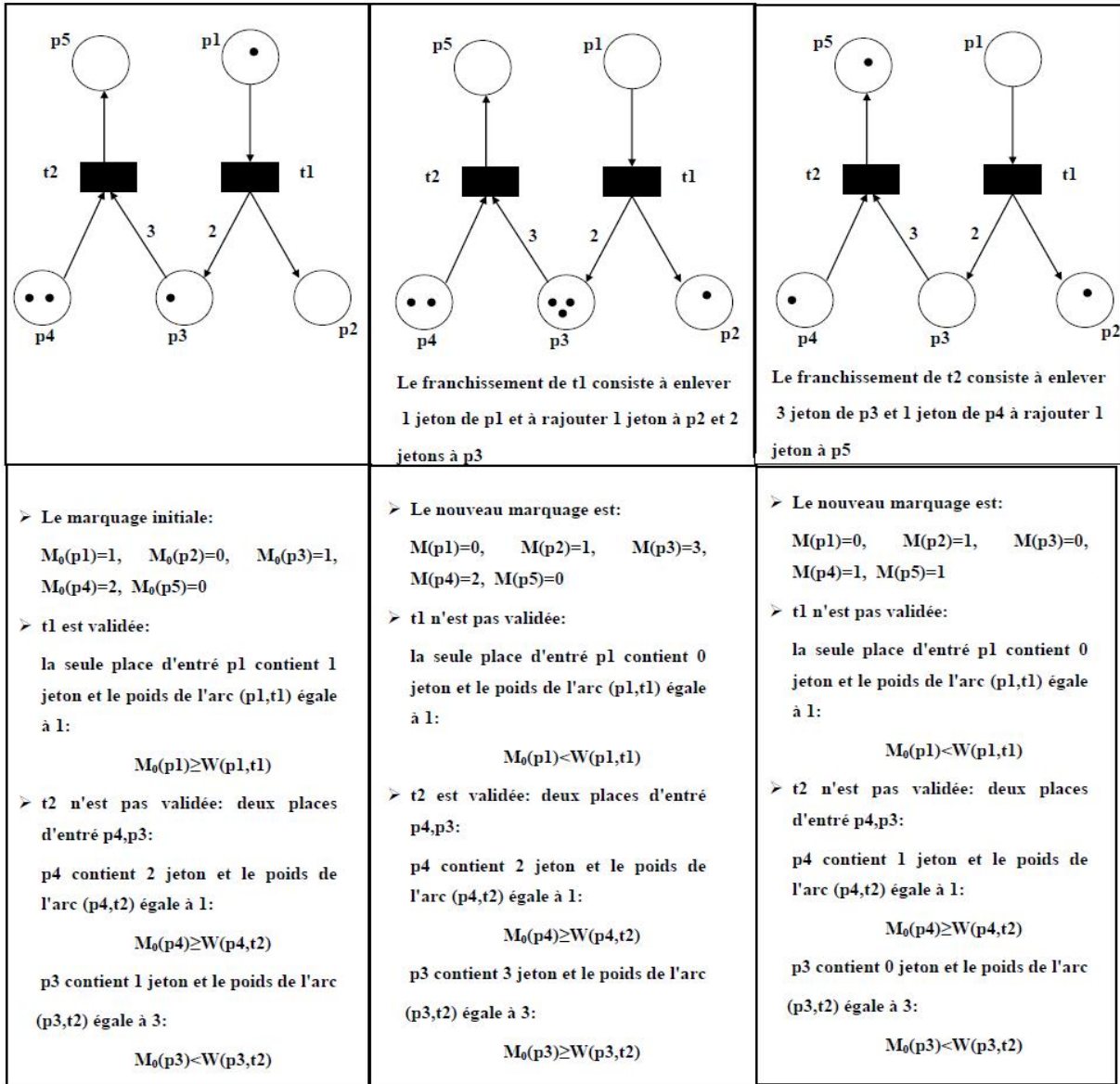


Figure 4.4: Evolution d'un réseau de Pétri

Lorsqu'une transition est validée, cela n'implique pas qu'elle sera immédiatement franchie ; Cela veut dire que les conditions nécessaires à son franchissement sont effectivement réunies. L'évolution du RdP se fait par le franchissement d'une seule transition à la fois. Quand plusieurs transitions sont simultanément validées, on ne peut pas savoir dans quel ordre elles seront effectivement franchies. L'évolution n'est donc pas unique.

4.4 Qualités des réseaux de Pétri

En tant que formalisme de spécification, les réseaux de Pétri présentent un certain nombre de qualités très importantes [109]:

- Ils disposent d'une définition formelle : ce caractère formel permet de produire des spécifications exemptes d'ambiguïté ; chaque construction des modèles possède une sémantique parfaitement définie.
- Ils présentent un grand pouvoir d'expression : les RdP sont notamment très bien adaptés à décrire des comportements complexes, réactifs ou concurrents.
- Ils sont exécutables : les modèles peuvent être interprétés par un programme construit à partir de la définition formelle de la notation, ce qui permet de simuler le fonctionnement du système en cours de spécification. Le modélisateur profite ainsi d'une vision dynamique du système qu'il spécifie pour approfondir la compréhension qu'il a de son comportement.
- Ils disposent de nombreuses techniques de vérification automatique des propriétés des modèles. Il est possible de rechercher des propriétés génériques telles que le caractère borné, vivant ou ré-initialisable.
- Ils disposent d'une représentation graphique attrayante, qui accroît la lisibilité et facilite la compréhension des modèles. Cette représentation graphique est également très utile lors de l'exécution interactive des modèles, servant alors de « débogueur » graphique.

4.5 Propriétés des réseaux de Pétri

La construction de la structure du RdP qui modélise le système réel est une tâche très importante. La tâche suivante qui est de même importance est l'étude ou l'analyse des propriétés de ce réseau. Il existe plusieurs propriétés, elles peuvent être classées en deux groupes [110],[111],[105]: les propriétés structurelles et les propriétés dynamiques.

4.5.1 Les propriétés dynamiques

Ces propriétés dépendent du marquage initial et sont liés à l'évolution du réseau (borné, vivant, réinitialisable...).

L'accessibilité est une base fondamentale pour l'étude des propriétés dynamiques de tout un système. Un marquage M est atteignable ou accessible depuis le marquage initial M_0 s'il existe une séquence des transitions franchissables S menant de M_0 à M .

4.5.1.1 Bornitude

Un RdP est borné si et seulement si toutes ses places sont bornées. Une place p est bornée si pour tout marquage accessible M , le nombre de jetons dans la place est inférieur à une constante k .

Le caractère borné d'un RdP renseigne sur les valeurs limites des ressources demandées par le système. Si un réseau est borné et la borne est égale à un, alors il est dit sauf.

4.5.1.2 Vivacité

L'évolution du marquage d'un RdP se fait par franchissement de transitions. Lorsqu'au cours de son évolution, certaines transitions ne sont jamais franchies, cela indique que l'évènement associé à la transition ne se produit pas et que le marquage d'une partie du RdP n'évolue pas. Cela indique que le sous-système modélisé par cette partie-là ne fonctionnera pas. Il y a donc un problème au niveau de la conception du système. L'idée est d'être capable de détecter systématiquement ce phénomène par l'analyse de propriétés du modèle RdP du système afin de disposer d'un outil d'aide à la conception des systèmes.

Une transition t est vivante pour un marquage initial M_0 si pour tout marquage M accessible à partir de ce marquage initial, il existe une séquence de franchissements à partir de M contenant t . Autrement dit, quelle que soit l'évolution, il existera toujours une possibilité d'arriver à franchir t .

Un RdP est vivant pour un marquage initial M_0 si toutes ses transitions sont vivantes pour ce marquage initial. Dans un tel réseau, on garantit que chaque transition est tirable éventuellement peu importe l'état du système. La propriété de vivacité est une propriété forte, souvent difficile à vérifier [105], [112].

4.5.1.3 Blocage

Un RdP est dit sans blocage pour un marquage initial M_0 si aucun marquage accessible n'est un blocage, autrement dit si le RdP peut continuellement évoluer. Un marquage M est un blocage si aucune transition n'est validée. D'une autre façon, on peut dire qu'un réseau de Pétri est sans blocage si et seulement si depuis tout marquage accessible il est possible de franchir au moins une transition.

4.5.1.4 Réinitialisabilité

Un RdP est réinitialisable (ou réversible) si et seulement si pour tout marquage M , il existe une séquence de transitions qui permet de revenir au marquage initial M_0 . Cette

propriété renseigne sur le fonctionnement répétitif, ce qui est pertinent pour la majorité des systèmes interactifs pratiques.

4.5.2 Les propriétés structurelles

Elles sont liées à la typologie du réseau et elles sont indépendantes du marquage initial du réseau. Parmi ces propriétés on trouve: p-invariant (ou p-semi-flot) et un t-invariant (ou t-semi-flot) [113],[114]. Leur but est de bâtir une passerelle entre la structure du réseau étudié et son comportement.

4.6 Les méthodes d'analyse des réseaux de Pétri

On aborde deux méthodes d'analyse : graphe de marquages et équation de matrice. La première méthode est de construire le graphe de tous les marquages de réseau, et on déduit des propriétés grâce aux techniques de théorie de graphe. La deuxième méthode est de trouver une représentation de matrice de réseau, et on utilise des techniques d'algèbre linéaire pour obtenir des propriétés du réseau.

La vérification des propriétés dynamiques se fait généralement en établissant l'arbre de couverture (ou le graphe de couverture). L'arbre de couverture est composé de nœuds qui correspondent aux marquages accessibles, et d'arcs correspondant aux franchissements de la transition qui fait passer d'un marquage à l'autre. Le nombre de nœuds dans cet arbre est fini. La construction de l'arbre de couverture permet de décider si un RdP est borné : on dit que la propriété de réseau borné est décidable. Dans ce cas, l'arbre sera appelé le graphe des marquages accessibles. A partir de ce graphe, on peut vérifier toutes les autres propriétés (vivacité, accessibilité d'un marquage, ...). Cependant, pour un RdP non borné, établir l'arbre de couverture ne suffit pas pour résoudre le problème d'accessibilité et le problème de vivacité. Ces deux problèmes sont également décidables, mais par des algorithmes plus compliqués. Malgré ceci, le graphe de couverture reste le seul outil pour vérifier les propriétés dynamiques d'un RdP pour un marquage initial donné[110],[111], [113].

L'analyse des propriétés structurelle repose essentiellement sur les techniques d'algèbre linéaire. En fait, la structure d'un RdP peut être définie par une matrice à éléments entiers, appelée matrice d'incidence.[113].

4.7 Structures fondamentales pour la modélisation des systèmes

Les RdP permettent de modéliser un certain nombre de comportements importants dans les systèmes : le parallélisme, la synchronisation, le partage de ressources...etc. [108]. Dans cette section, sont présentées des structures apparaissant dans un réseau de Pétri reproduisant ce type de comportements.

4.7.1 Parallélisme

Le parallélisme représente la possibilité que plusieurs processus évoluent simultanément au sein du même système. On peut provoquer le départ simultané de l'évolution de deux processus à l'aide d'une transition ayant plusieurs places de sortie. Pour cela, le RdP doit contenir la structure présentée dans la figure 4.5, gauche. Le franchissement de la transition T1 met une marque dans la place P2 (ce qui marque le déclenchement du processus 1) et une marque dans la place P3 (ce qui marque le déclenchement du processus 2). Il est ensuite possible de synchroniser l'achèvement des deux processus, voir figure 4.5, centre. La place P22 correspond à la fin du processus 1 et la place P23 à la fin du processus 2. Le RdP évoluera par franchissement de la transition T12. Pour cela, il est nécessaire que les places P22 et P23 contiennent chacune au moins un jeton, c'est-à-dire que les processus 1 et 2 soient terminés. Le RdP total est représenté figure 4.5 droite.

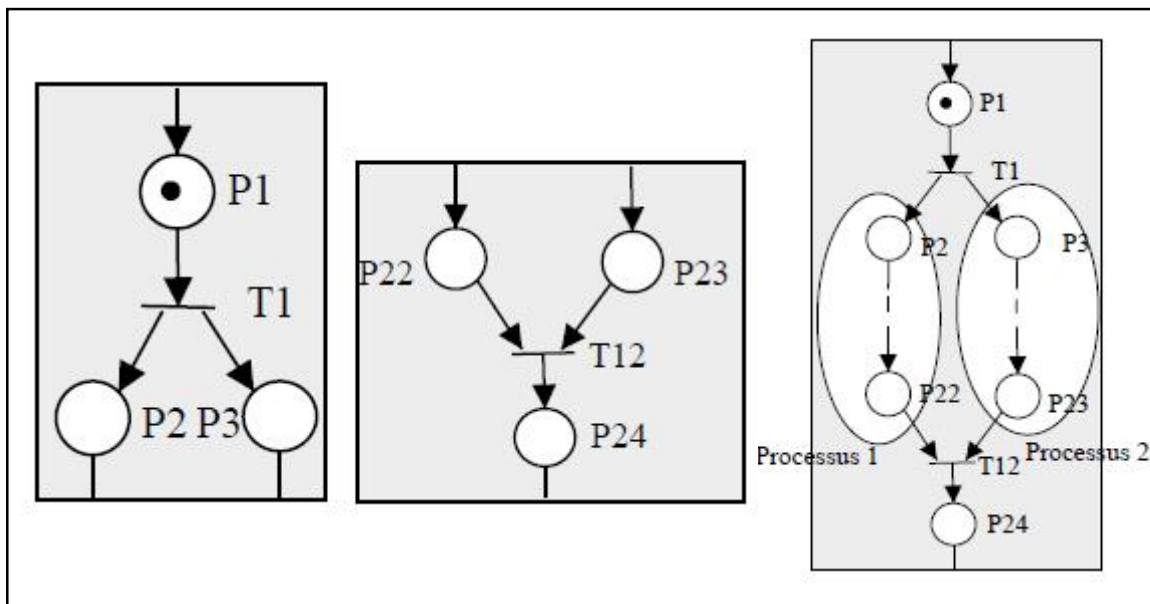


Figure 4.5: Parallélisme [108]

4.7.2 Synchronisation Mutuelle

La synchronisation mutuelle ou rendez-vous permet de synchroniser les opérations de deux processus. Un exemple est donné dans la figure 4.6. Le franchissement de la transition T7 ne peut se faire que si la place P12 du processus 1 et la place P6 du processus 2 contiennent chacun au moins une marque. Si ce n'est pas le cas, par exemple la place P12 ne contient pas de marque, le processus 2 est "bloqué" sur la place P6 : il attend que l'évolution du processus 1 soit telle qu'au moins une marque apparaisse dans la place P12.

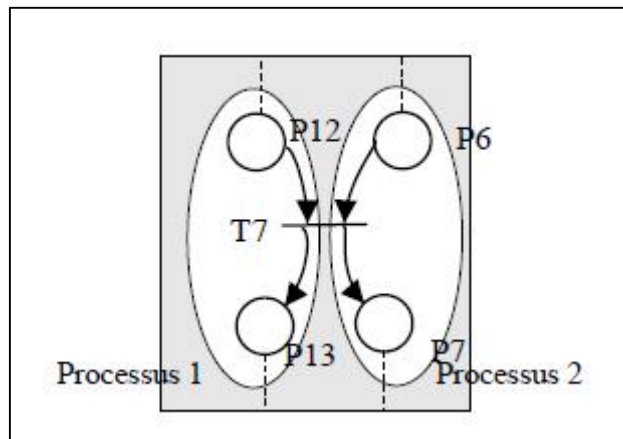


Figure 4.6: Synchronisation mutuelle [108]

4.7.3 Partage de ressources

Cette structure va modéliser le fait qu'au sein du même système plusieurs processus partagent une même ressource. Dans la figure 4.7, la marque dans la place P0 représente une ressource mise en commun entre le processus 1 et le processus 2. Le franchissement de la transition T17 lors de l'évolution du processus 1 entraîne la "consommation" de la marque présente dans la place P0. La ressource que constitue cette marque n'est alors plus disponible pour l'évolution du processus 2 puisque le franchissement de la transition T7 n'est plus possible. Lors de l'évolution du processus 1, lorsque la transition T18 est franchie, une marque est alors "redonnée" à la place P0 : la ressource redevient alors disponible pour l'évolution des deux processus.

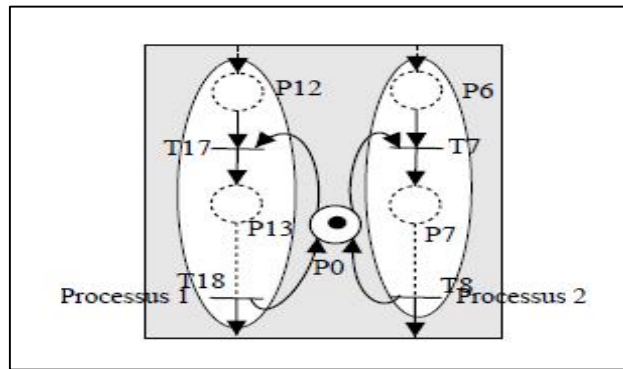


Figure 4.7: Partage de ressource [108]

4.8 Quelques extensions des réseaux de Pétri

Par RdP, il est possible de modéliser des comportements très divers ; synchronisation, parallélisme, partage de ressources, etc. Cependant cette modélisation souffre d'une limitation. En effet, la modélisation des systèmes réels mène parfois à des RdP de taille ingérable, ce qui diminue leur lisibilité et rend leur manipulation et même leur analyse difficile (problème d'explosion combinatoire). Pour apporter des solutions à cette limitation et augmenter la puissance de modélisation, des extensions du modèle RdP dites des RdP de haut niveau ont été proposées dans lesquelles d'autres aspects ont été pris en compte, tel que ; la colorisation, la temporisation, et autres. Dans ce qui suit nous explorons quelques extensions des RdP.

4.8.1 Réseaux de Pétri coloré

Les réseaux de Pétri colorés sont des réseaux de Pétri dans lesquels les jetons portent des couleurs. Une couleur est une information attachée à un jeton. Cette information permet de distinguer des jetons entre eux et peut-être de type quelconque [115]. Ainsi, les arcs ne sont pas seulement étiquetés par le nombre de jetons mais par leurs couleurs. Le franchissement d'une transition est alors conditionné par la présence dans les places en entrée du nombre de jetons nécessaires, qui en plus satisfont les couleurs qui étiquettent les arcs. Après le franchissement d'une transition, les jetons qui étiquettent les arcs d'entrée sont retirés des places en entrée tandis que ceux qui étiquettent les arcs de sortie sont ajoutés aux places en sortie de cette transition. Les réseaux colorés n'apportent pas de puissance de description supplémentaire par rapport aux réseaux de Pétri, ils permettent juste une condensation de l'information.

4.8.2 Réseaux de Pétri temporisés

Aucune durée n'est liée au franchissement des transitions et/ou au temps de séjour des marques dans les places en ce qui concerne les RdP. Cependant il y a beaucoup de systèmes à événements discrets dont l'évolution dépend du temps. D'autre part la notion de temps est capitale lorsque l'on veut évaluer les performances ou étudier les problèmes d'ordonnancement d'un système dynamique. La nécessité de modéliser et d'étudier de tels systèmes a donné naissance aux RdP temporisés (RdPT).

Le premier travail sur les RdPT a été réalisé par Ramchandani. Son modèle (les réseaux de Pétri t-temporisé (t-RdPT)) associe à chaque transition d'un RdP autonome un paramètre temporel dont la sémantique correspond à la durée de tir de cette transition. Un autre modèle a été proposé plus tard par Sifakis, : les réseaux de Pétri p-temporisés (p-RdPT). Un paramètre temporel est associé à chaque place. La sémantique de ce paramètre correspond au temps de séjour minimum d'une marque dans une place (temps d'indisponibilité). Sifakis montrera plus tard, que son modèle et celui de Ramchandani sont équivalents [113].

4.8.3 Les réseaux de Pétri t-temporels

Les RdPT ne permettent que de spécifier une durée minimale de traitement. Bien qu'ils permettent de modéliser un large nombre de systèmes manufacturiers, ils ne sont pas capables de modéliser des systèmes pour lesquels la durée de traitement est incluse entre une valeur minimale et une valeur maximale. Les RdP t-temporels (t-RdP) sont destinés principalement à l'étude des systèmes de télécommunication dont les évolutions dépendent des contraintes de type temps de réponse (time-out).

Dans ce modèle, un intervalle $[a, b]$ de temps est associé à chaque transition du réseau. L'intervalle associé à la transition t est relatif au moment où la transition devient validée. Supposons que t est validée à l'instant c , alors elle peut-être franchie seulement entre $a+c$ et $b+c$, sauf si elle devient non-validée à cause du franchissement d'une autre transition avec laquelle elle était en conflit [113].

4.8.4 Réseaux de Pétri synchronisés

Dans les modélisations RdP que nous avons vues précédemment, le fait qu'une transition soit franchissable indique que toutes les conditions sont réunies pour qu'elle soit effectivement franchie. Le moment où se produira le franchissement n'est pas connu. Un

RdP synchronisé est un RdP où à chaque transition est associée un événement. La transition sera alors franchie si elle est validée mais quand l'événement associé se produit.

Dans un RdP synchronisé, une transition validée n'est pas forcément franchissable. La transition est validée quand la condition sur les marquages est satisfaite. Elle deviendra franchissable quand l'événement externe associé à la transition se produit : elle est alors immédiatement franchie. Si en fonction du marquage de ses places d'entrée, plusieurs franchissements sont possibles, un seul se produira effectivement, celui dont l'événement associé se produit en premier [108].

4.8.5 Le paradigme (nets-within-nets) et les Réseaux de référence

Le paradigme des réseaux dans les réseaux (nets-within-nets), dû à Valk [116], [117], constitue la suite logique de ses travaux antérieurs portant sur les réseaux de flux de tâches (task-flow nets). Ce paradigme formalise le fait que les jetons dans un réseau de Pétri peuvent symboliser une structure de donnée complexe voire même un réseau de Pétri. Partant de ce principe, il est possible de modéliser des structures hiérarchiques d'une manière relativement simple mais très élégante. Dans ce qui suit, nous allons donner une brève introduction d'une implémentation de certains aspects des réseaux dans les réseaux appelés les réseaux de référence [118].

Les réseaux de référence [119] sont une notation graphique très appropriée à la description et à l'exécution des processus concurrents et complexes. Telle la majorité des autres formalismes Pétri, il existe un outil de simulation de ces réseaux appelé RENEW (Reference net workshop) [120]. Les réseaux de référence étendent les réseaux de Pétri classiques et les réseaux de Pétri colorés par l'introduction de nouvelles notions telles que : les instances réseaux, les réseaux considérés comme jetons objets, la communication via des canaux synchrones et l'utilisation de différents types d'arcs. En dépit de tout ceci, ces réseaux restent très similaires aux réseaux de Pétri colorés tels qu'ils ont été définis par Jensen [115]. Les définitions de ces extensions sont données dans ce qui suit [121], [122]:

- Les instances de réseaux: Selon le même principe d'instanciation d'un objet à partir d'une classe dans les langages de programmation objet, les instances de réseaux sont une copie instanciée à partir d'un réseau gabarit (un réseau moule). Notons que différentes instances d'un même réseau peuvent prendre des états différents à un instant donné et sont indépendantes les unes des autres dans tous leurs aspects.

- Le réseau est considéré comme jeton objet : les réseaux de référence implémentent le paradigme des réseaux dans les réseaux. Les places dans ces réseaux (appelés aussi réseaux systèmes) peuvent contenir des jetons schématisant un autre réseau (également appelé réseau objet). Partant de ce principe, on obtient facilement une hiérarchie de réseaux : un réseau système contenant un jeton schématisant un réseau objet peut lui-même être un jeton schématisant un réseau objet dans un autre réseau système.
- Les canaux synchrones: les canaux synchrones ont été proposés pour la première fois par Christensen et Hansen [123]. Ils permettent de synchroniser deux transitions pour les tirer automatiquement en même temps. Pour que cette synchronisation ait lieu, les deux transitions doivent porter le même nom de canal et le même nombre de paramètres. Notons qu'il est possible de synchroniser plus de deux transitions à un instant donné en les inscrivant, tout simplement, par plusieurs canaux synchrones.
- Les types d'arc : En plus des arcs usuels, les réseaux de référence offrent deux autres types d'arcs : les arcs de réservation et les arcs test. Ces deux types d'arcs sont similaires dans le sens où ils ne changent pas le marquage associé à une place. Les arcs de réservation sont en réalité une autre manière de représenter deux arcs classiques dont les directions sont opposées. Ils permettent de réserver un jeton lors du tirage d'une transition. Les arcs test sont dénués d'orientation et permettent d'accéder (tester) à un jeton donné.

4.8.5.1 Modélisation de la mobilité

Intuitivement, tels les jetons noirs ou les jetons colorés utilisés dans les réseaux de Pétri « classiques », les réseaux de référence autorisent une place à contenir un jeton symbolisant un autre réseau de Pétri. Voyons cette manière de faire sur les figures 4.8 et 4.9. D'abord, il est inutile de préciser qu'une telle représentation n'est pas très pratique lorsqu'il s'agit de modéliser un système assez complexe. Pour cela, l'outil de modélisation Renew utilise une sorte de pointeur référençant (d'où le nom : réseau de référence) la fenêtre contenant le réseau jeton. Comme il a été précisé précédemment, dans la terminologie des réseaux de référence, le réseau jeton est appelé réseau objet et le réseau le contenant est appelé réseau système.

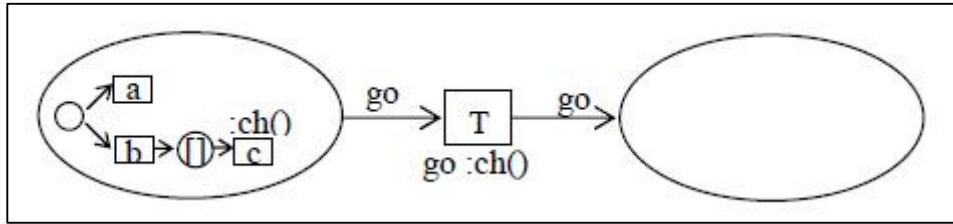


Figure 4.8: Un réseau objet contenu dans un réseau système [118]

Le réseau objet dans la place de gauche a trois transitions (a, b, c) et deux places, la première place est vide et la deuxième possède un jeton ("[]" veut dire jeton). La transition "c" est inscrite avec un canal synchrone ":ch()" (on peut considérer pour facilité la compréhension que ":ch()" est un autre nom pour la transition "c"). "go" est le nom de ce réseau objet. Le réseau système a une transition et deux places, la première place possède le réseau objet "go" et la deuxième est vide. Le mot « go » écrit sur l'arc de la transition "T" définit le type du jeton à retirer (l'arc en entré) et a mètre(l'arc en sortie), alors le franchissement de "T" provoque l'enlèvement du réseau objet "go" depuis la place gauche et le déposer dans la place droite. La transition "T" est inscrite par le canal "go:ch()". Ça veut dire que, pour franchir "T", il faut que la transition "c" soit validée (la transition "c" inscrit avec le canal ":ch()" est une précondition pour la transition "T"). Dans la figure 4.8 cela est possible et le tirage synchrone du réseau objet(transition "c") et du réseau système(transition "T") peut-être déclenché. Ceci nous mène à la situation décrite dans la figure 4.9 dans laquelle le réseau objet a été déplacé vers la place de droite de la figure. En même temps et de façon synchrone, le marquage du réseau objet a changé et désormais, aucun tirage supplémentaire de la transition "c" n'est plus possible.

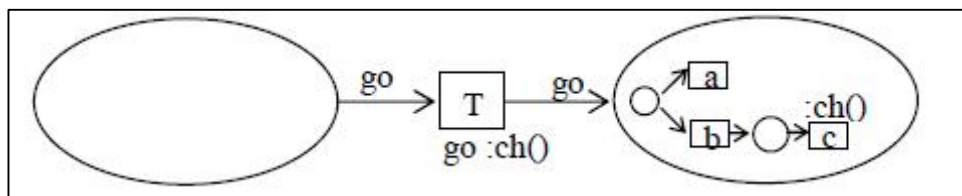


Figure 4.9: Les réseaux système et objet après tirage de la transition [118]

Cet exemple met en avant la manière avec laquelle sont exploitées les interactions entre le réseau objet et le réseau système pour modéliser une entité se déplaçant d'une position vers une autre. En effet, le réseau système peut offrir (ou éventuellement décliner) les possibilités de déplacement autorisant ainsi, à des instants opportuns, les mouvements du réseau objet en activant respectivement la transition inscrite par la pré-condition du canal synchrone (la transition "c") et la transition schématisant le mouvement dans le réseau

système (la transition "T"). En l'absence d'un tel point de vue (des jetons vus comme des réseaux), la personne chargée de la modélisation aura à modéliser ou à coder l'entité mobile en faisant appel à des structures de données d'un autre formalisme. Par conséquent, les actions internes de ces entités mobiles vont être négligées lors de cette étape et vont être reléguées à la personne chargée de la modélisation du réseau système enfreignant totalement la réalité. En utilisant le paradigme des réseaux dans les réseaux, on peut aisément harmoniser la concurrence du système et de l'entité mobile dans un seul et unique modèle sans pour autant désobéir aux obligations d'abstraction [124].

Par ailleurs, la majorité des travaux de recherche relatifs aux agents mobiles distingue, principalement, entre une mobilité simple et une mobilité totale [125] ou bien entre une faible mobilité et une forte mobilité [126]. Contrairement à ces classifications, les interactions entre le réseau objet et le réseau système font ressortir, pour un agent donné, quatre possibilités de se déplacer ou de se faire déplacer, tableau 4.2.

Réseau Objet	Réseau système	Types de mobilité	Commentaires
Non impliqué	Non impliqué	Mouvement spontané	Aucune influence du réseau objet et du réseau système sur le mouvement. Aucune pré ou post condition n'est nécessaire.
Non impliqué	impliqué	Mouvement Objectif (transportation)	Le réseau système contrôle le mouvement, le réseau objet est transporté d'une position à une autre.
Impliqué	Non impliqué	Mouvement Subjectif	Difficile à imaginer dans la pratique, mais théoriquement un réseau objet peut contrôler le mouvement.
Impliqué	impliqué	Mouvement Consensuel	Les deux réseaux ont une influence sur le mouvement

Tableau 4.2: Les types de mobilité [118]

4.9 Outils de modélisation des réseaux de Pétri

L'aspect formel des RdP a encouragé les développeurs à mettre au point une multitude d'outils de simulation et de vérification des RdP selon la technique de vérification de modèle (CPNTools, CPNAMI, PROD, JARP, MARIA, LOLA, Petri Net Kernel, GreatSPN, INA, Artifex, ExSpect, FLOWer, f-net, GD ToolKit, Helena, HPSim, INA, JARP, JFern, JPetriNet, Opera, ORIS, PACE, RENEW, ...etc.). La plupart de ces outils présente un environnement graphique d'édition des RdP avec la possibilité de simuler le modèle et d'analyser des propriétés génériques des RdP. Tous les détails sur ces outils et autres outils peuvent être trouvés dans [127].

4.10 Utilisation des réseaux de Pétri dans le domaine des SMA

De nombreux travaux ont été réalisés autour de l'utilisation des RdP pour représenter les modèles multi-agents. Il s'agit, de modéliser l'entité agent moyennant les RdP, les interactions entre agents, l'aspect social (organisation), la migration des agents mobiles,...etc.

4.10.1 Modélisation des agents mobiles

D'autres travaux comme [128], [124] ont été menés sur l'utilisation de RdP pour modéliser les agents mobiles. Les agents sont représentés par des jetons dans des places qui représentent des locations, les mouvements possibles des agents correspondent au franchissement des transitions.

4.10.2 Modéliser l'aspect social

En contraste avec Ferber, qui utilise le RdP coloré pour représenter le fonctionnement d'un agent et d'un SMA, Weyns et Holvoet [129] utilisent un RdP coloré pour modéliser et étudier le comportement social d'agents. Köhler et ses collègues [130] proposent la modélisation de l'aspect social dans un SMA par les RdP. Les deux vues du system niveaux macro/ micro sont représentées par les RdP pour indiquer le lien entre les deux niveaux macro/micro. Les auteurs utilisent une variante de RdP de haut niveau où les jetons d'un réseau correspondent aux RdP sur un niveau inférieur.

4.10.3 Modélisation des interactions

Les RdP ont été largement utilisés pour l'analyse, la modélisation et la validation des interactions entre agents. Les travaux [131], [132], [133] proposent la spécification de protocoles d'interaction par RdP Colorés. Dans les travaux d'El-Fallah et ses collègues, les modèles obtenus ont servis comme des filtres pour la reconnaissance ensuite l'apprentissage des schémas d'interaction. [134] propose une approche vise à analyser les interactions entre agents au cours de leur exécution en modélisant les protocoles par les RdP.

4.11 Conclusion

On a essayé de présenter dans ce chapitre le paradigme de réseau de Pétri et ses extensions, en commençant par la modélisation par Réseaux de Pétri dans le cadre général, ensuite ces extensions sont présentées, parmi ces extensions on trouve les réseaux de

référence. Finalement une présentation de quelque travaux qui ont utilisé les réseaux de pétri pour la modélisation des systèmes multi-agents est exposée.

Maintenant, Après avoir expliqué le principe des réseaux de références dans ce chapitre, On peut discuter sur l'approche de test basé modèle des agents mobiles utilisant le paradigme des réseaux dans les réseaux, et donc présenté notre propre algorithme de concrétisation, ainsi c'est le but du prochain chapitre.

Chapitre 5: Test basé modèle des agents mobiles utilisant le paradigme des réseaux dans les réseaux

L'approche de test basé sur le modèle des agents mobiles est une approche basée sur le MBT (model-based testing), qui est une technique de génération automatique des cas de test abstraits à partir de la modélisation du comportement du système sous test. Pour que ces cas de test abstraits puissent être soumis au système sous test, ils doivent être transformés en des cas de test concrets. Dans cette approche la modélisation est fondée sur les réseaux de références, mais au niveau de la concrétisation il y a des ambiguïtés; donc nous essayons dans ce chapitre de proposer un algorithme clair de concrétisation.

5.1 Test basé modèle des agents mobiles utilisant le paradigme des réseaux dans les réseaux

L'approche de test des SMA[135], discutant dans ce chapitre, est une approche de test basé modèle pour les agents mobiles. La technique de test basée sur le modèle[136] implique habituellement quatre étapes : (a) établir un modèle abstrait du système sous test, (b) valider le modèle (typiquement par l'intermédiaire de l'animation), (c) produire des tests abstraits du modèle et (d) concrétiser des tests abstraits (raffiner ces tests d'abstrait en des tests exécutables concrets). Après ceci, les tests concrets peuvent être exécutés sur le système sous test, afin de détecter des anomalies (où les sorties du système sous test sont différentes à celles prévues par les tests). Ces quatre étapes sont soutenues dans cette approche et les détails sont comme suit :

- Établissement d'un modèle abstrait du système sous test : cette étape est basée sur le modèle des réseaux de référence[119] qui est basé sur le paradigme des réseaux dans les réseaux[116].
- Validation du modèle : Telle la majorité des autres formalismes Pétri, il existe un outil de simulation de ces réseaux appelé Renew (Reference net workshop) [120], un tel outil, robuste et facile à manipuler, permet de réduire considérablement l'important effort initial en termes d'heures-homme nécessaires à la construction et la validation du modèle de test.

- Génération et production des cas de tests abstraits : par l'utilisation de Renew, il est possible de récupérer toute la trace de simulation sous forme d'un fichier texte. Les étapes de la simulation écrites dans le fichier de trace de simulation sont considérées comme étant les cas de tests abstraits.
- Concrétisation des cas de test abstraits : Pour établir le lien d'abstraction entre le modèle de test et le système sous test, un agent spécial appelé « agent testeur » est employé, il traduit les cas de test abstraits à une série de messages codés selon les spécifications de structure de message de FIPA ACL [137].

Le but de ce document est la construction et donc la validation de l'agent testeur, il s'agit particulièrement de l'étape de la concrétisation des cas de tests abstraits.

5.1.1 Modélisation

Le paradigme des réseaux dans les réseaux (nets-within-nets), formalise le fait que les jetons dans un réseau de Pétri peuvent symboliser une structure de donnée complexe voire même un réseau de Pétri. Partant de ce principe, il est possible de modéliser des structures hiérarchiques d'une manière relativement simple mais très élégante. Une implémentation de certains aspects des réseaux dans les réseaux appelés les réseaux de référence [118].

Les réseaux de référence sont une notation graphique très appropriée à la description et à l'exécution des processus concurrents et complexes, telle la majorité des autres formalismes Pétri, il existe un outil de simulation de ces réseaux appelé Renew (Reference net workshop). Les réseaux de référence étendent les réseaux de Pétri classiques et les réseaux de Pétri colorés par l'introduction de nouvelles notions telles que : les instances réseaux, les réseaux considérés comme jetons objets, la communication via des canaux synchrones et l'utilisation de différents types d'arcs. Les définitions de ces extensions sont données dans le chapitre précédent. En utilisant le paradigme des réseaux dans les réseaux, on peut aisément harponner la concurrence du système et de l'entité mobile dans un seul et unique modèle sans pour autant désobéir aux obligations d'abstraction.

Pour la modélisation via l'utilisation du paradigme des réseaux de référence, nous allons faire appel à l'architecture MULAN (Multi-Agent Nets) [138], [124], utilisée pour décrire l'aspect hiérarchique des systèmes multi-agents. Mulan a la structure générale comme représenté sur la figure 5.1 : Chaque boîte décrit un niveau d'abstraction en termes de réseau

système. Chaque réseau système contient des réseaux d'objets, que la structure est visible par les lignes de Zoom.

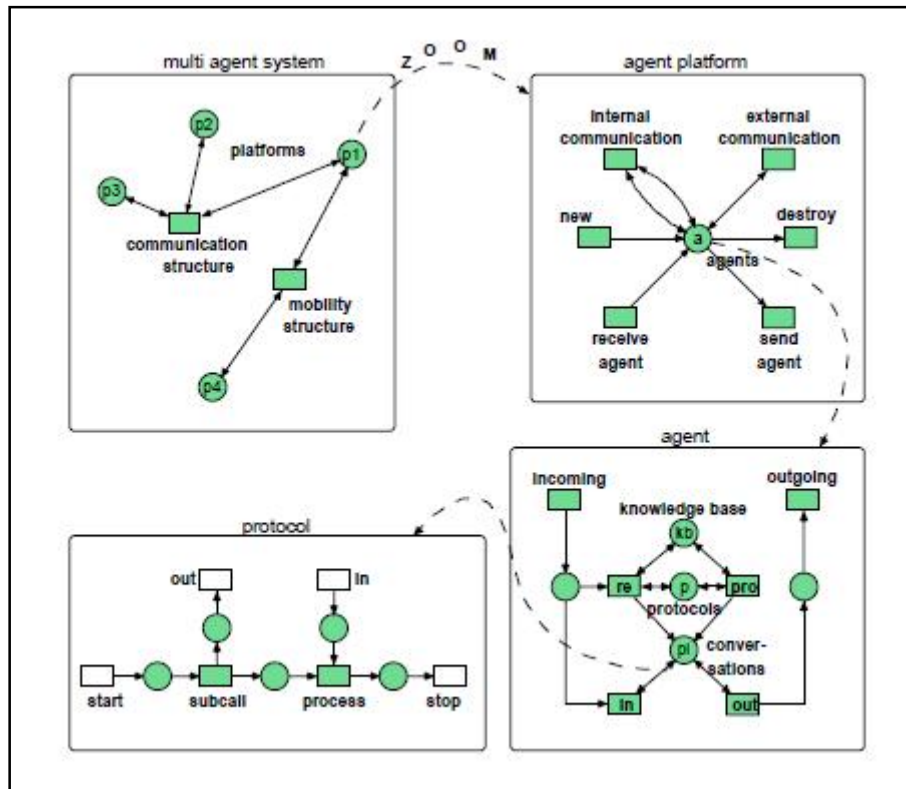


Figure 5.1: Système multi-agents modélisé avec les réseaux dans les réseaux[124]

Lors de la modélisation des systèmes complexes, il est fortement déconseillé de vouloir gérer toute la complexité du système au moment de sa modélisation et/ou de son exécution. Ainsi, on introduit la notion de vue d'un système. Plusieurs vues sont possibles : le système multi-agents dans sa totalité, les plateformes sur lesquelles les agents sont positionnés, l'agent lui-même ou tout simplement son comportement [118]. Cette notion est prise en compte par Mulan comme le montre la figure 5.1.

5.1.2 Validation du modèle

Plus qu'un outil d'édition des réseaux de référence, Renew est également un simulateur très puissant. Ainsi, l'ingénieur de test (chargé de la modélisation) peut de manière dynamique et interactive explorer les états du modèle qu'il a mis au point. Il peut, par exemple, imaginer différents scénarios d'exécution. Tout en simulant le comportement du modèle pour chacun des scénarios d'exécution imaginés, l'ingénieur de test pourra, en fonction des résultats obtenus, corriger et/ou raffiner progressivement le modèle abstrait de test [118].

5.1.3 Simulation et génération des cas de tests abstraits

Avec Renew, il est possible de récupérer toute la trace de simulation sous forme d'un fichier texte. En réalité, chaque fois qu'une transition est tirée, son nom tel qu'il a été utilisé dans le réseau, est écrit dans la trace de simulation. Les noms des places sont également écrits dans cette trace de simulation à chaque fois qu'un jeton est retiré ou déposé dans la dite place. On obtient très vite une suite d'étapes de simulation (une étape est toutes les transitions franchies au même temps avec leurs entrées et sorties) dont chacune est composée des noms des places en entrée avec leurs marquages, de la liste des transitions tirées (tirées au même temps) et de la liste des places en sortie avec leurs marquages courants. Cette combinaison (place en entrée, transitions tirées et places en sortie) telle qu'elle a été écrite dans la trace de simulation est considérée comme étant le cas de test abstrait et doit être (après concrétisation) soumis au système sous test. Excepté la première étape de la trace de simulation qui est consacrée à l'initialisation de système et pour instancier automatiquement le premier réseau, toutes autres étapes de simulation suivent la forme du tableau 5.1.

Fichier de trace de simulation	remarques
(numero_etap -1).....	l'étape précédente
(numero_etap)----- Synchronously -----	début de l'étape
(numero_etap) Removing jeton in nom_reseau[numero_instance_reseau].nom_place	on peut trouver 0...N
(numero_etap) Firing nom_reseau[numero_instance_reseau].nom_transition	on peut trouver 1...M
(numero_etap) New net instance nom_reseau[numero_instance_reseau] created.	on peut trouver 0...L
(numero_etap) Initializing jeton into nom_reseau[numero_instance_reseau].nom_place	if L>0 on peut trouver 0...P
(numero_etap) Putting jeton into nom_reseau[numero_instance_reseau].nom_place	on peut trouver 0...Q
(numero_etap+1)----- Synchronously -----	début de l'étape suivante
(numero_etap +1).....	

Tableau 5.1: La forme d'une étape de simulation (cas de test abstrait)

Depuis ce tableau nous pouvons voir la forme d'une étape de simulation ou comme elle est considérée précédemment, on peut voir la forme d'un cas de test abstrait, qui contient six lignes, notant que les cas de test abstrait ne contiennent pas obligatoirement toutes ces lignes comme mentionner dans la colonne "remarques". Les mots écrits en gras sont des mots-clés de simulation ("Removing"=enlever, "Firing"=tiré, "New net instance"=nouvelle

instance de réseau, "Initializing"=initialiser, "Putting"= mettre). Le "numero_etap" représente le numéro de l'étape de simulation (un nombre séquentiel), "nom_place" et "nom_transition" représentent le nom d'une place et d'une transition respectivement, comme elles sont spécifiées dans leurs réseaux. La ligne "Initialising" apparaît quand il y a au minimum la ligne "New net instance" ; son travail dans la simulation est d'initialiser le marquage des places. Quand une simulation d'un réseau est commencée, Renew le simulateur crée une instance de ce réseau. Le "nom_réseau[numero_instance_reseau]" représente le nom d'une instance, il se compose de nom de réseau ainsi qu'une numérotation entre crochets, notant que chaque instance d'un réseau a son numéro unique. Ce numéro est produit pour la première fois, quand l'instance est créée, avec précision dans la ligne de "New net instance" dans le fichier de simulation.

5.2 Concrétisation des cas de test abstraits

Pour établir le lien d'abstraction entre le modèle de test et le système sous test, un agent spécial appelé « agent testeur » est employé, il traduit les cas de test abstraits à une série de messages codés selon les spécifications de structure de message de FIPA ACL. Ces messages représentent les cas de test concrets. L'agent testeur prend comme entrées le système sous test (son implémentation) et le fichier de trace de simulation (cas de test abstraits), les sorties sont des cas de test concrets qui vont être exécutés sur le système sous test. Le travail de l'agent testeur est divisé en plusieurs étapes [135] : analyse statique, l'instrumentation, la partie de construction des cas de test concrets (le prologue, le corps de test, le verdict et l'épilogue), l'exécution de système et finalement, l'évaluation de test. Notant que dans ce travail, les deux dernières étapes sont assemblées avec les autres étapes.

5.2.1 Analyse statique

La phase d'analyse statique permet à l'agent testeur d'avoir un point de vue global sur l'organisation interne des agents sous test. Exactement nous cherchons à trouver comment le programmeur a employé les mots clés dans son programme. Les mots clés sont identiques à ceux employés dans les réseaux simulés. Ils sont chaque nom des places et des transitions qui ont été employés dans les réseaux.

Dans la première étape, l'agent testeur cherche à trouver, pour tous les agents sous test : chaque mot clé dont lequel nous pouvons accéder depuis cet agent, la nature de ce mot clé

(une variable ou une méthode (avec ses paramètres et ses types) ou un comportement) et en fin où ce mot clé s'est appelé (des méthodes ou des comportements).

Dans la deuxième étape, l'analyse statique doit spécifier quelques informations sur le nom du premier réseau. L'agent testeur doit définir comment ce nom est implémenté : comme nom de classe simple (avec son constructeur, les paramètres et ses types) ou comme nom de classe d'agent (avec ses arguments). L'agent testeur utilise un autre agent dans son travail appelé "agent créateur", son travail essentiel est de créer (après réception d'un message "request" de l'agent testeur) l'instance d'agent ou l'objet de classe qui ont le même nom que celui du premier. Si celui-ci est employé dans l'implémentation sous test comme nom de classe simple, la première étape de l'analyse statique sera appliquée pour l'agent créateur aussi parce que par la création de l'objet de la classe qui a le même nom que le premier réseau, l'agent créateur peut accéder à toutes les variables et les méthodes de cette classe. Toutes les informations (obtenues en première et deuxième étape) sont alors stockées dans un fichier appelé "le fichier d'informations d'agent".

5.2.2 Instrumentation

Pour permettre aux agents sous test de communiquer avec l'agent testeur, le module d'instrumentation a comme tâche l'adjonction dans le code source initial d'un certain ensemble de routines déterminées (primitives d'instrumentation), lesquelles en s'exécutant et sans modifier le comportement initial du programme (sauf en ce qui concerne les aspects occupation mémoire et temps d'exécution).

L'agent testeur va insérer un comportement dans chaque code source des agents du système sous test et même dans le code source de l'agent créateur, le travail de ce comportement est d'envoyer un message à l'agent testeur, ce message peut être : un message "inform" contenant la confirmation de la création de l'instance de l'agent lui-même, soit un message "inform" contenant la valeur d'une variable, soit un message "agree" après l'exécution d'une méthode ou un comportement, soit, dans le cas de l'agent créateur, un message "agree" après la création de l'instance d'agent ou de l'objet de la classe qui a le même nom que le nom du premier réseau. Les variables, les méthodes et les comportements concernés sont ceux stockés dans le fichier d'informations d'agent. Notant que, tout le travail des agents sera à travers ce comportement.

5.2.3 La construction des cas de test concrets

Cette étape représente le vrai début de l'étape de concrétisation. On distingue différentes parties. Ces parties sont : le prologue, le corps de test, le verdict et l'épilogue.

- Le prologue : au tout début, le prologue initialise le système sous test. Après l'avoir exécuté on peut dire que le système a atteint l'état considéré comme point d'origine du cas de test actuel.
- Le corps de test : il contient les opérations et les stimuli correspondants à l'objectif du test en cours
- Le verdict : définit des critères qui déterminent, en fonction des sorties ou des états observés du système sous test, si les tests passent ou échouent.
- L'épilogue : à la fin, l'épilogue libère le système sous test dans un état final bien défini. Plus précisément, l'épilogue prépare le système pour un nouveau cas de test ou, dans certain cas, libère le système sous test.

Dans ce qui suit on va voir comment ces étapes sont générées.

5.2.3.1 Génération de la partie prologue

Quel que soit le scénario imaginé et simulé par le testeur, les premières lignes du fichier de la trace de simulation (numérotés par le chiffre 1) seront toujours considérées comme étant la partie prologue. Ainsi, une analyse textuelle de ces lignes permettra de déterminer les principales caractéristiques du système. Utilisant ces informations et le fichier d'informations d'agent, l'agent testeur envoie un message à l'agent créateur lui demandant de créer l'instance d'agent ou l'objet de la classe qui a le même nom que le nom du premier réseau et attend la réception du message "agree". Deux cas sont possibles : (1) l'agent testeur reçoit le message "agree" seulement, alors il sauvegarde l'identificateur de l'agent créateur ; (2) l'agent testeur reçoit avec le message "agree", un message "inform" contenant la confirmation de la création de l'instance d'un agent (naturellement l'expéditeur sera l'instance d'agent qui a le même nom que le nom du premier réseau), alors l'agent testeur sauvegarde l'identificateur de l'expéditeur.

5.2.3.2 Génération de la partie corps de test et du verdict

Après la génération de la partie prologue, l'agent testeur est maintenant prêt à établir le corps de test et le verdict de chaque cas de test abstrait. Il lit le fichier de trace de simulation pour

chaque étape de simulation (le lecteur peut suivre la forme d'une étape de simulation du tableau 5.1), il va faire le suivant (dans le même ordre):

1. Pour chaque ligne "New net instance", l'agent testeur doit définir un groupe pour la nouvelle instance du réseau qui apparaît dans cette ligne, en la sauvegardant dans le groupe de l'instance du réseau que le franchissement de sa transition a fait la création de cette nouvelle instance du réseau, produisant un groupe de subordonné. Notant que l'instance du premier réseau est un membre dans tous les groupes.
2. Pour chaque ligne "Removing", l'agent testeur envoie un message "request" à l'instance de l'agent sous test, pour récupérer la valeur de la variable qui a le même nom que "nom_place", pour la comparer au jeton respectif qui représente la valeur de "nom_place" dans la simulation.
3. Pour chaque ligne "Firing", l'agent testeur envoie un message "request" à l'instance de l'agent sous test, lui demande d'exécuter la méthode ou le comportement qui a le même nom que le "nom_transition", ensuite l'agent testeur attend la réception du message "agree" ou "refuse" ou "not understood". S'il y a beaucoup de "Firing" à la même étape, l'agent testeur doit contrôler à partir de fichier d'informations d'agent si les méthodes ou les comportements correspondants sont dans une relation d'appelle, si c'est le cas; l'agent testeur doit éviter la répétition. De plus, si l'agent testeur a trouvé déjà dans cette étape, la ligne de "New net instance", et s'il reçoit des messages "inform" contenant la confirmation de la création d'une instance d'agent, il sauvegarde les identificateurs de ces expéditeurs dans le groupe de la nouvelle instance du réseau. Notant que l'agent testeur ajoute à chaque groupe l'identificateur sauvegardé dans la partie prologue.
4. Pour chaque ligne "Putting", l'agent testeur envoie un message "request" à l'instance de l'agent sous test, pour récupérer la valeur de la variable qui a le même nom que "nom_place", pour la comparer au jeton respectif qui représente la valeur de "nom_place" dans la simulation.

Chaque message FIPA-ACL a un expéditeur (son identificateur) et un récepteur ou une liste de récepteurs (leurs identificateurs), l'expéditeur des messages dans "Removing", "Putting" et "Firing", est l'agent testeur mais qui est (ou qui sont) le récepteur. Pour chaque "Removing", "Putting" et "Firing" il y a l'instance d'un réseau

"nom_réseau[numero_instance_reseau]" dans le fichier de simulation, et pour chaque "nom_réseau[numero_instance_reseau]" il y a au minimum un identificateur d'agent dans son groupe, obtenu à la création du "nom_réseau[numero_instance_reseau]" lui-même ou à la création de n'importe quel autre instance d'un autre réseau du même groupe. Prenant cette identificateur d'agent (ou les identificateurs des agents) et utilisant le fichier d'informations d'agent qui définit pour chaque agent sous test ses possibilités d'accès aux mots clés ; l'agent testeur peut savoir pour quel agent il va envoyer son message. Autrement un test est échoué si l'agent testeur ne trouve aucun identificateur d'agent à laquelle il peut accéder à ce mot clé.

Grâce à l'instrumentation, l'agent sous test concerné peut maintenant exécuter la partie instrumentée (le comportement supplémentaire) et renvoyer à l'agent testeur les états qu'elles ont atteints. Maintenant nous pouvons dire qu'un corps de test d'un cas de test abstrait est tous les messages FIPA-ACL que l'agent testeur obtient à partir de la transformation des lignes "Removing", "Putting" et "Firing" d'une étape de simulation. Le résultat prévu après l'exécution du corps de test courant par un agent sous test est également dérivé à partir du fichier de trace de simulation. Plusieurs scénarios sont possibles. L'agent testeur décide que le test échoue si les valeurs de variables contenues dans la réponse de l'agent diffèrent de celles contenues dans le fichier de trace de simulation. Le succès ou l'échec des tests peut être vérifié également selon la réponse de l'agent sous test après un message "request" explicite de l'agent testeur (le test échoue si la réponse de l'agent est de type "refuse" ou "not understood").

5.2.3.3 Génération de la partie épilogue

Tant que la fin de fichier de trace de simulation n'a pas encore été atteinte, l'agent testeur produira le prochain corps de test, l'envoie, reçoit des réponses et les compare au verdict. Autrement, il envoie des messages pour la libération et la destruction des instances des agents sous test et de l'instance de l'agent créateur, ainsi il fera toutes les actions nécessaires pour l'arrêt du processus de test.

5.2.4 Exécution du système et évaluation des résultats

Comme il a été mentionné dans le début de la section 5.2, l'exécution de système et l'évaluation de test sont assemblées avec les autres étapes. Ici nous devons savoir que

la concrétisation est l'obtention des cas de test concrets seulement, dans notre cas, les cas de test concrets sont des messages FIPA-ACL avec leur verdict, "l'envoi" de ces messages aux agents correspondants représente l'étape d'exécution de système, et "la comparaison" de la réponse d'agents avec le verdict et « la décision » du passage ou non du test représentent l'étape d'évaluation de test. Ceci n'interdit pas l'indépendance, ce qui signifie que l'agent testeur peut produire tous les cas de test concrets et ensuite il les envoie. Tout le travail de l'agent testeur est schématisé dans la figure 5.2, et il peut-être récapitulé par l'algorithme suivant :

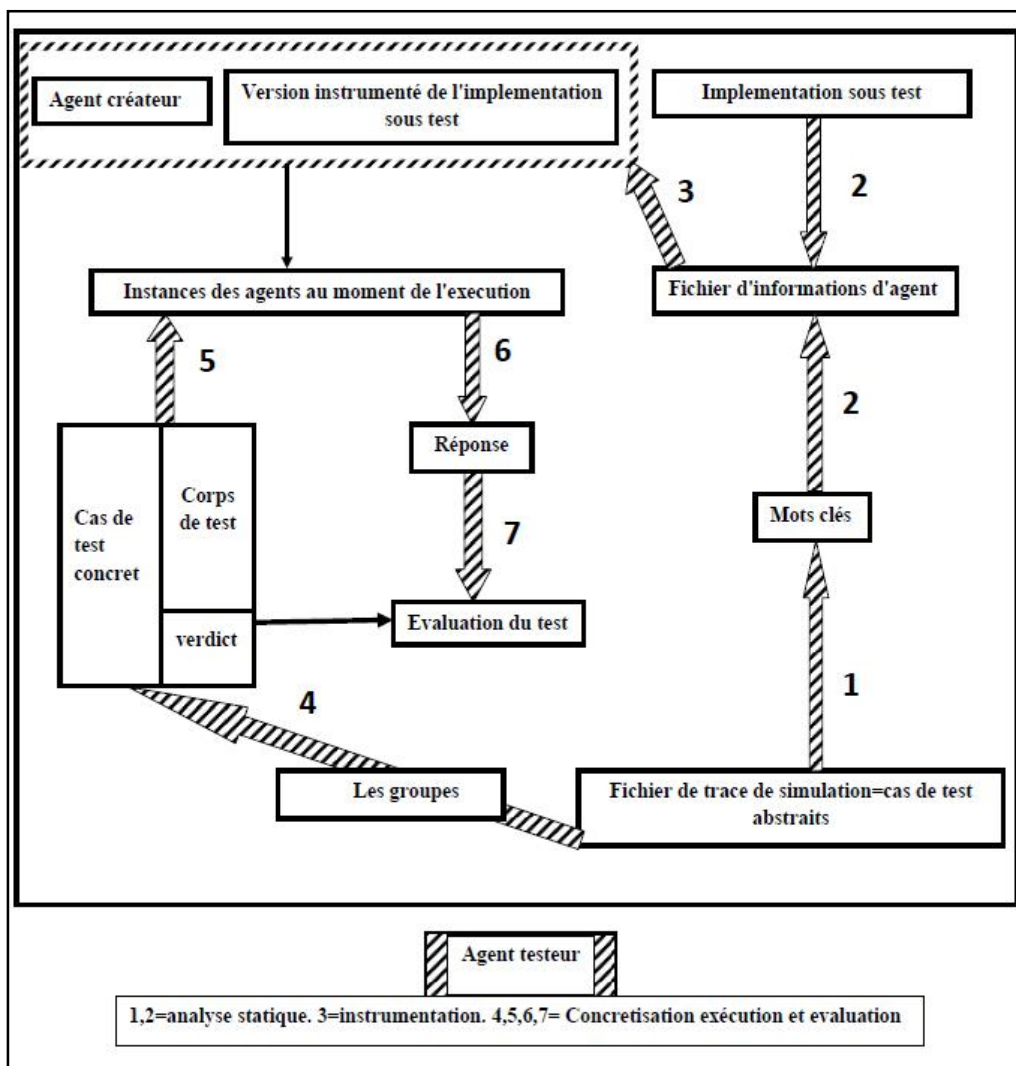


Figure 5.2: Concrétisation des cas de test abstraits (travail de l'agent testeur)

1. Commencer l'analyse statique et remplir fichier d'informations d'agent.
2. Commencer l'instrumentation pour générer la version instrumentée de l'implémentation.
3. Lire les étapes de simulation à partir du fichier de trace de simulation :

- (a) Si c'est la première étape de simulation, envoyer un message à l'agent créateur pour lui demander de faire son travail.
- (b) Pour chaque étape de simulation excepté la première étape faire le suivant:
 - (i) S'il y a "New net instance":
 - (1) défini un groupe pour la nouvelle instance de réseau.
 - (ii) Pour chaque "Removing", "Firing" et "Putting" (avec cet ordre) :
 - (1) choisis un identificateur d'agent du groupe de l'instance du réseau concerné.
 - (2) envoie un message à l'instance d'agents correspondant et attends :
 - (3) soit la valeur d'une variable, et la compare au jeton respectif, soit un message "agree" ou "refuse" ou "not understood".
 - (4) si la ligne de "New net instance" apparaît dans cette étape, il faut sauvegarder tous les identificateurs d'expéditeurs des messages contenant la confirmation de la création d'une instance d'agent dans le groupe de la nouvelle instance de réseau.
 - (iii) Édition de rapport de test : Si les valeurs sont égales ou l'agent testeur reçoit un message "agree" le test est passé, autrement si les valeurs sont différentes ou l'agent testeur reçoit un message "refuse" ou "not understood" le test a échoué.
4. Si la fin du fichier de trace de simulation n'est pas atteinte, retourne à l'étape 3, autrement le test est fini.

5.3 Conclusion

Ce Chapitre présente une description de l'étape de concrétisation des cas de test abstraits, qui est l'étape la plus difficile dans l'approche de test basé modèle des agents mobiles. On a commencé par la présentation des différentes étapes qui précèdent celle de la concrétisation, ensuite on l'a expliquée. Il nous reste à l'appliquer et l'implémenter ce qui est l'objectif du chapitre suivant.

Chapitre 6: Etude de cas et implémentation

Dans ce chapitre on va valider notre algorithme proposé, pour cela on va appliquer l'approche de test basé sur le modèle des agents mobiles, en utilisant comme un cas d'étude le problème de transport de ressources par des robots.

6.1 le problème de transport de ressources par des robots

Le problème de transport de ressources par des robots [139] est une tâche classique en robotique collective, souvent inspirée par des mécanismes mis en place par les insectes sociaux pour s'adapter aux conditions de leur milieu. L'environnement est présenté par deux salles séparées par plusieurs couloirs étroits (voir la figure 6.1). La salle de dépôt (l'espace blanc mentionné avec le nombre "2" dans la figure) et la salle de retrait (l'espace blanc mentionné avec le nombre "1" dans la figure) ont respectivement la zone de dépôt (rectangle gris mentionné avec nombre "4" dans la figure) et la zone de retrait (rectangle gris mentionné avec nombre "3" dans la figure). Ces zones sont au fond des salles, contre le mur qui est en face les couloirs, afin de forcer les robots (représentés par des triangles dans la figure) pour bien rentrer dans la salle et prendre ou déposer des ressources (représentées par des cercles dans la figure). Les couloirs "5", "6", "7" et "8" sont trop étroits pour que deux robots puissent y passer côte à côte. Une fois engagé dans un couloir, que doit faire un robot rencontrant un autre robot circulant en sens inverse ? Ce problème a été résolu par plusieurs méthodes ; voir [139], [140]. Cette situation est évitée dans ce travail. Un robot peut passer à travers un couloir seulement s'il est vide, ou s'il y a un robot dans le couloir (ou plusieurs) qui a la même direction que lui. Cet exemple représente un système multi-agents bien qu'il se rapporte aux robots, puisque la considération des robots comme des agents n'est pas nouvelle et plusieurs travaux l'ont prise en compte, voir par exemple : [124], [141].

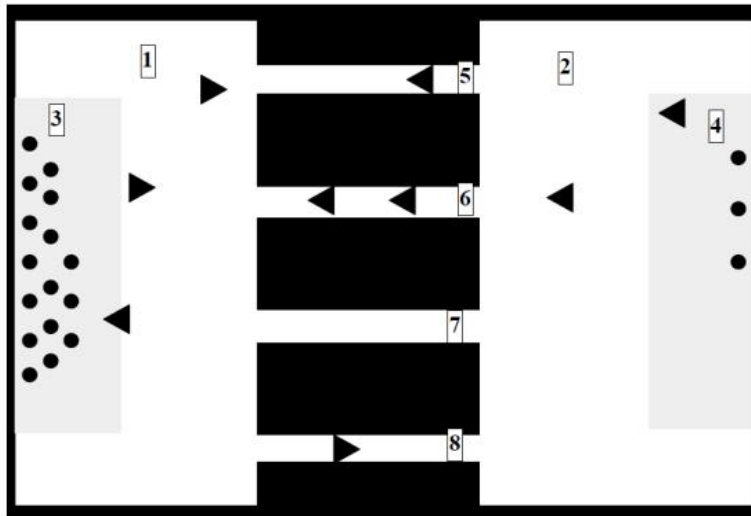


Figure 6.1: Le problème de transport de ressources par des robots

6.2 Modélisation du problème de transport de ressources

Dans Cette section, nous allons voir comment modéliser l'exemple précédent via l'utilisation du paradigme des réseaux de référence. Nous allons voir particulièrement, comment un agent mobile (réseau objet) va-t-il raisonner, agir sur les ressources et se déplacer entre différentes positions (réseau système). Pour cela, nous ferons appel à l'architecture MULAN (Multi-Agent Nets), utilisée pour décrire l'aspect hiérarchique des systèmes multi-agents. Le système global est modelé et implémenté avec Renew.

Comme on a dit dans le chapitre précédent, pour modéliser un système multi-agents, plusieurs vues sont présentées : le système multi-agents dans sa totalité, les plateformes sur lesquelles les agents sont positionnés, l'agent lui-même ou tout simplement son comportement. Commençons par le système dans sa totalité. Il sera modélisé comme un réseau système dont les places schématisent les positions (les plateformes) et les transitions représentent les déplacements possibles d'une position vers une autre comme le montre la figure 6.2.

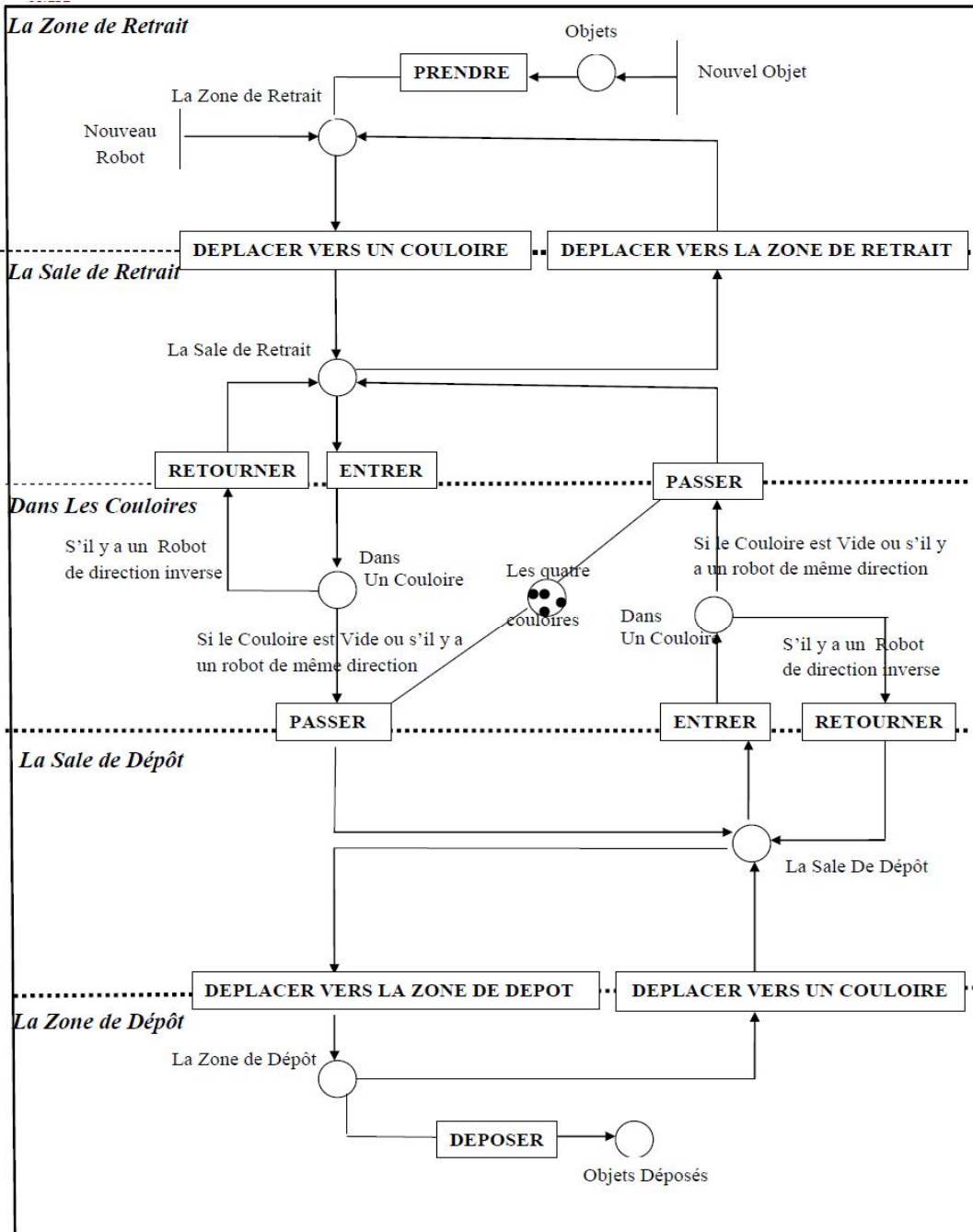


Figure6.2: Le réseau système du problème de transport de ressources

Il contient cinq places : "la zone de retrait", "la salle de retrait", "les couloirs", "la salle de dépôt" et "la zone de dépôt". Les mouvements possibles entre ces places sont : "déplacer", "entrer", "retourner" et "passer". Ces mouvements ne sont pas toujours possibles, par exemple, un agent (robot) ne peut pas se déplacer vers la zone de retrait s'il possède déjà un objet (une ressource). Dans ce travail, le nombre de couloir proposé est de quatre. Pour des raisons de simplicité, les inscriptions et tous les canaux synchrones ont été omis. Les actions externes, non accessibles à l'agent, sont représentées par des traits non orientés. Les étapes de passage d'un agent par un couloir sont les suivantes :

- 1) Choisir un couloir et entrer.
- 2) Déterminer la situation du couloir (déterminer la direction des agents qu'il contient) et décider :
- 3) De retourner (si le couloir contient au minimum un agent de direction inverse) et choisir un autre couloir, ou de passer (si le couloir est vide ou s'il contient au minimum un agent de même direction que lui).
- 4) Sortir du couloir après passage.

Avec cette méthode, le problème de blocage est évité, notant qu'un agent ne peut pas choisir un couloir si :

- Il y a un autre agent dans ce couloir dans une situation de décision (l'étape 1 et 2) quelle que soit sa direction.
- Il y a un autre agent entrain de sortir du couloir quelle que soit sa direction (soit il est sorti après passage, soit il est sorti puisqu' il n'a pas pu passer par ce couloir "le retour de l'étape 3").

En d'autre terme, un agent ne peut "choisir" un couloir et entrer dedans que si ce dernier est vide ou s'il contient des agents entrain de passer par celui-ci. Toutes les informations qui concernent les couloirs sont stockées dans une base de données appelée "la base de données des couloirs" modélisée par un réseau de pétri en utilisant les canaux synchrones.

Si on zoome une place arbitraire de la figure 6.2(excepté: la place "objets", "Objets déposés" et "les quatre couloirs"), sa structure devient visible, voir figure 6.3.

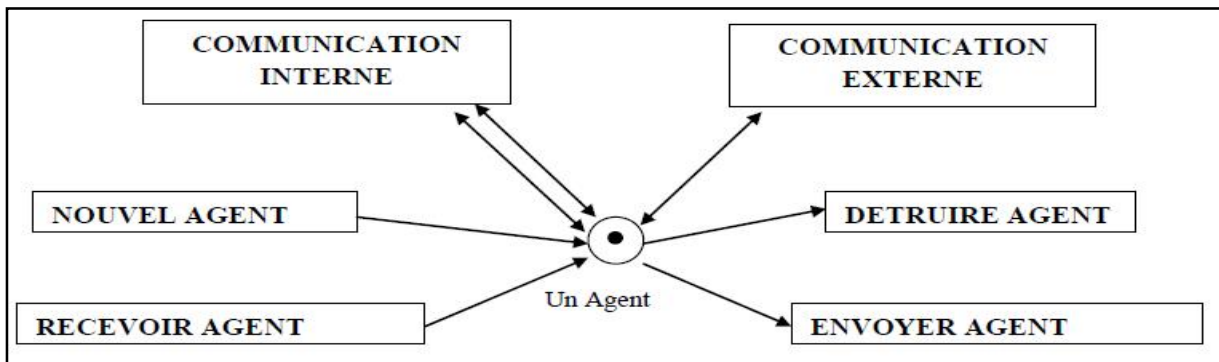


Figure 6.3: La structure d'une plateforme [118]

Sur un tel réseau, la place au centre (agent) contient tous les agents qui se trouvent sur une telle plateforme. Chacune de ces plateformes offre aux agents s'y trouvant un certain nombre de services indiqués sur la figure. Un agent peut-être créé ou détruit. La transition "communication externe" relie deux agents se trouvant sur deux plateformes distinctes et la transition "communication interne" relie deux agents se trouvant sur la même plateforme. La

plate-forme offre également des commodités pour tenir compte de la mobilité : un agent peut quitter la plateforme via la transition “envoyer agent ”ou y accéder via la transition “recevoir agent ” [118].

Le réseau schématisant la structure d’un agent sera visible si l’on zoome la place Agent de la figure 6.3 (voir figure 6.4). Ces derniers sont encapsulés, puisque l’unique manière d’interaction possible est par envoi de message. Ils sont intelligents, puisqu’ils ont accès à la place “implémentation interne” (détaillée plus loin). Le comportement des agents est décrit en termes de protocoles, qui sont une fois de plus des réseaux de Pétri. Ces protocoles sont à la portée de l’agent (sous forme de gabarits ou moules) dans la place “protocoles”. De tels protocoles seront instanciés si un message parvient à l’agent ou s’il décide d’entreprendre une action de manière proactive. Un protocole instancié est appelé "exécution" et est déposé dans la place "exécution". Chaque agent peut contrôler un nombre arbitraire de protocoles, mais ne peut posséder qu’un seul réseau comme celui de la figure 6.4. Tous les messages que l’agent transmet ou reçoit, doivent obligatoirement transiter par ce réseau.

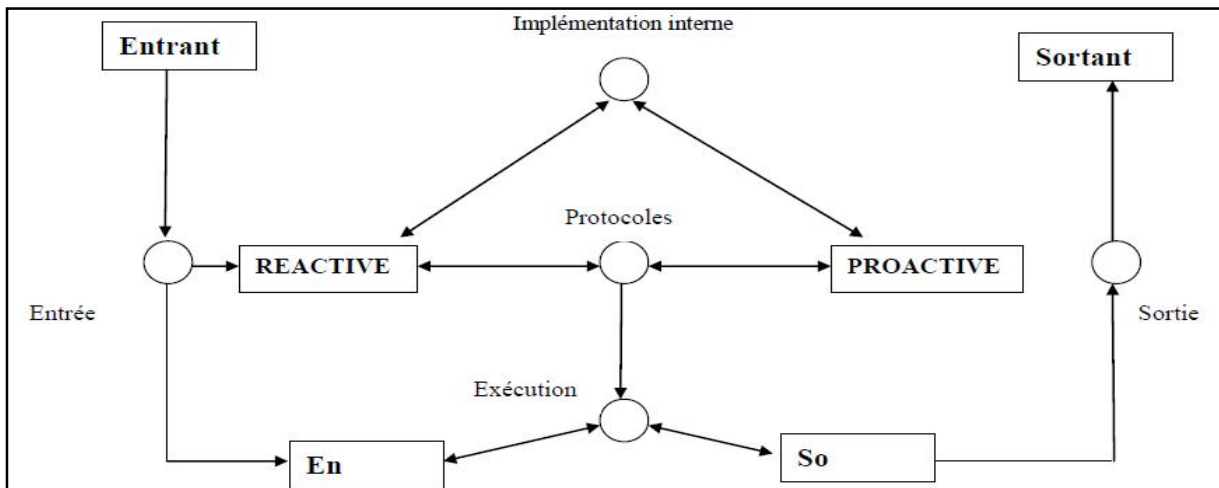


Figure 6.4: La structure d’un agent [adapté de 118]

La principale activité chez un tel agent est la sélection d’un protocole et par conséquent le commencement d’une exécution. Cette sélection peut se faire de façon “proactive” ou “réactive”. Cette distinction correspond à l’accès bilatéral à la place contenant les protocoles. L’unique différence entre ces deux transitions est l’arc reliant la place “entrée” et la transition “réactive”. Ceci signifie qu’une telle transition ne peut être autorisée que par l’arrivée d’un message entrant. En revanche, quelle que soit la situation (sélection réactive ou sélection proactive) le commencement d’une “exécution” est tout le temps influencé par la place “implémentation interne”. Mais dans le cas d’une sélection proactive d’un protocole, la place “implémentation interne” est la seule et unique condition de sélection [118].

La place “implémentation interne” est encore une fois un réseau de Pétri constitué d’une place “base de connaissances” et d’une place “but”, voir figure 6.5. La place “base de connaissances” correspond aux croyances de l’agent, elle décrit ces connaissances sur son environnement ainsi que des informations sur son état(état, position,direction...). La place “but”, comme son nom le dit, correspond au but de l’agent qui est sélectionné par rapport à ces connaissances. La place “base de connaissances” est implémentée par un réseau de Pétri selon le même principe “des réseaux dans les réseaux”.

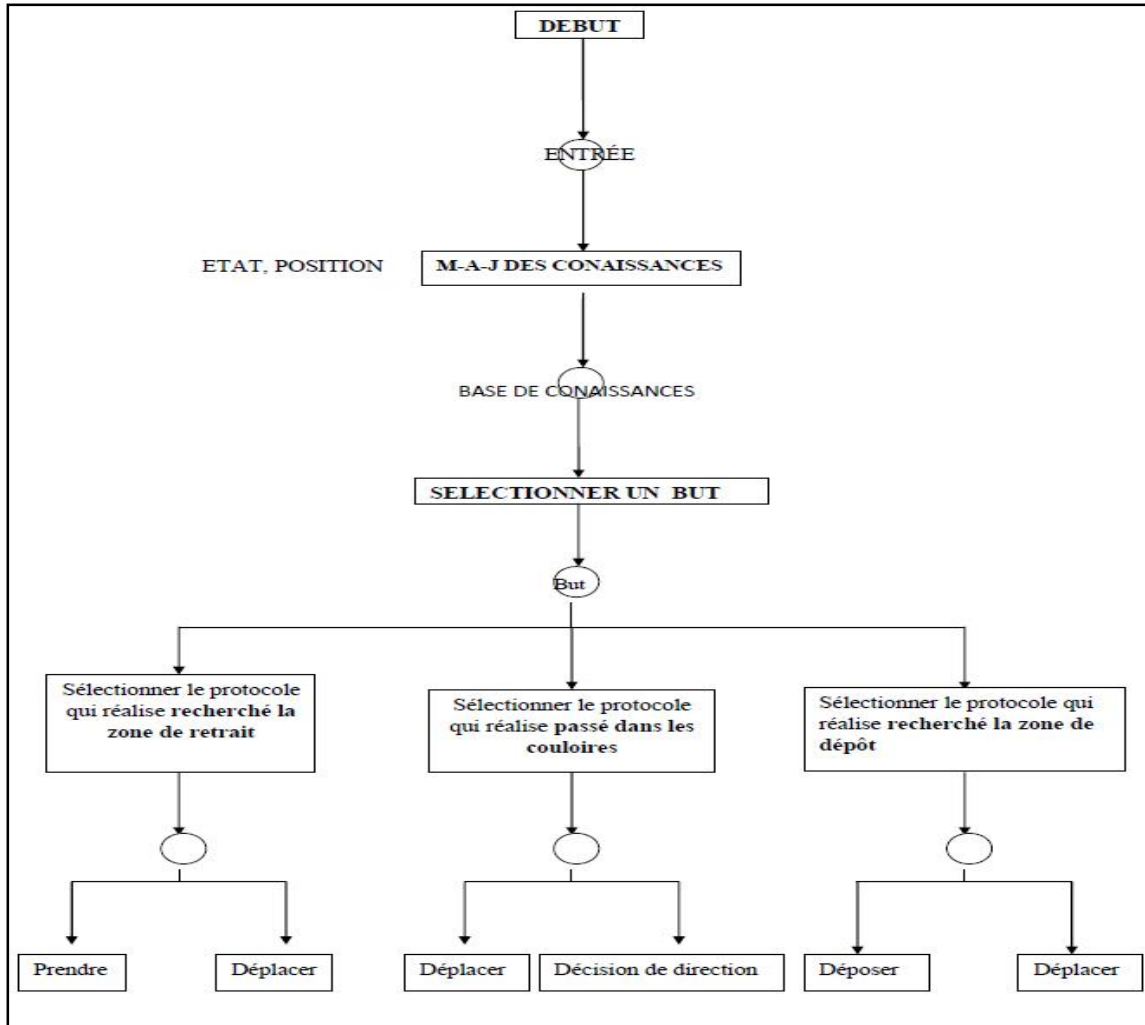


Figure 6.5:La structure de l’implémentation interne [adapté de 118]

Un agent ne peut pas entreprendre une action que s’il est doté de certaines aptitudes et qu’il dispose de certains nombres d’informations. En d’autres termes, avant qu’il décide d’exécuter une action, l’agent doit tenir compte de son état et de celui de son environnement. Par exemple pour qu’un agent décide de déposer un objet, il doit d’abord vérifier qu’il est arrivé à la zone de dépôt. Dans ce modèle, l’agent maintient des informations sur sa position et son état (libre=ne possède pas un objet, ou occupé=possède un objet) (transition “M-A-j

des connaissances”). Guidé par ses informations, l’agent sélectionne un but (transition “Sélectionner un but”).

Le but principal d’un agent est de transporter les ressources de la zone de retrait jusqu’à la zone de dépôt passant par les couloirs. On remarque que l’objectif principal peut être divisé en trois sous-buts: rechercher un objet dans la zone de retrait et s’en saisir, passer par un couloir, rechercher la zone de dépôt et déposer l’objet. C’est-à-dire que les agents vont agir selon un cycle répétitif guidé par ces trois objectifs. Quel que soit le but sélectionné, il est en fait un ensemble d’actions à exécuter. De telles actions correspondent ici à des canaux synchrones qui invoqueront des protocoles comportementaux dont l’exécution accomplira le but désiré (les protocoles comme: "prendre", "déposer" et "décision de direction"). Ce cycle est répété jusqu’à ce que tous les objets aient été transmis à la zone de dépôt.

La figure 6.6 représente les réseaux de protocoles qui montrent le comportement de base d’un agent. Les transitions inscrites par les canaux synchrones “Début” et “Fin” ont la même utilisation pour tous les réseaux de protocoles. Le canal “Début” sert à faire parvenir l’ensemble des paramètres au protocole et quand ce dernier termine sa tâche, la transition inscrite avec le canal “Fin” est autorisée. En l’exécutant, l’agent détruit le protocole ou plus précisément l’instance du protocole.

Lorsqu’un agent arrive à la zone de retrait, il est effectivement libre et l’objet est à la portée de ses mains, alors le protocole “prendre” est sélectionné. En revanche, lorsqu’un agent arrive à la zone de dépôt il est effectivement occupé et il doit placer l’objet dans la zone de dépôt alors le protocole “déposer” est sélectionné. Après l’instanciation du protocole “prendre” (respectivement “déposer”), la transition “saisir” (respectivement “posé”) produit une performative N définissant le nouvel état de l’agent qui sera transmise au protocole implémentation interne à travers le canal “Sortir”. On dit que l’agent est désormais “occupé” (respectivement “libre”) et le protocole est ainsi terminé (en exécutant la transition “Fin”). Dans les deux cas, l’émission de la performative N provoque la reprise du protocole implémentation interne et le canal "Debut" de ce protocole est utilisé pour lui faire passer ce paramètre. Ceci permettra la mise à jour de la base de connaissances, et la sélection d’un nouveau but. Ainsi, l’agent soit, il est arrivé à la zone de retrait et il est capable de sélectionner et d’instancier le protocole “prendre”, soit il est arrivé à la zone de dépôt et il est capable de sélectionner et d’instancier le protocole “déposer”, soit tout simplement il décide de se “déplacer” pour chercher la zone désirée.

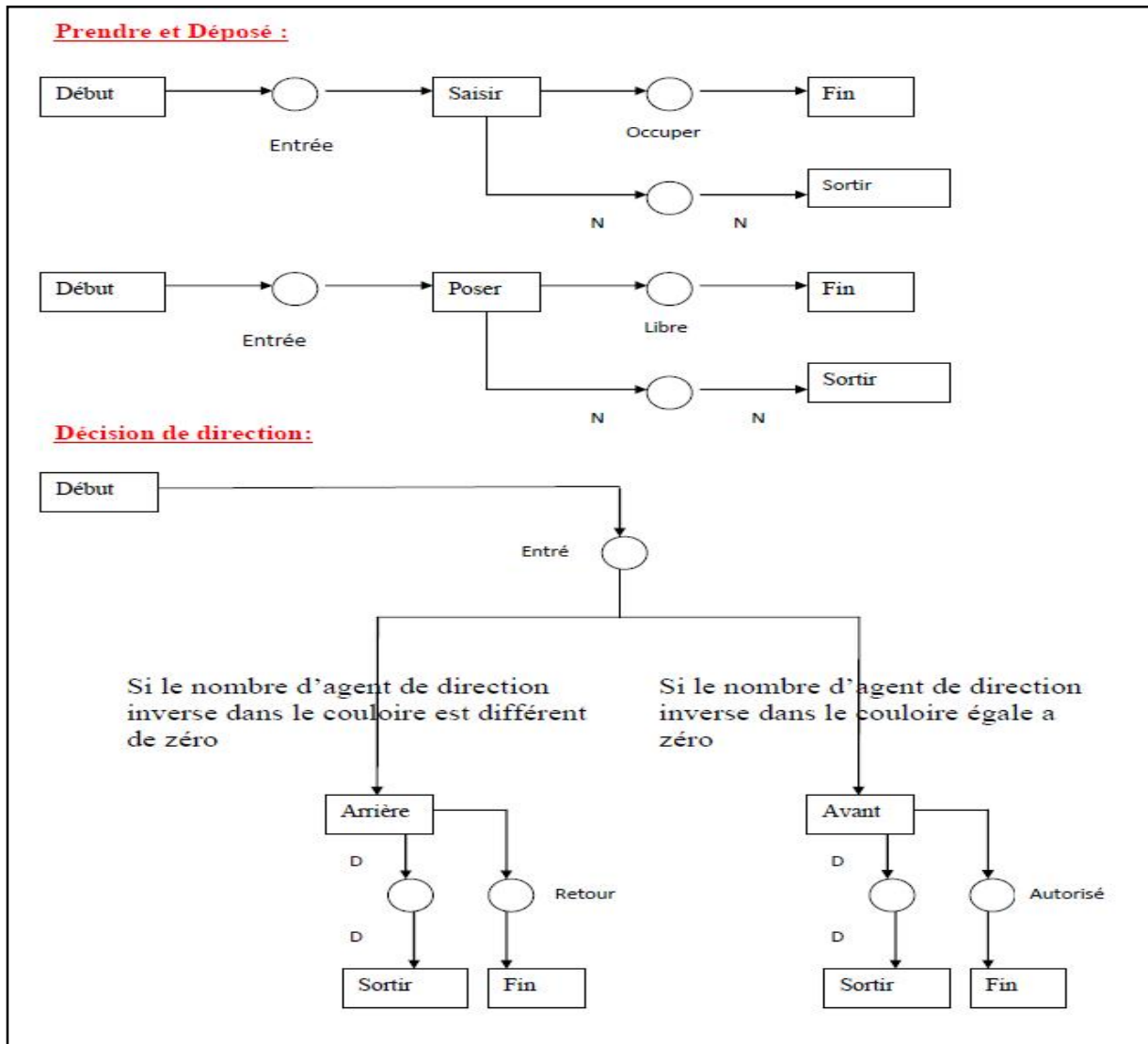


Figure6.6: Les protocoles prendre, déposer et décision de direction

Un agent qui cherche la zone de retrait ou la zone de dépôt doit passer par un couloir. Les couloirs sont très étroits (la largeur du couloir ne permet pas le passage de deux agents en même temps) ce qui oblige l'agent à être très prudent pour qu'il ne tombe pas dans une situation d'interblocage. Pour éviter ce problème, des conditions ont été proposées. Celles-ci sont liées au choix d'un couloir (voir les premiers paragraphes de cette section). Alors, il ne reste à l'agent qu'à décider sa direction : en arrière pour faire le retour ou en avant pour passer. Dans ce cas, le protocole "Décision de direction" est sélectionné et instancié. Consultante "la base de données des couloirs", l'agent peut décider.

S'il y a des agents dans le couloir qui ont une direction inverse que lui, il décide de retourner, sinon il a l'autorisation de passer. Dans les deux cas (la transition "avant" ou "arrière") le performative D est produit pour définir la nouvelle direction qui sera transmise

au protocole "implémentation interne" à travers le canal "Sortir". Le protocole est terminé en exécutant la transition "Fin". L'émission du performative D provoque la reprise du protocole implémentation interne et le canal "Debut" de ce protocole est utilisé pour lui faire passer ce paramètre. Ceci permettra la mise à jour de la base de connaissances. Un agent autorisé à passer, va "déplacer" simplement dans le couloir pour continuer son chemin, sans oublier de mettre à jour la base de données des couloirs.

6.3 Validation du modèle

L'ingénieur de test (chargé de la modélisation) peut, en utilisant Renew, imaginer différents scénarios d'exécution en agissant sur le nombre d'agents ou sur le nombre d'objets. Tout en simulant le comportement du modèle pour chacun des scénarios d'exécution imaginés, l'ingénieur de test pourra, en fonction des résultats obtenus, corriger et/ou raffiner progressivement le modèle abstrait de test.

6.4 Simulation et génération des cas de tests abstraits

Quand une simulation de réseau est commencée, Renew le simulateur crée une instance du réseau qui est simulé. Un réseau qui est dessiné dans l'éditeur est une structure statique (fond blanc sur la figure 6.7). Cependant, une instance du réseau a un marquage qui peut changer avec le temps (fond gris sur la figure 6.7). Toutefois qu'une simulation est commencée, une nouvelle instance est créée. Chaque réseau dans Renew a un nom, ce nom est dérivé à partir du nom de fichier où il se trouve en enlevant le nom du répertoire et le suffixe. Le nom de l'instance de ce réseau est composé du nom de réseau ainsi qu'une numérotation entre crochets, notant que chaque instance d'un réseau a son numéro unique [120].

Le réseau système du problème de transport de ressources et son instance (SMA[0]) sont présentés dans la figure 6.7, notant que dans la figure, il existe trois agents qui doivent transporter trois objets à la zone de dépôt. Deux agents entre eux sont déjà dans la zone de retrait (la transition "Nouveau_Agent" est franchie deux fois), agent[6] et agent[10] sont leur nom d'instance de réseau et leur fenêtre apparaît en bas de la figure.

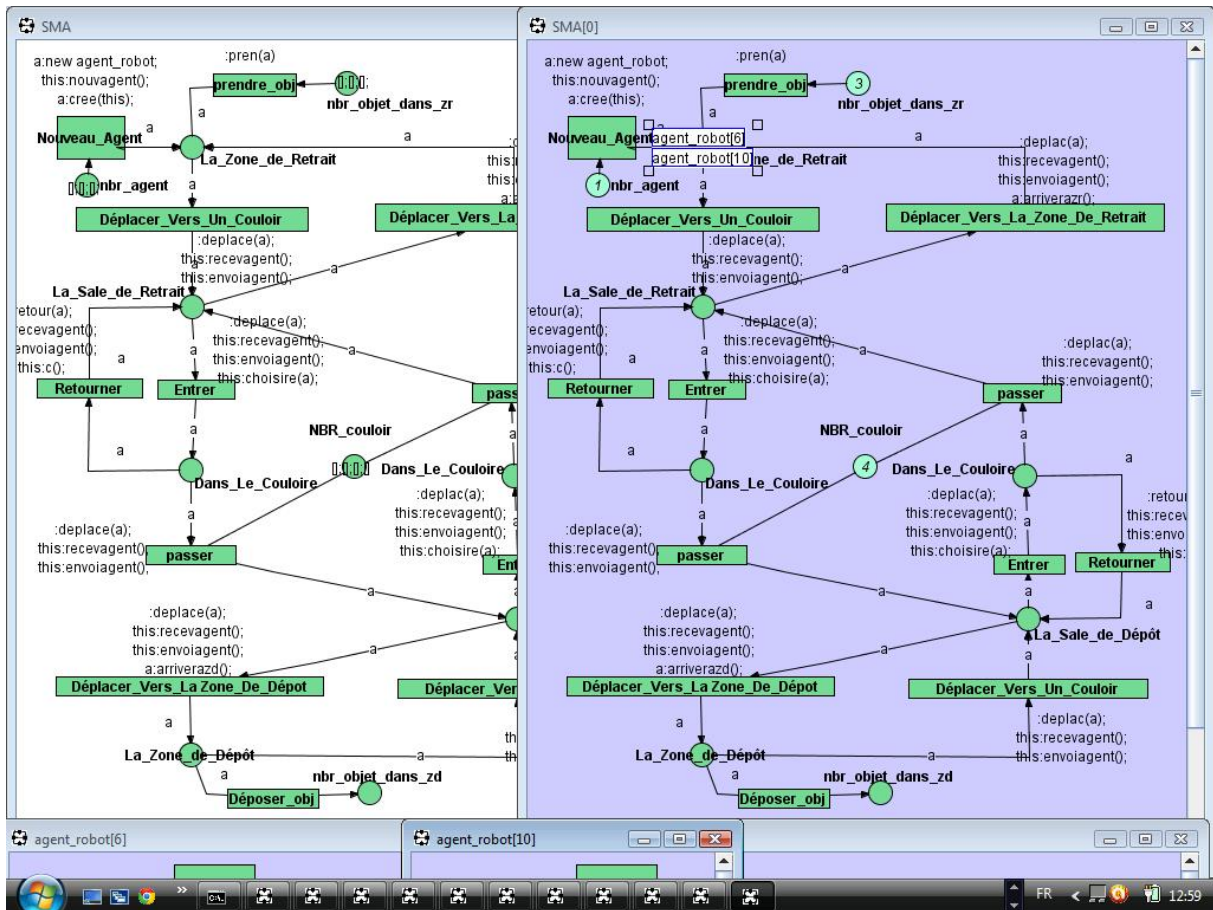


Figure6.7: Une simulation de réseau système du problème de transport de ressources dans Renew

Cette simulation du réseau de transport de ressources peut-être déclenchée si l'on charge tous les réseaux décrits précédemment dans la section 6.2. D'après le chapitre précédent, il est possible de récupérer toute la trace de simulation sous forme d'un fichier texte, c'est le cas pour la figure 6.8 qui représente la trace de simulation de la figure 6.7.

Les phrases après «//» sont justes pour expliquer la signification de chaque ligne. Les premières lignes de cette trace de simulation indiquent que le simulateur démarre par la création de l'instance de réseau système du problème de transport de ressources. Il initialise, ensuite, le nombre de couloirs, le nombre d'agents et le nombre d'objets dans la zone de retrait. Le franchissement de la transition "Nouveau_Agent", provoque l'exécution du constructeur de réseau agent et par conséquent de son instanciation, alors le réseau agent "agent_robot[6]" est créé et prend place dans "La_Zone_de_Retrait". Le franchissement de cette transition cause également l'exécution du constructeur du réseau implémentation interne et son instanciation "implementation_inter[7]". Si cette transition est franchie encore,

le même est fait mais avec une nouvelle instance du réseau agent "agent_robot[10]" qui a une nouvelle instance du réseau implémentationinterne "implementation_inter[11]".

```

simulation fr - Bloc-notes
Fichier Edition Format Affichage ?
(1)New net instance SMA[0] created. //le réseau SMA (réseau de transport) est crée
(1)Initializing [] into SMA[0].NBR_couloir //
(1)Initializing [] into SMA[0].NBR_couloir //initialiser quatre jeton pour quatre couloirs
(1)Initializing [] into SMA[0].NBR_couloir //
(1)Initializing [] into SMA[0].NBR_couloir //
(1)Initializing [] into SMA[0].nbr_objet_dans_zr //initialiser trois jeton pour trois objets (c'est un exemple)
(1)Initializing [] into SMA[0].nbr_objet_dans_zr //il peut être plus
(1)Initializing [] into SMA[0].nbr_objet_dans_zr
(1)Initializing [] into SMA[0].nbr_agent //initialiser trois jeton pour trois agents (c'est un exemple)
(1)Initializing [] into SMA[0].nbr_agent //il peut être plus
(1)Initializing [] into SMA[0].nbr_agent //

(2)----- Synchronously -----
(2)Firing SMA[0].Nouveau_Agent //constructeur d'agent

(2)New net instance agent_robot[6] created. //le réseau agent est crée

(2)Firing agent_robot[6].Nouv_implementation_inter //constructeur de l'implémentation interne

(2)New net instance implementation_inter[7] created. //le réseau implémentation interne est crée

(2)Putting agent_robot[6] into SMA[0].La_Zone_de_Retrair //l'agent est dans la zone de retrait
(3)----- Synchronously -----
(3)Firing SMA[0].Nouveau_Agent //
(3)New net instance agent_robot[10] created. //le même est fait pour crée un autre agent
(3)Firing agent_robot[10].Nouv_implementation_inter //avec une autre implémentation interne
(3)New net instance implementation_inter[11] created. //
(3)Putting agent_robot[10] into SMA[0].La_Zone_de_Retrair //
    
```

Figure6.8:Trace de simulation

6.5 Concrétisation des cas de test abstraits

Dans ce travail une implémentation du problème de transport de ressources est employée. Comme exemple nous avons huit classes simples et une classe d'agent, comme mentionné dans la figure6.9.

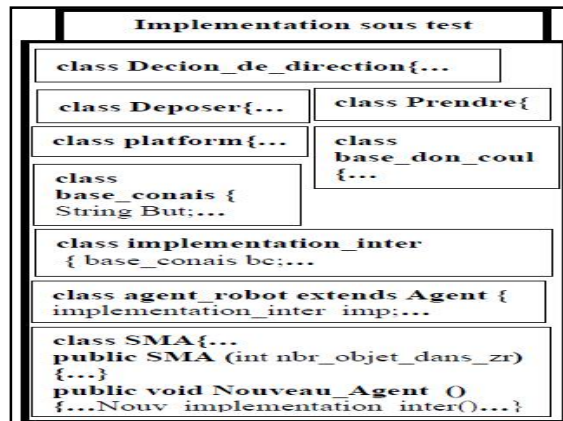


Figure 6.9: Implémentation sous test

6.5.1 Analyse statique

Depuis la trace de simulation de la figure 6.8 et de la figure 6.10 (la trace de simulation de la figure 6.10 est le complément de la figure 6.8), l'agent testeur pour tous les mots clés, dans notre exemple ils sont: NBR_couloire, nbr_objet_dans_zr, nbr_agent, Nouveau_Agent, Nouv_implementation_inter, La_Zone_de_Retrait, But, Selectionner_un_but...

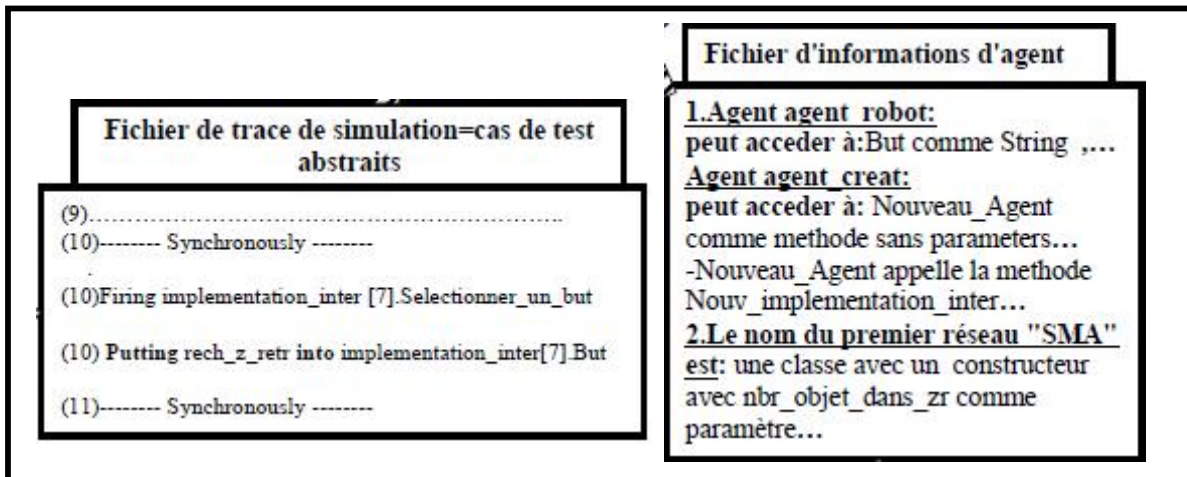
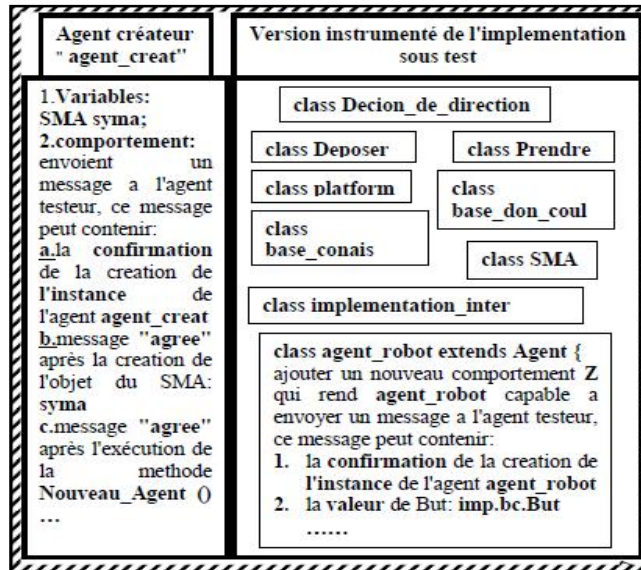


Figure 6.10: Analyse statique

La figure 6.10 représente aussi une partie de fichier d'informations d'agent, obtenue à l'étape de l'analyse statique. Il est clair depuis la figure 6.9 que l'agent "agent_robot" peut accéder au mot clé "But" qui est utilisé dans l'implémentation sous test comme une variable, cependant l'agent créateur ("agent_creat" est son nom dans l'implémentation) peut accéder à la méthode "Nouveau_Agent" puisque "SMA" qui est le nom du premier réseau est implémenter comme classe.

6.5.2 Instrumentation

La figure 6.11 représente une modélisation du code de l'agent créateur ainsi que le code de l'implémentation sous test après instrumentation. Comme on a dit dans le chapitre précédent, l'instrumentation consiste à ajouter un comportement qui permet d'envoyer un message à l'agent testeur, ce message peut contenir : la confirmation de la création de l'instance de l'agent lui-même, ou la valeur d'une variable, ou un message "agree" après l'exécution d'une méthode ou un comportement, ou, dans le cas de l'agent créateur, un message "agree" après la création de l'instance d'agent ou de l'objet de la classe qui a le même nom que le nom du premier réseau.



+Figure 6.11: Instrumentation

6.5.3 La construction des cas de test concrets

La figure 6.12 représente toutes les étapes après l'analyse statique et l'instrumentation. L'agent testeur démarre avec la création de l'instance de l'agent créateur, ensuite, à l'étape prologue, il lui envoie un message. Après réception du message "agree", l'agent testeur sauvegarde l'identificateur de l'agent créateur "ag_creat_ident". Le cas de test abstrait numéro "2" oblige l'agent testeur à créer "Group1", celui-ci contient : implementation_inter[7] (qui est créé par le franchissement de la transition "Nouv_implementation_inter" de agent_robot[6]), agent_robot[6] (qui est créé par le franchissement de la transition "Nouveau_Agent" de SMA[0]), et SMA[0] qui est la racine. En utilisant le fichier d'informations d'agent, l'agent testeur détecte que "Nouveau_Agent" et "Nouv_implementation_inter" sont en relation d'appel, à ce moment là, il décide d'ignorer le "Nouv_implementation_inter" et d'envoyer un message à l'agent créateur (le seul identificateur dans le groupe actuellement) lui demande d'exécuter la méthode "Nouveau_Agent". Le message "agree" vient avec un message "inform" contenant la confirmation de la création de l'instance d'agent "ag_rob1_ident", dans ce cas là, l'agent testeur sauvegarde cet identificateur dans "Group1". Le même est fait pour le cas de test abstrait numéro "3" mais avec un nouveau groupe et un nouveau identificateur d'instance d'agent. Dans le cas de la ligne "Putting" du cas de test abstrait numéro "10", l'agent testeur trouve que "implementation_inter[7]" est membre dans "Group1", ensuite il choisit l'identificateur d'agent "ag_rob1_ident" pour lui envoyer un message et récupérer la valeur de "But". L'agent testeur choisit "ag_rob1_ident" mais pas "ag_creat_ident" parce que

le fichier d'informations d'agent indique que l'agent robot peut accéder à la variable "But".

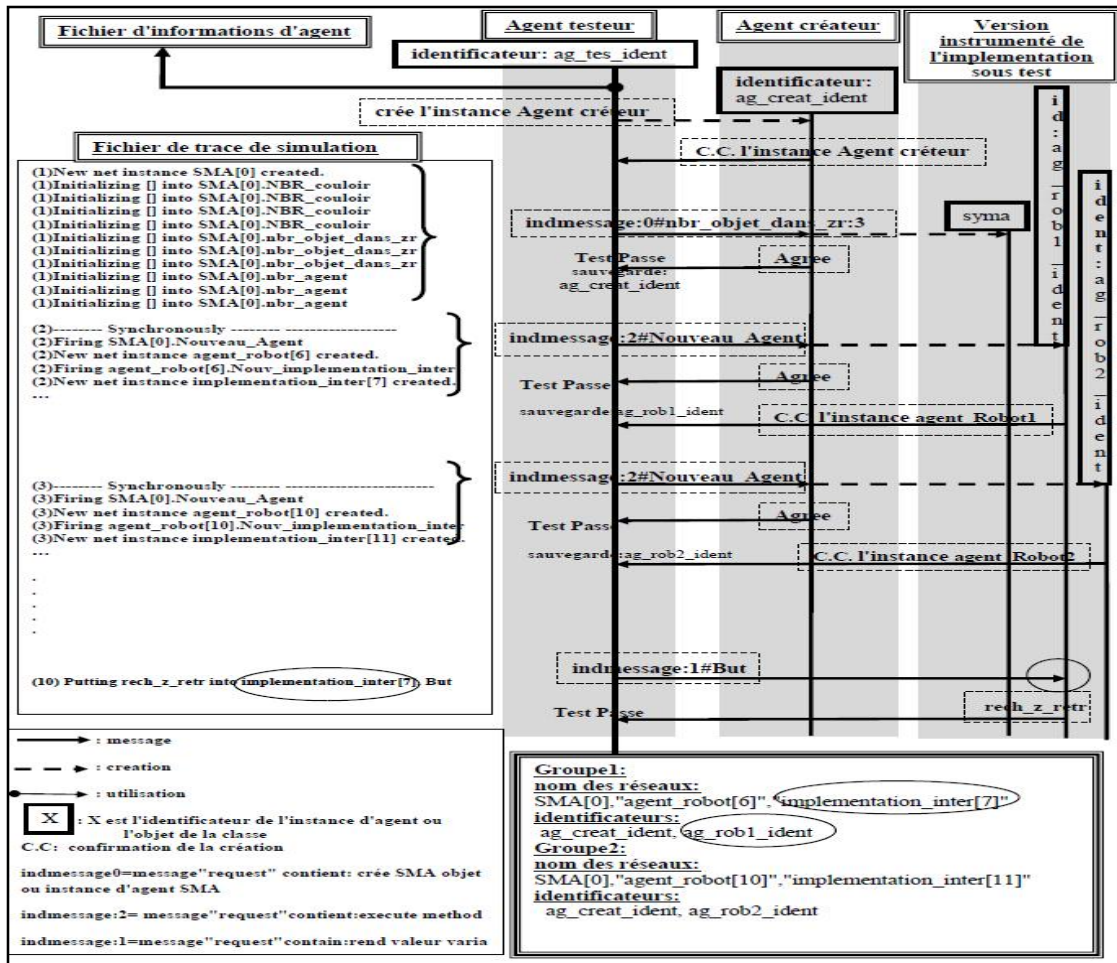


Figure 6.12: La concrétisation des cas de test abstraits

La figure 6.13 rassemble tout le travail de l'agent testeur.

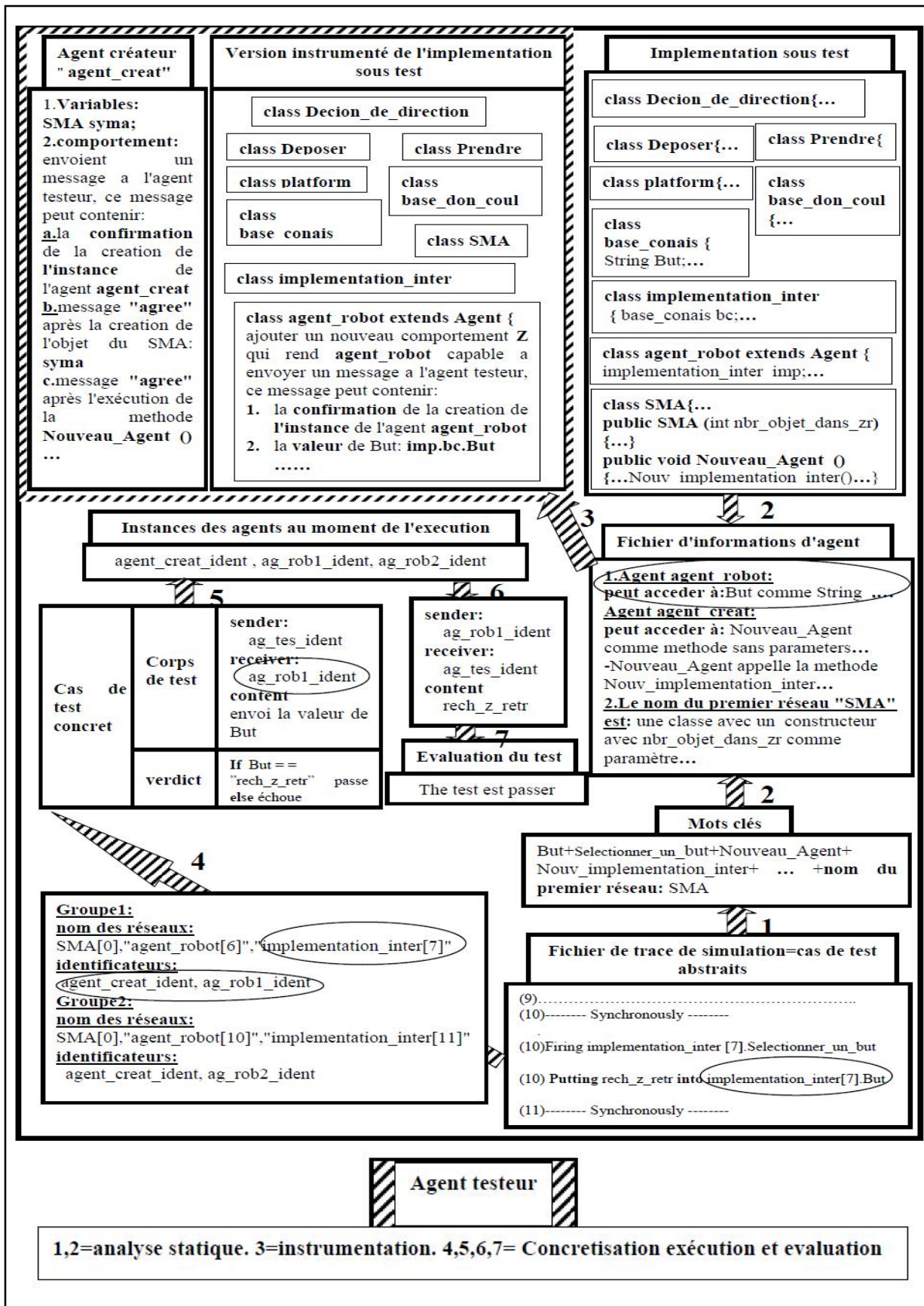


Figure 6.13: Application de l'agent testeur sur le problème de transport

6.6 Présentation de l'application développée

6.6.1 Choix techniques

Le meilleur moyen pour construire un système multi-agents est d'utiliser une plate-forme multi-agents. Une plate-forme multi-agents est un ensemble d'outils nécessaire à la construction et à la mise en service d'agents au sein d'un environnement spécifique. Il existe actuellement plusieurs plates-formes pour le développement des systèmes multi-agents. Notre choix s'est porté sur le langage de programmation Java et la plate-forme de développement des SMA : JADE pour les raisons suivantes :

- Il est simple de créer des agents avec jade.
- Jade gère la communication entre les agents et offre des interfaces de gestion des agents.
- Le mode de communication direct (avec des messages) est supporté par la plate-forme JADE en utilisant le langage FIPA ACL.
- La plate-forme Jade répond à plusieurs fonctionnalités et offre une large gamme de bibliothèques.
- Les agents développés en JADE sont écrits totalement en JAVA qui est un langage facile et basé sur la notion d'objet.
- JAVA a été choisie pour ses riches outils facilitant son utilisation.

La programmation s'est effectuée sous l'environnement de développement Eclipse qui depuis quelques années est arrivé à maturité et propose une richesse et une ergonomie exemplaire.

6.6.2 Interface

Nous avons développé un outil de test des systèmes multi-agents, la figure 6.14 représente la première fenêtre de celui-ci. Elle est partagée en deux fenêtres, la première rassemble les boutons et la deuxième représente les différentes étapes de travail de l'agent testeur.

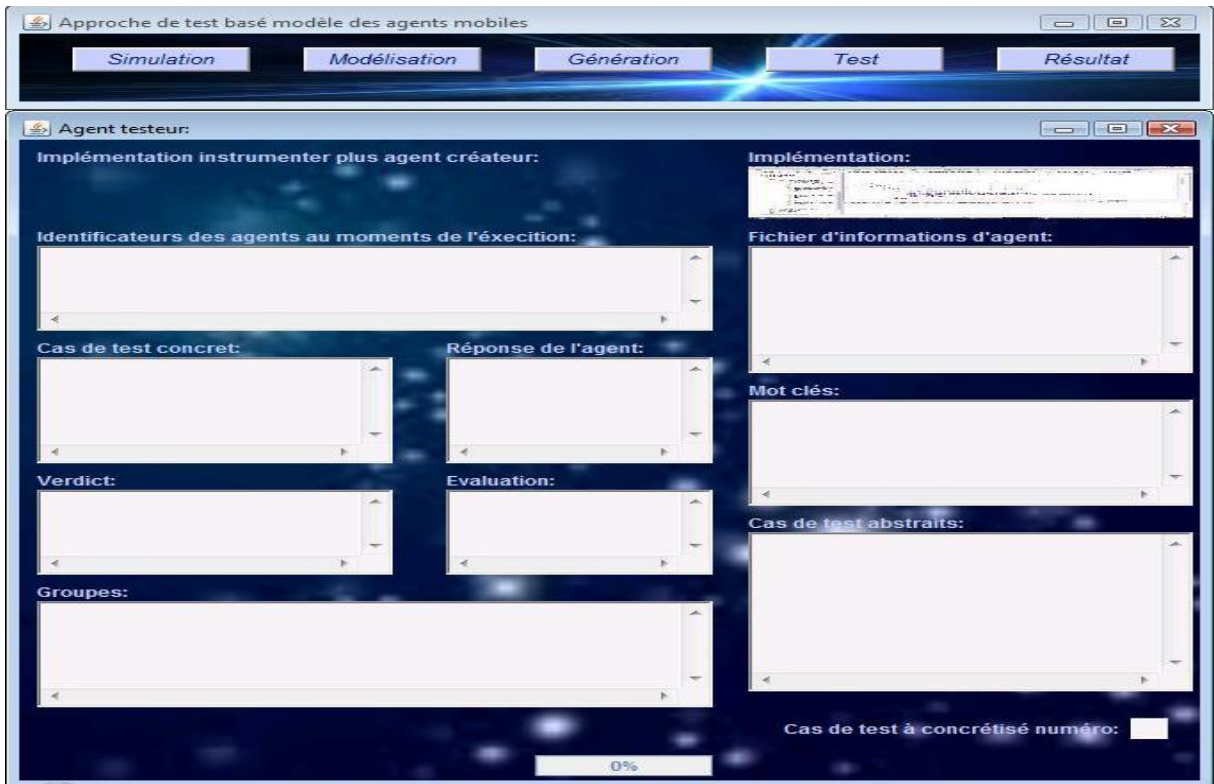


Figure 6.14: Première fenêtre de notre outil de test

Si on click sur le bouton "Simulation" on va voir l'exécution de l'implémentation de notre exemple représenté par la figure 6.15. Dans cette figure on peut voir quatre agents entrain de déplacer huit objets (c'est un exemple).

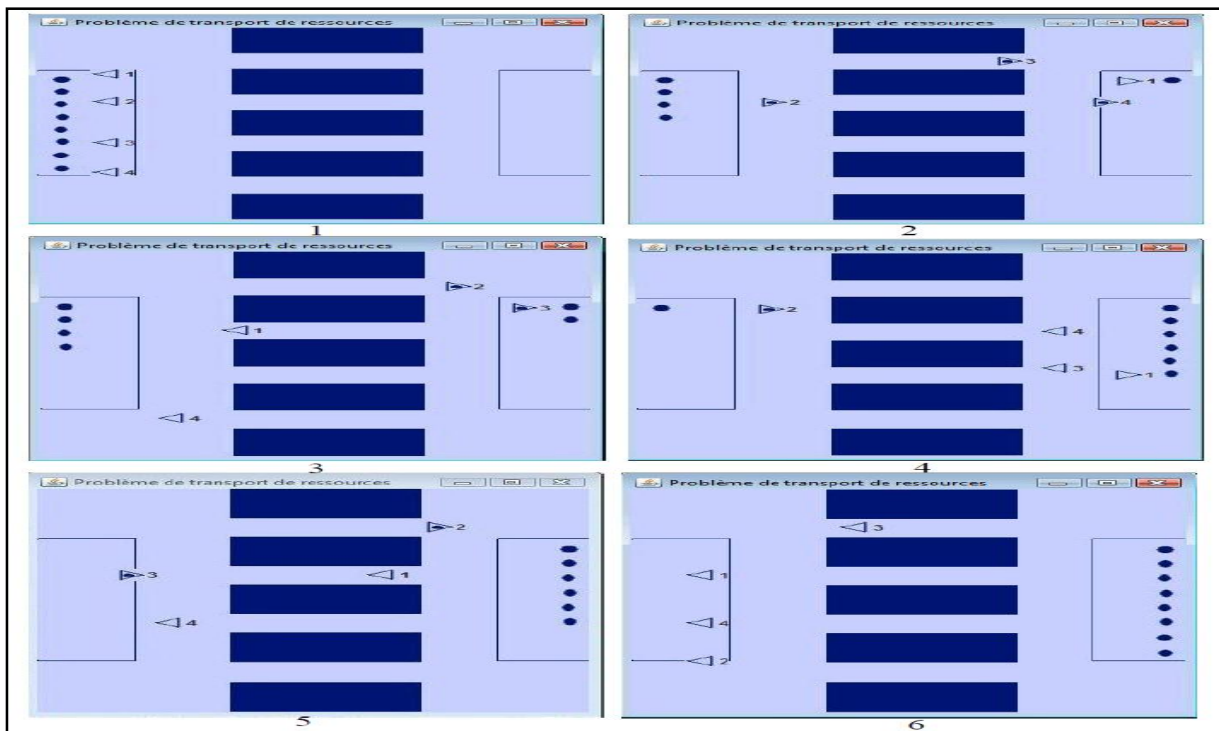


Figure 6.15: Implémentation du problème de transport de ressources

Si on click sur le bouton "Modélisation", la figure 6.7 va apparaître, elle représente nos modèles deRenewet alors on peut faire une simulation qui nous permet à générer les cas de test abstrait, ces derniers peuvent être affichés sur notre fenêtre si on click sur le bouton "Génération" comme le montre la figure 6.16.

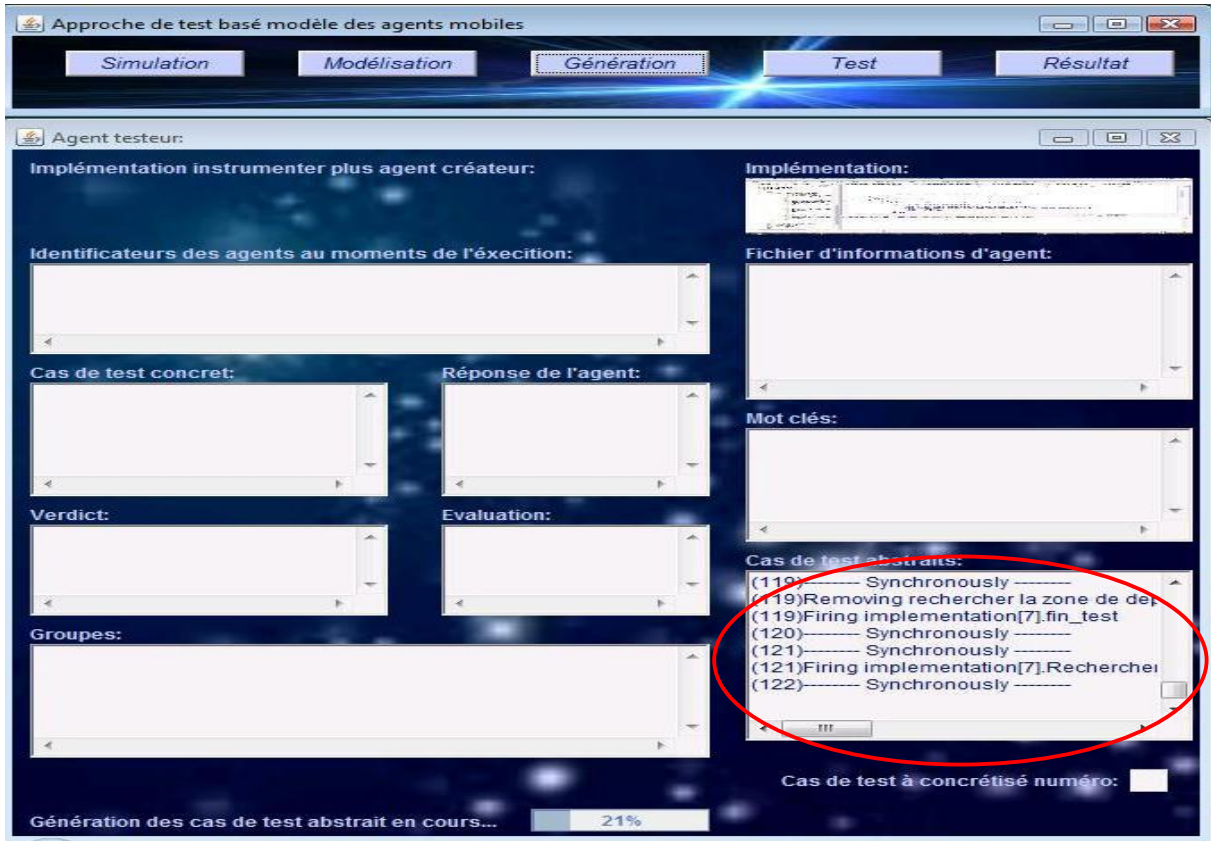


Figure 6.16: Génération des cas de test abstraits

Maintenant on peut lancer l'agent testeur avec le bouton "Test" et alors on peut obtenir la figure 6.17 et la figure 6.18. Ces figures représentent l'étape d'analyse statique (obtenir les mots-clé "figure6.17" et ensuite remplir l'agent d'informations d'agent "figure6.18"). La petite fenêtre qui apparaît dans les deux figures à droite en bas, représente l'agent "Sniffer" qui est un outil de JADE qui permet d'intercepter les messages ACL pendant leur changement et les montrer graphiquement utilisant une notation semblable à UML.

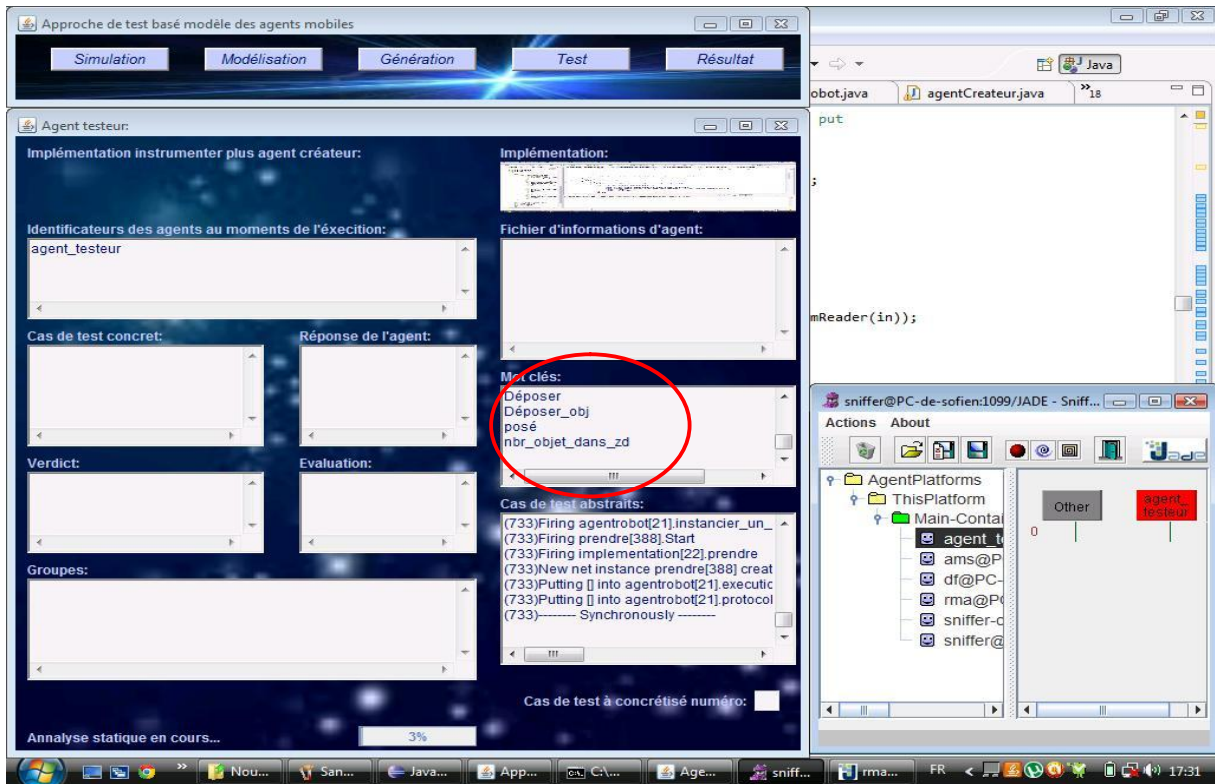


Figure 6.17: Analyse statique (mots-clé)

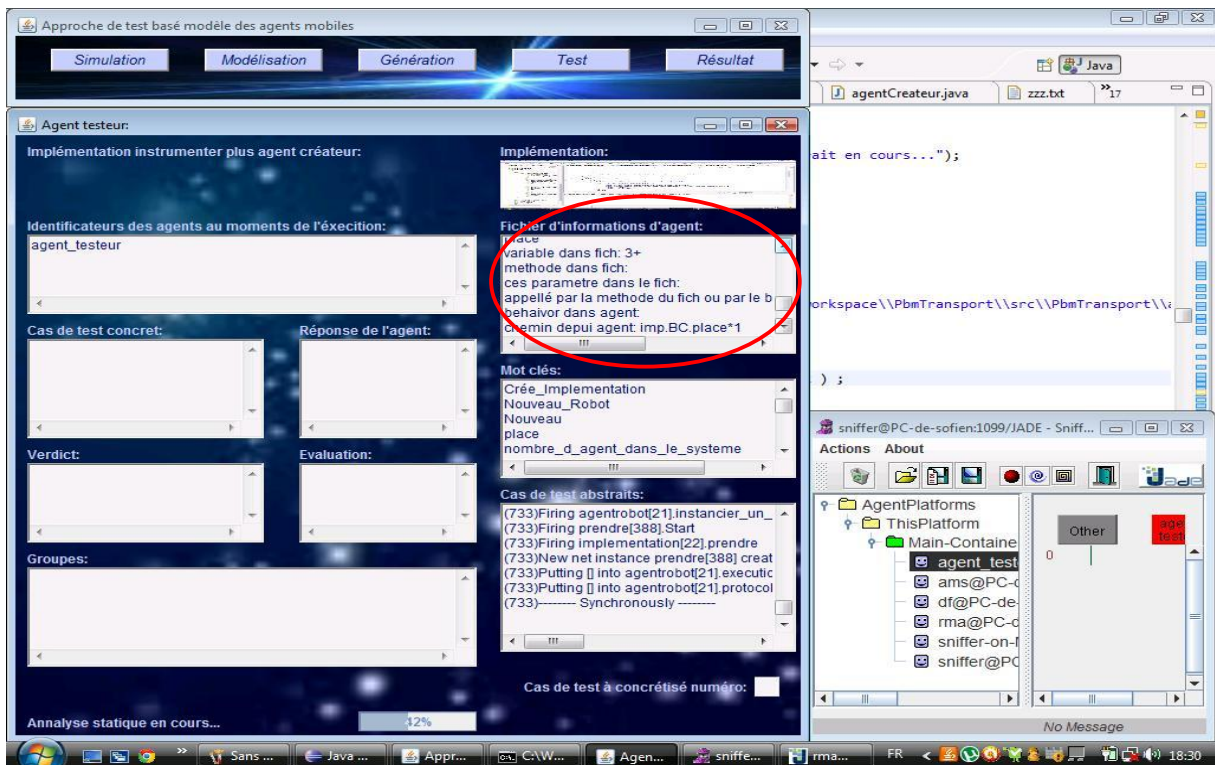


Figure 6.18: Analyse statique (remplir le fichier d'informations d'agents)

La figure 6.19 représente l'étape de l'instrumentation, notant que les codes sources des agents vont apparaître à la fin de cette étape.

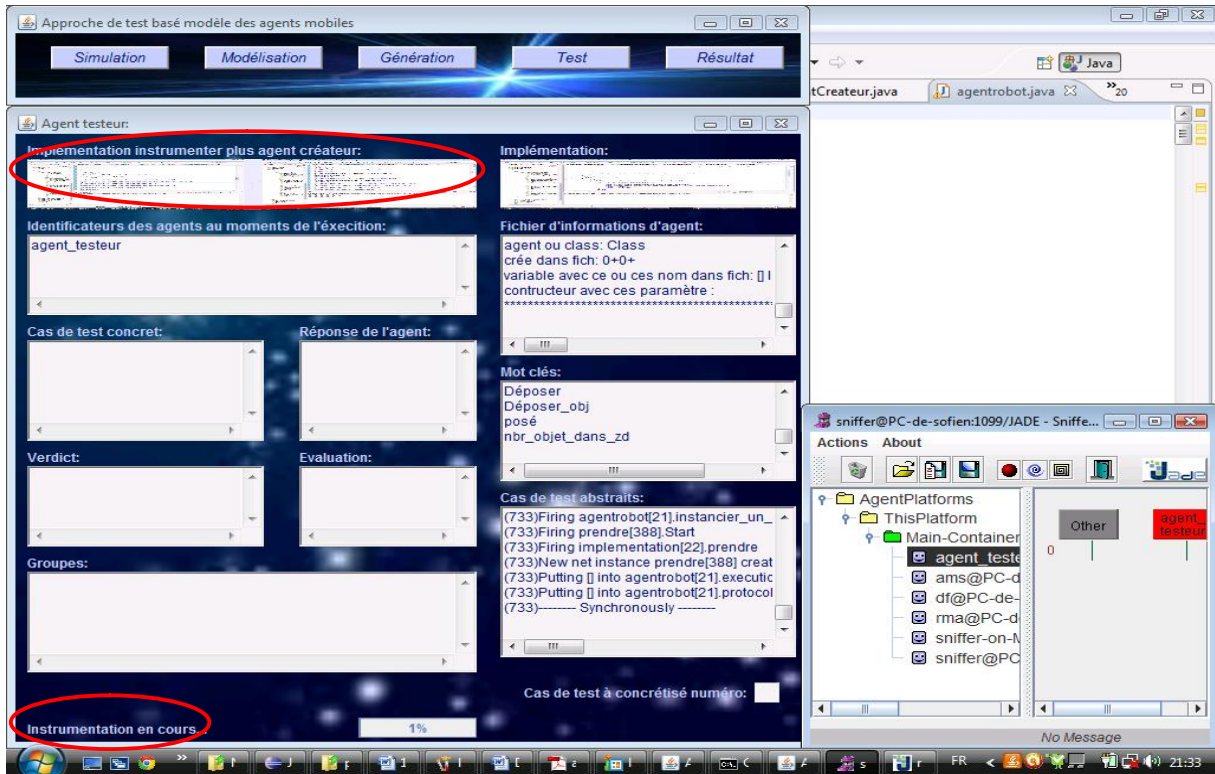


Figure 6.19: Instrumentation

Le début de la concrétisation est représenté dans la figure 6.20. Cette figure représente le changement de message entre l'agent testeur et l'agent créateur qui ramène au lancement de l'exécution de l'implémentation sous test (prologue).

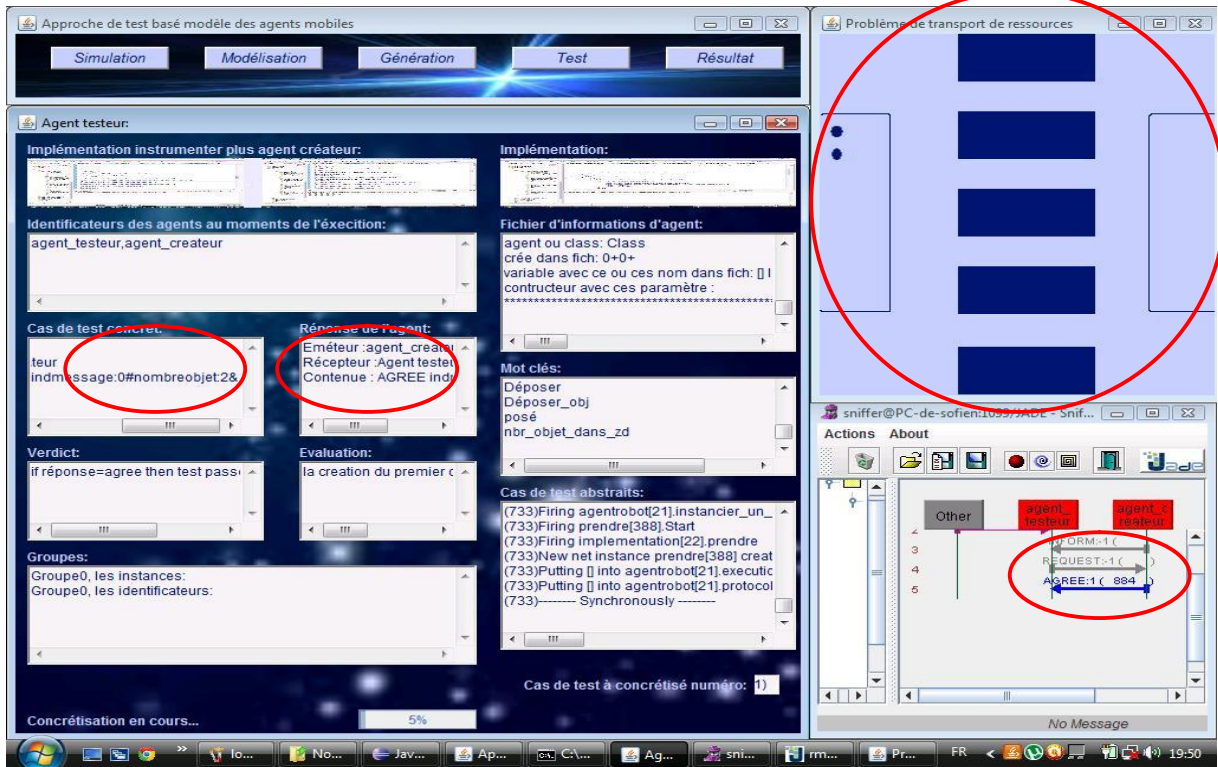


Figure 6.20: Début de la concrétisation (prologue)

Lafigure 6.21 représente l'exécution du message ACL envoyé par l'agent testeur qui demande d'exécuter le comportement ou la méthode qui a le même nom que "Nouveau_Robot".

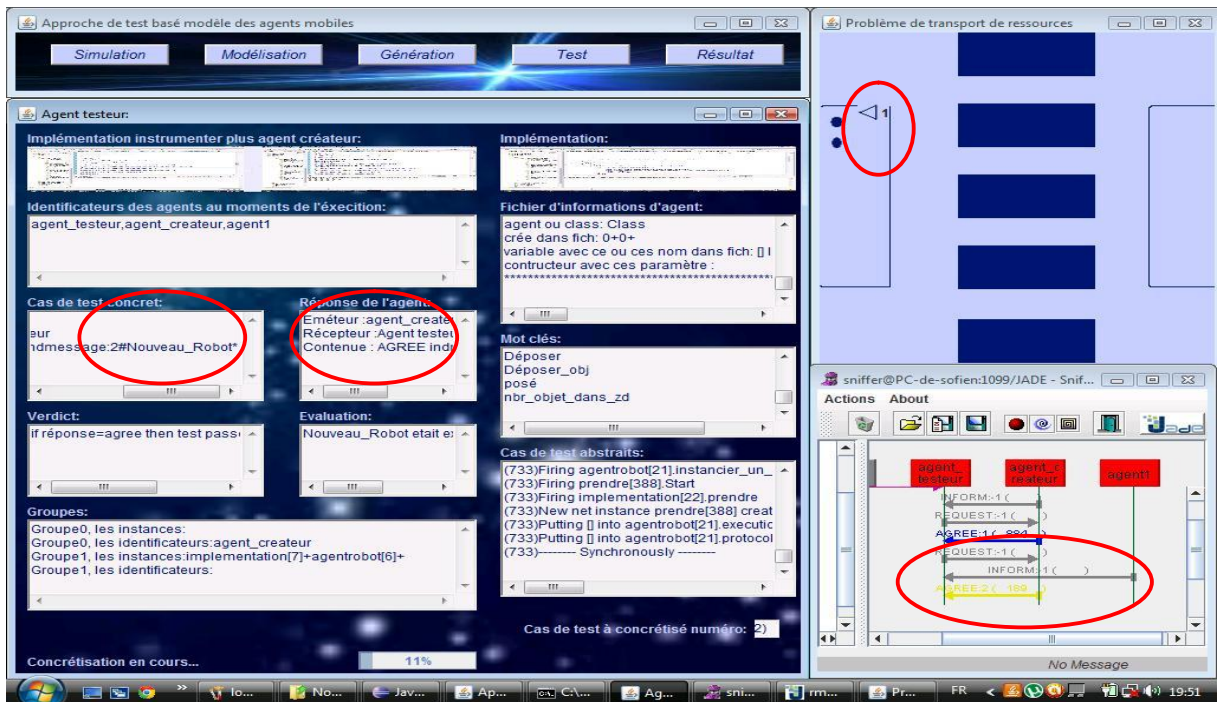


Figure 6.21: Nouveau robot

La figure 6.22 représente la demande de la valeur du variable "but" dans l'implémentation sous test, cette recherche finie par un passage de test (les valeurs sont égales).

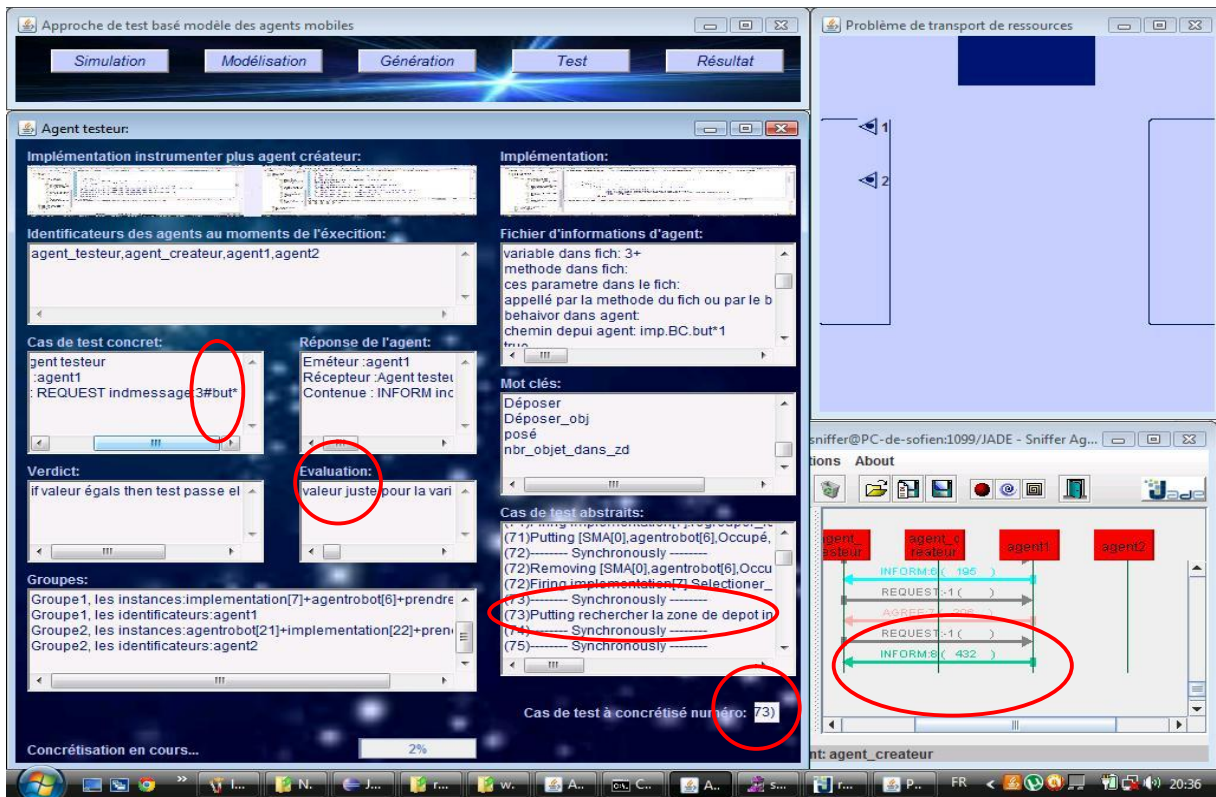


Figure 6.22: Valeur juste pour la variable "but"

Maintenant, on va "tester" l'efficacité de notre agent testeur par l'insertion d'une erreur dans l'implémentation sous test, la validation de l'agent testeur peut-être considéré s'il détecte cette erreur. On va changer la partie du code de l'implémentation sous test qui concerne la sélection de but, comme le montre la figure 6.23.

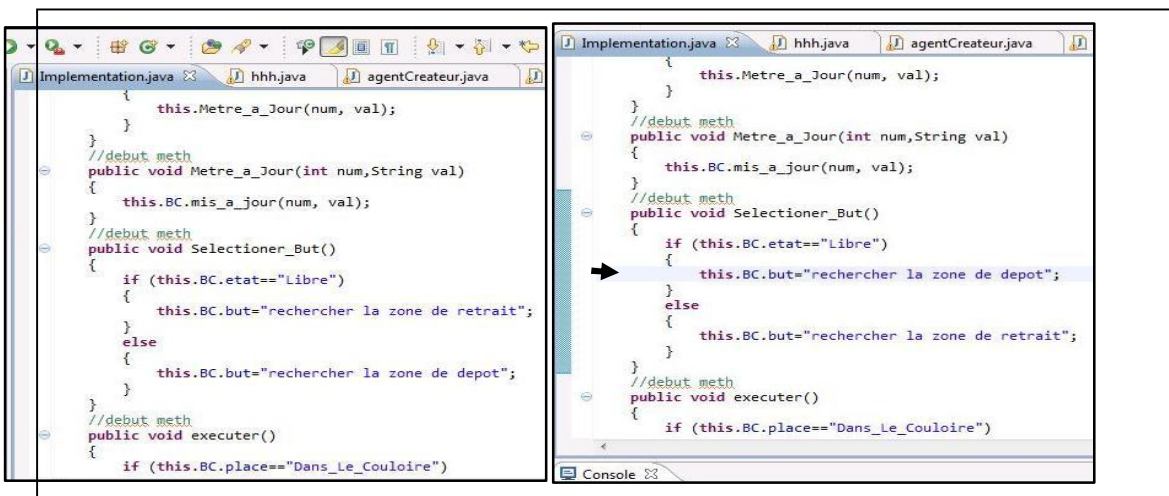


Figure 6.23: Changement du code (L'insertion d'une erreur)

La figure 6.24 représente la validation de l'agent testeur, puisque après le changement du code et le lancement de l'application, l'erreur est détectée.

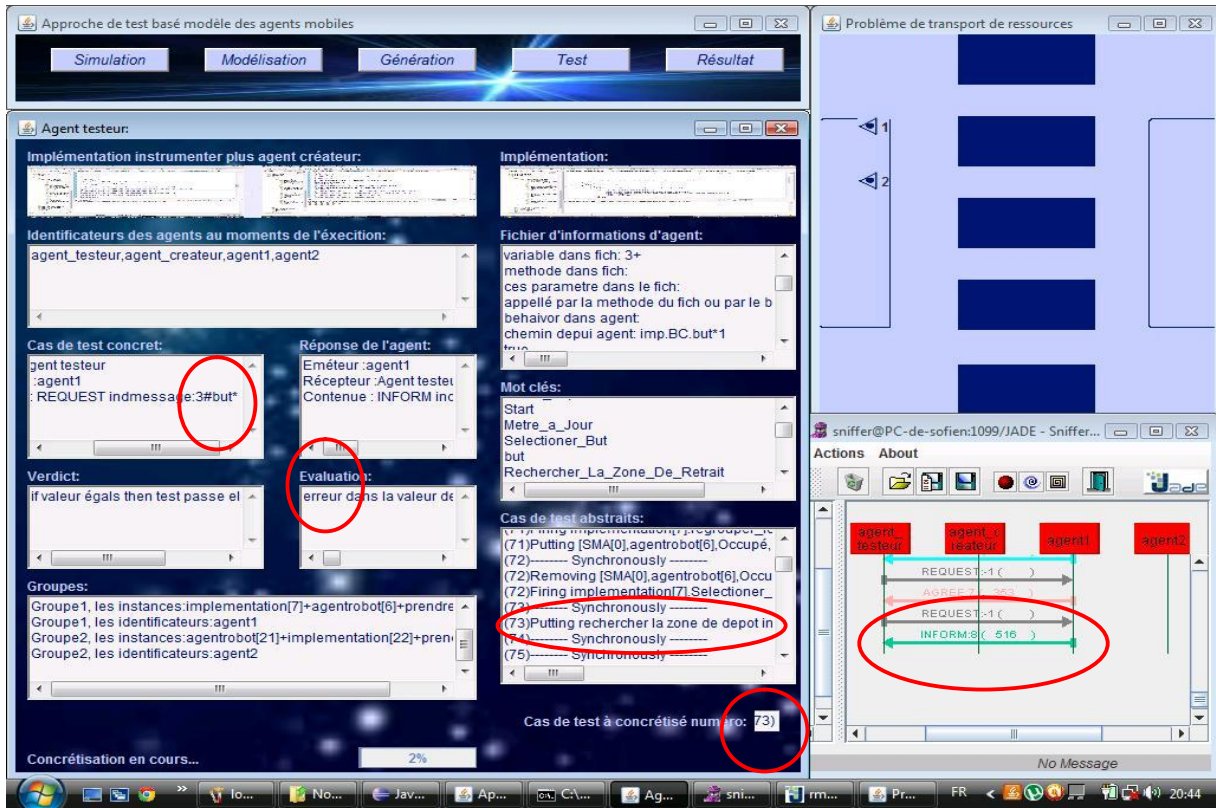


Figure 6.24: Erreur détectée par l'agent testeur

A la fin de la concrétisation on peut tout le résultat obtenu par l'agent testeur (prologue, cas de test concrets, épilogue)et même les cas de test abstraits (Figure 2.25), en cliquant sur le bouton "Résultats".

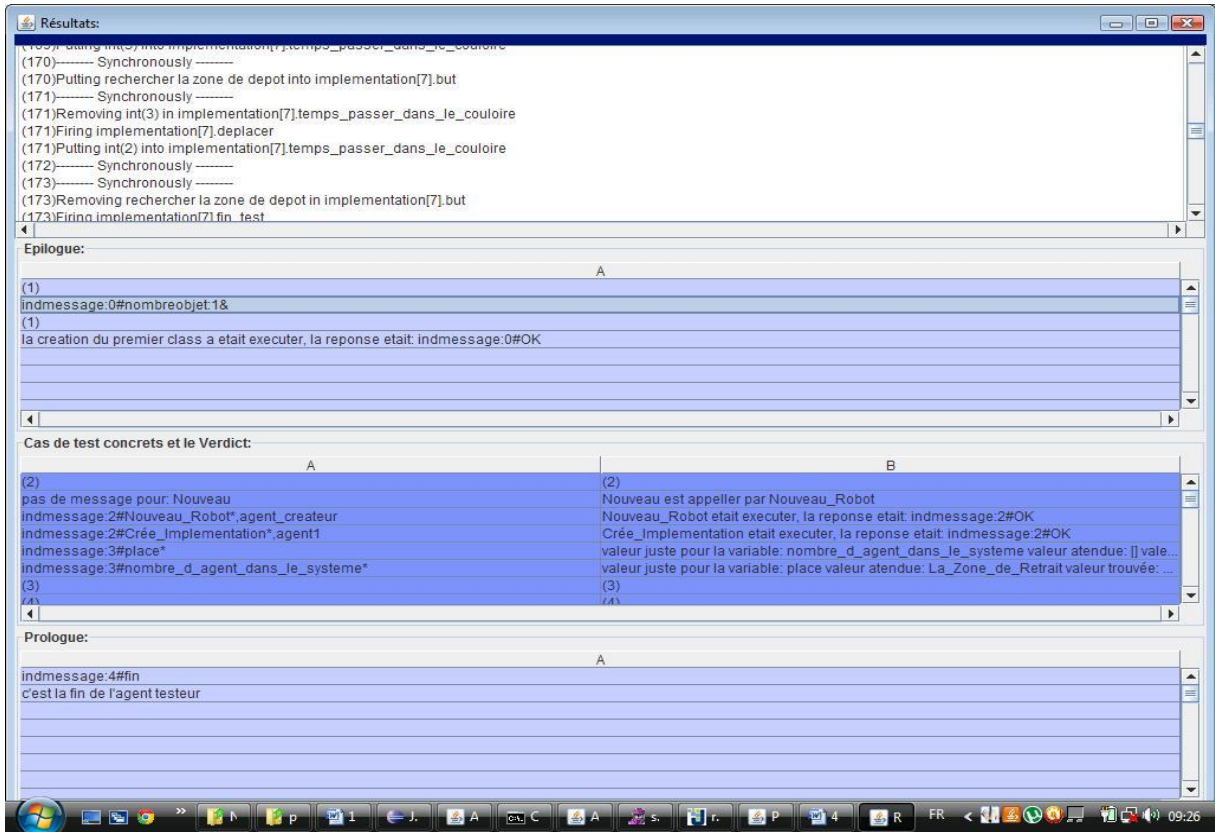


Figure 6.25: Résultats de la concrétisation

6.7 Conclusion

Ce chapitre présente une validation de l'étape de concrétisation des cas de test abstraits proposée dans le chapitre précédent qui est une étape de l'approche de test basé modèle des agents mobiles. Cette étape est la plus difficile dans cette approche. Tout le travail est conduit par un agent testeur et expliqué, en utilisant le problème de transport de ressource par des robots comme cas d'étude.

Chapitre 7: Conclusion et perspectives

Le test apparaît aujourd'hui comme le moyen principal pour la validation du fonctionnement d'un logiciel. Il a pour objectif d'examiner ou d'exécuter un programme dans l'intention d'y détecter un maximum de défauts et de mettre en évidence des points éventuels ou le comportement n'est pas celui attendu. Le test demeure un moyen efficace pour estimer la confiance vis-à-vis d'un logiciel ou d'un composant logiciel.

L'activité du test, omniprésente tout au long du cycle de vie du logiciel, est mise en œuvre par différentes techniques permettant la validation des différentes étapes du développement. L'approche étudiée dans ce mémoire est basée sur la technique de test basé sur les modèles. Cette technique a attiré l'attention à cause de: la facilité de la compréhension et de la maintenance des modèles, la population des modèles au niveau de la conception et de développement des logiciels et dernièrement les avantages qu'elle fournit en termes de génération des cas de test automatique.

Dans notre approche, le modèle utilisé est les réseaux de référence. Un argument important pour l'utilisation du paradigme des réseaux de référence est son élégance et sa capacité d'exprimer la mobilité tout en se basant sur une sémantique formelle. De plus, le processus de modélisation, avec un tel paradigme, conclut non seulement avec un modèle fonctionnel du système mais également avec un ensemble de cas de tests abstraits. Un autre avantage de ce modèle est son adhésion parfaite au principe de la composition des systèmes multi-agents: les protocoles sont composés en comportement, le comportement de l'agent est composé en agents, les agents sont composés en groupes, les groupes se constituent au sein de plates-formes d'agent et les plates-formes sont composées en systèmes multi-agents.

L'outil Renew robuste et facile à manipuler, permet de réduire considérablement l'important effort initial en termes d'heures-homme nécessaires à la construction et la validation du modèle de test. De plus, la structuration en quatre couches (multi-agents, plateforme, agent et protocoles comportementaux) du système selon l'architecture Mulan, rend aisé le processus de modélisation, de validation du modèle et de génération des cas de tests en raison de la taille réduite des parties considérées. L'approche proposée s'appuie fortement sur les résultats de la simulation produits par cet outil. En fait, les étapes de la simulation écrite dans le fichier de trace de simulation sont considérées comme étant les cas de tests abstraits.

Le travail de l'agent testeur était le but de ce travail, celui-ci reçoit le fichier de simulation et le système à tester en entrée pour réaliser le processus de test dans sa totalité. En particulier on parle de l'étape de concrétisation. Plus précisément, une analyse statique des structures internes des agents sous test offre à l'agent testeur une vue globale sur la manière selon laquelle les agents sont structurés, ensuite, l'instrumentation va aider l'agent testeur à contrôler le déroulement de l'exécution des agents en assurant la communication entre les agents sous test et l'agent testeur, ce dernier peut maintenant transformer les cas de tests abstraits en des cas de tests concrets qui sont réellement des messages FIPA ACL, ces derniers vont être envoyés aux agents sous test qui ont été instrumentés. Les réponses des agents seront comparées à la partie verdict des cas de test concrets. L'approche tout entière et le travail de l'agent testeur en particulier étaient appliqués dans ce mémoire sur le problème de transport de ressources par des robots. L'implémentation était réalisée en utilisant la plateforme JADE sous éclipse.

On avait vu dans le chapitre du test des SMA qu'il y a des travaux de test qui se concentrent principalement sur un niveau particulier de test, et des travaux qui concernent plus d'un niveau. Notre approche concerne les quatre premiers niveaux: unitaire, agent, intégration et système et ceci grâce à: l'utilisation des réseaux de référence et ainsi l'architecture Mulan qui décompose le système multi-agents en des niveaux qui conviennent avec ceux des tests des SMA, et l'utilisation de la technique de test basé sur les modèles qui peut-être appliquée à tous les niveaux de test à partir du test unitaire jusqu'au test de système. Ces deux points donnent à notre approche la possibilité de choisir un scénario qui convient avec le niveau de test désirer.

L'exécution prouve la réussite de l'approche et particulièrement si les programmeurs emploient les mots clés (nom des places et des transitions) au maximum dans leur programme, mais ceci ne force pas l'utilisation de tous les mots clés. Pour cette raison, si l'agent testeur détectent l'absence d'un mot clé, cela ne signifie pas qu'il y a une grande erreur dans le programme. La vraie limite pour cette approche est l'appel des méthodes dans l'implémentation sous test (qui n'est pas pris en compte dans le modèle) qui peut changer la valeur d'une variable dans l'implémentation sous test avant qu'elle change dans le fichier de trace de simulation. Une autre limite est qu'on ne peut pas forcer les programmeurs à utiliser les mots clés, ils peuvent utiliser d'autres mots clé ou ils peuvent les utiliser mais dans des endroits difficiles à accéder, par exemple une variable locale dans un comportement

"behaviour". Le problème de l'explosion combinatoire du nombre de cas de tests est un problème qui fait face à toutes les approches de test (le nombre d'étape dans le fichier de trace de simulation avec deux objets et deux agents est 733) et c'est le cas pour notre approche.

Comme perspective et pour améliorer cette approche nous proposons de rechercher des solutions aux problèmes mentionnés dans le paragraphe précédent. On propose par exemple de donner la possibilité à l'agent testeur de détecter si une méthode (nom de transition) est déjà exécuter indirectement (sans l'intervention de l'agent testeur) par une autre méthode. Pour le deuxième problème on peut faire une remarque au début du test pour avertir les programmeurs de ne pas utiliser ce genre de programmation. Le problème de l'explosion combinatoire peut être résolu soit par l'utilisation des spécifications fonctionnelles de test, des spécifications structurelles et/ou stochastiques; soit en faisant appel à la capacité du testeur qui, à partir de ses expériences, ses connaissances et son intuition, peut prédire où les erreurs peuvent se manifester dans un système sous test.

Bibliographie

- [1] Ferber J., "Les systèmes multi-agents : vers une intelligence collective", IIA Informatique Intelligence Artificielle, Edition Inter Editions, 1995.
- [2] J. R. Searle, «Intentions in Communication, chapter 19: Collective Intentions and Actions», MIT PRESS, London, pages 401-415, 1990.
- [3] A. Newell, «The Knowledge Level. Artificial Intelligence» ,18(1), pages 87-127, 1982.
- [4] Nadine RICHARD, « Langage de communication agent basé sur les engagements par l'entremise des jeux de dialogue », École Nationale Supérieure des Télécommunications, octobre 2001
- [5] A. H. Bond, «Commitment: A Computational Model for Organizations of Cooperating Intelligent Agents», In Proceedings of the 1990 Conference on Office Information Systems, Cambridge, pages 21-30, 1990.
- [6] N. R. Jennings and M. Wooldridge, "Applications of Intelligent Agents", Queen Mary & Westfield College, University of London, 1998.
- [7] MBALA Hikolo Aloys. Thèse doctorat : "Analyse, conception, spécification et développement d'un système multi-agents pour le soutien des activités en formation à distance", Université de Franche-Comté, France. 2003.
- [8] Foisel R. "Modèle de réorganisation des systèmes multi-agents : Une approche descriptive et opérationnelle". Thèse de doctorat en informatique, Université de Nancy1, 1998.
- [9] http://www.kalanko.org/affichage_chapitre.php?id_chapitre=3
- [10] Erceau J., Ferber J., " L'intelligence artificielle distribuée". La recherche, 22(233). pp. 750-758, Juin 1991.
- [11] Labidi S., Lejouad W., "De l'Intelligence Artificielle Distribuée aux Systèmes Multi-Agents". Rapport de recherche INRIA N°2004, Août 1993.
- [12] Chaib-draa B., "Interaction Between Agents in Routine, Familiar and Unfamiliar Situations". International Journal of Intelligent & Cooperative Information Systems, Vol. 5(1), pp. 1-25, 1996.
- [13] Aknine S. "Modèles et méthodes de coordination dans les systèmes multi-agents". Thèse de doctorat en informatique, Université de Paris IX Dauphine, 2000.
- [14] Imane BOUSSEBOUGH "LES SYSTEMES MULTI-AGENTS DYNAMIQUEMENT ADAPTABLES" Thèse de doctorat Université Mentouri Constantine 2011.
- [15] http://www.memoireonline.com/02/09/1930/m_Integration-de-protocoles-de-securite-pour-la-communication-inter-agents-dans-la-plate-forme-Aglets4.html
- [16] Jean-Pierre Briot et Yves Demazeau Principes et architecture des systèmes multi-agents 11 octobre 2001 .Paris.

- [17] M. Chaker MEZIOUD "Recherche sur la Résolution des Problèmes Complexes d'Affectation de Fréquences Basses Bandes pour les Opérateurs de la Téléphonie Mobile" Thèse de doctorat Université Mentouri de Constantine 2011.
- [18] DROGOUL, A.; Corbara, B.; Lalande, "MANTA: New Experimental Results on the Emergence of (Artificial) Ant Societies", S. 1995.
- [19] B. H. Roth, M. Hewett, R. Washington, R. Hewett, A. Seiver : Distributing intelligence within an individual, Volume 2 of Research Notes in Artificial Intelligence.
- [20] CHAIB-DRAA Brahim: "Agents et systèmes multiagents", Notes de cours, Département d'informatique, Faculté des sciences et de génie, Université Laval, Québec, Novembre 1999.
- [21] B. Chaib-draa, I. Jarras et B. Moulin " Systèmes multiagents : Principes généraux et applications" Article à paraître dans : J. P. Briot et Y. Demazeau « Agent et systèmes multiagents » chez Hermès en 2001
- [22] Stuart Russell & Peter Norvig, P, "Artificial intelligence: A modern approach", second edition, Englewood Cliffs, NJ: Prentice Hall, 1995.
- [23] Franklin,Stan and Graesser, "is it an Agent ,or just a program?:A Taxonomy For Autonomous Agents", Proccedings of the Third International Workshop on Agent Theories,Architecture, And Languages Springer-Verlag 1997.
- [24] Didier Anzieu, Jacques-Yves Martin, 1968, La Dynamique des Groupes Restreints, Puf.
- [25] Thierry Bouron " STRUCTURES DE COMMUNICATION ET D'ORGANISATION POUR LA COOPERATION DANS UN UNIVERS MULTI-AGENTS" Thèse de doctorat l'UNIVERSITE PARIS 6.1992
- [26] http://www-sante.ujf-grenoble.fr/imtc/download/cours/Crs_O-Hansen_SMA_07-08.pdf
- [27] Austin J.L., "How to do thing with words", Oxford University Press, 1962.
- [28] Lahlouhi A., "Méthodologie de développement d'environnements de développement de SMA". Techniques et sciences informatiques, Vol. 1 n°1, 2001.
- [29] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an agent communication language", In Third international conference on information and knowledge management. ACM Press, November 1994.
- [30] Marc-André LABRIE, « Description de comportements d'agents autonomes évoluant dans des mondes virtuels », Université Laval, janvier 2004
- [31] Jean-Paul Sansonnet "ACL Introduction aux formalismes des Langages de Communication Agent" LIMSI-CNRS – jps@limsi.fr
- [32] www.univ-tlemcen.dz/~benmammar/IA3.pd

- [33] Alexandre PAUCHET, « Modélisation cognitive d'interactions humaines dans un cadre de planification coopérative », Université Paris 13, septembre 2004
- [34] TRINH Thanh Hai « Utilisation et optimisation de la communication entre agents dans les cas de tâches collaboratives ou concurrentes » Institut de la Francophonie pour l'Informatique 2007
- [35] N. Minar The swarm simulation system: a toolkit for building multi-agent simulations, 1996, <http://www.santafe.edu/project/swarm>
- [36] B. Azvine, N. Azarmi, , and D. Nauck, (2000). Intelligent Systems and Soft Computing: Prospects, Tools and Applications, volume 1804 of LNCS. Springer- Verlag.
- [37] Plate-forme MADKIT, 2003 <http://www.madkit.org/>
- [38] F Bellifemine, G Caire, T Trucco, et G Rimassa. Jade Programmer's Guide. JADE 3.0b1. Available at <http://sharon.cselt.it/projects/jade/>, 2003.
- [39] www.futura-sciences.com/fr/comprendre/glossaire/definition/t/high-tech-1/d/logiciel_561/
- [40] R. Pawlak, et al., "Programmation orientée aspect pour Java / J2EE", édition Eyrolles, 2004.
- [41] G. J. Myers, C. Sandler, T. Badgett, and T. M. Thomas, "The Art of Software Testing", Second Edition, Wiley, 2004
- [42] T. Gil, "Conception orientée aspect, version 2.1", DotnetGuru, 2007, <http://www.lulu.com/content/8309554>
- [43] L'AFCIQ Agence Française de Contrôle Industriel de la Qualité
- [44] F. Duclos, "Environnement de Gestion de Services Non Fonctionnels Dans les Applications a Composants", Doctoral dissertation, Université Joseph Fourier, Octobre 2002.
- [45] Xanthakis S, Régnier P and Karapoulios C, Le test des logiciels, Hermes sciences publications, 2000.
- [46] [http //fr.wikipedia.org/wiki/Sp%C3%A9cification_\(informatique\)](http://fr.wikipedia.org/wiki/Sp%C3%A9cification_(informatique))
- [47] Department of Defense, "Military Standard Defense System Software Development," February 1988.
- [48] AP.TELLE & S.MILOVANOVIC Test de logiciel GLG101MAI 2007
- [49] <http://www.afis.fr/nm-is/Pages/Evaluation/Evaluation%20de%20la%20qualit%C3%A9%20technique%20-%20V%C3%A9rification%20et%20validation.aspx>

- [50] Glossaire CFTL/ISTQB des termes utilisés en tests de logiciels Version 1.0FR Traduction française de la Version 1.0 produite par 'Glossary Working Party' International Software Testing Qualification Board en date du (08 Décembre, 2004)
- [51] <http://dico.developpez.com/html/1111-Gestion-de-projet-cas-de-test.php>
- [52] <http://dico.developpez.com/html/1118-Gestion-de-projet-jeu-de-test.php>
- [53] Hamlet R, Test du logiciel & confiance, Génie logiciel et systèmes experts, 30, mars 1993.
- [54] Craig, Rick, et Stefan Jaskiel 2002. Systematic Software Testing. Boston: Artech House Publishers.
- [55] Perry, William E. 2000. Effective Methods for Software Testing, 2^e C^e édi. John Wiley and Sons
- [56] <http://dico.developpez.com/html/1097-Gestion-de-projet-cycle-en-V.php>
- [57] I. Burnstein. "Practical software testing: A Process-oriented approach", Ed. Springer, 2002.
- [58] Vu Le Hanh : « Test et modèle UML : stratégies de planification des tests ». Thèse Présentée devant L'université de Rennes pour obtenir le grade de docteur, Mention Informatique. Année 2002.
- [59] I. Jacobson, G. Booch, J. Rumbaugh. "Le Processus unifié de développement logiciel". Ed. Eyrolles, 2003.
- [60] Huey-Der Chu, "An evaluation scheme of software testing techniques", IFIP TC5 WG5.4 3rd international conference on reliability, quality and safety of softwareintensive systems. Athens, Greece. Pages: 259-262, 1997. Ed. Chapman&Hall, Ltd, London, UK.
- [61] Sandrine-Dominique GOURAUD: «Utilisation des structures combinatoires pour le test statistique». Thèse Présentée devant L'université de Paris-Sud-Orsay pour obtenir le grade de docteur, Mention Informatique. Année 2004.
- [62] [59www.infeig.unige.ch/support/se/lect/prg/tst/node14.html](http://www.infeig.unige.ch/support/se/lect/prg/tst/node14.html)
- [63] 67Franck Fleurey " Langage et méthode pour une ingénierie des modèles fiable". Thèse Présentée devant L'université de Rennes 1 pour obtenir le grade de docteur, Mention Informatique , Année 2006.

- [64] Ostrand TJ and Balcer MJ, The category partition method for specifying and generating functional test, communication of the ACM, 31(6):676-686, 1988
- [65] Weis SN and Weyuker EJ, An extended domain based model of software reliability, Transactions on software engineering, 14(10):1512-1524, 1988
- [66] M. C. Gaudel et al., "Précis de Génie Logiciel", Masson, 1996.
- [67] Régis Tissot " Contribution à la génération automatique de tests a partir de modèles et de schémas de test comme critères de sélection dynamiques". Thèse Présentée devant L'université de Franche-Comté pour obtenir le grade de docteur, Spécialité Informatique, Année 2009.
- [68] Christophe GRANDPIERRE" STRATÉGIES DE GÉNÉRATION AUTOMATIQUE DE TESTS À PARTIR DE MODÈLES COMPORTEMENTAUX UML/OCL". Thèse Présentée devant L'université de Franche-Comté pour obtenir le grade de docteur, Spécialité Informatique, Année 2008.
- [69] Mark Utting," Position Paper: Model-Based Testing". The University of Waikato, New Zealand Email: marku@cs.waikato.ac.nz
- [70] Stephan Weißleder," Test Models and Coverage Criteria for Automatic Model-Based Test Generation with UML State Machines" Thèse Présentée devant L'université de Humboldt berlin pour obtenir le grade de docteur, Spécialité Informatique, Année 2009.
- [71] G. Antoniol, L. Briand, M. DiPenta, and Y. Labiche. A case study using the round-trip strategy for state-based class testing. In Proceedings of ISSRE'02 (Int. Symposium on Software Reliability Engineering), Annapolis, MD, USA, 2002. IEEE Computer Society.
- [72] A.J. Offutt and A. Abdurazik. Generating tests from uml specifications. In Proceedings of UML'99 (Unified Modeling Language), Fort Collins, CO, USA, 1999.
- [73] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection : Help for the practicing programmer. IEEE Computer, April 1978.
- [74] Low CK, Chen TY and Rönnquist R, Automated test case generation for BDI agents, Autonomous agents and multi-agent.
- [75] www-lsr.imag.fr/users/Catherine.Oriat/TutorielJML/tutoriel_1.htm
- [76] www-list.cea.fr/labs/fr/LSL/test/docs/fr/memoirePM.pdf
- [77] Zina Houhamdi, Multi-Agent System Testing: A Survey, (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 2, No. 6, 2011

- [78] Huget MP and Demazeau Y, Evaluating multiagent systems: a record/replay approach. *Intelligent Agent Technology, 2004. (IAT 2004)*. Proceedings. IEEE/WIC/ACM International Conference on 2004, pp. 536-539.
- [79] Nguyen, C.D.: *Testing Techniques for Software Agents*, PhD thesis, International Doctorate School in Information and Communication Technologies - University of Trento (2008)
- [80] Moreno, M., Pavón, J., Rosete, A.: Testing in agent oriented methodologies. In: Omatu, S., Rocha, M.P., Bravo, J., Fernández, F., Corchado, E., Bustillo, A., Corchado, J.M. (eds.) *IWANN 2009*. LNCS, vol. 5518, pp. 138–145. Springer, Heidelberg (2009)
- [81] Zhang, Z., Thangarajah, J., Padgham, L.: Automated unit testing for agent systems. In: *2nd International Working Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2007)*, Barcelona, Spain, pp. 10–18 (2007)
- [82] Zhang, Z., Thangarajah, J., Padgham, L.: Automated unit testing intelligent agents in pdt. In: *AAMAS (Demos)*, pp. 1673–1674 (2008)
- [83] Zhang, Z., Thangarajah, J., Padgham, L.: Model based testing for agent systems. In: *AAMAS*, vol. (2), pp. 1333–1334 (2009)
- [84] Padgham, L., Winikoff, M.: *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley and Sons, Chichester (2004)
- [85] Ekinci, E.E., Tiryaki, A.M., Cetin, O., Dikenelli, O.: Goal-Oriented Agent Testing Revisited. In: *Proc. of the 9th Int. Workshop on Agent-Oriented Software Engineering*, pp. 85–96 (2008)
- [86] Lam, D.N., Barber, K.S.: Debugging agent behavior in an implemented agent system. In: Bordini, R.H., Dastani, M.M., Dix, J., El Fallah Seghrouchni, A. (eds.) *PROMAS 2004*. LNCS (LNAI), vol. 3346, pp. 104–125. Springer, Heidelberg (2005)
- [87] Nunez, M., Rodríguez, I., Rubio, F.: Specification and testing of autonomous agents in e-commerce systems. *Software Testing, Verification and Reliability* 15(4), 211–233 (2005)
- [88] Coelho, R., Kulesza, U., von Staa, A., Lucena, C.: Unit testing in multi-agent systems using mock agents and aspects. In: *SELMAS 2006: Proceedings of the 2006 International Workshop on Software Engineering for Large-scale Multi-agent Systems*, pp. 83–90. ACM Press, New York (2006), <http://dx.doi.org/http://doi.acm.org/10.1145/1138063.1138079>
- [89] Gamma, E., Beck, K.: *JUnit: A Regression Testing Framework* (2000), <http://www.junit.org>
- [90] Nguyen, C.D., Perini, A., Tonella, P.: Ontology-based Test Generation for Multi Agent Systems. In: *Proc. of the International Conference on Autonomous Agents and Multiagent Systems* (2008)

- [91] Nguyen, C.D., Miles, S., Perini, A., Tonella, P., Harman, M., Luck, M.: Evolutionary testing of autonomous software agents. In: The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), pp. 521–528. IFAAMAS (2009)
- [92] Serrano, E., Botia, J.A.: Infrastructure for forensic analysis of multi-agent systems. In: Hindriks, K.V., Pokahr, A., Sardina, S. (eds.) ProMAS 2008. LNCS, vol. 5442, pp. 168–183. Springer, Heidelberg (2009)
- [93] Serrano, E., Gómez-Sanz, J.J., Botia, J.A., Pavon, J.: Intelligent data analysis applied to debug complex software systems. *Neurocomputing* 72(13-15), 2785–2795 (2009)
- [94] Tiriyaki, A.M., Öztuna, S., Dikenelli, O., Erdur, R.C.: SUNIT: A unit testing framework for test driven development of multi-agent systems. In: Padgham, L., Zambonelli, F. (eds.) AOSE VII / AOSE 2006. LNCS, vol. 4405, pp. 156–173. Springer, Heidelberg (2007)
- [95] Dikenelli, O., Erdur, R.C., Gumus, O.: Seagent: a platform for developing semantic web based multi agent systems. In: AAMAS 2005: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1271–1272. ACM Press, New York (2005), <http://dx.doi.org/http://doi.acm.org/10.1145/1082473.1082728>
- [96] Gomez-Sanz, J.J., Botía, J., Serrano, E., Pavón, J.: Testing and debugging of MAS interactions with INGENIAS. In: Luck, M., Gomez-Sanz, J.J. (eds.) AOSE 2008. LNCS, vol. 5386, pp. 199–212. Springer, Heidelberg (2009)
- [97] Nguyen, C.D., Perini, A., Tonella, P.: Goal-oriented testing for MASs. *Int. J.Agent-Oriented Software Engineering* 4(1), 79–109 (2010)
- [98] Anu Maria: Introduction to Modelling and Simulation, Proceedings of the 29th conference on Winter simulation, Atlanta, Georgia, United States, Pages: 7 – 13, 1997
- [99] Pierre-Alain Muller De la modélisation objet des logiciels à la metamodélisation des langages informatiques, Thèse de HDR présentée devant L'Université de Rennes.
- [100] Robert Valette, "Qu'est ce qu'un bon modèle ?" LAAS-CNRS F- 31077 Toulouse Cedex 4 e-mail robert@laas.fr <http://www.laas.fr/> robert Version Juillet 1999
- [101] Laurent Audibert : UML 2.0, Institut Universitaire de Technologie de Villetaneuse, Département Informatique, Adresse du document : <http://www-lipn.univ-paris13.fr/audibert/pages/enseignement/cours.htm>, novembre 2007.
- [102] TRAN Vinh Duc, Victor MORARU Réseau de Petri Institut de la Francophonie pour l'Informatique - Promotion 10 15 juillet 2005.

[103] C. A. Petri. "Kommunikation mit Automaten". Bonn : Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962 (en allemand) aussi dans : New York : Griffiss Air Force Base, Technical Report RADC-TR-65--377, Vol. 1, pp. Suppl. 1. 1966. (traduction anglaise).

[104] CHRISTINE BALAGUE,"LES SYSTÈMES MULTI-AGENTS EN MARKETING : MODÉLISATION PAR LES RÉSEAUX DE PETRI" Thèse Présentée devant L'université France, pour l'obtention du titre de Docteurs en sciences de gestion 2005.

[105] TADAO MURATA "Petri Nets: Properties, Analysis and Applications" Fellow, IEEE,Invited Paper. Prodeedings of IEEE, vol. 77, NO 4, April 1989.

[106] Alexandre Pauchet," Modélisation des Systèmes Complexes Introduction aux Réseaux de Petri" 800A-2, pauchet@insa-rouen.fr INSA Rouen - Département ASI.

[107] http://www.tn.refer.org/hebergement/cours/sys_disc/notions_de_base_RdP.html

[108] G. Scorletti et G. Binet Réseaux de Petri Maîtres de conférences à l'Université de Caen, France 20 juin 2006. Page web: http://www.greyc.ensicaen.fr/EquipeAuto/Gerard S/mait_Petri.html.

[109] Rémi Bastide, « Spécification comportementale par réseaux de Petri : application aux systèmes distribués à objets et aux systèmes interactifs », Habilitation à diriger des recherches de l'Université Toulouse 1, 2000.

[110] G. W. Brams ; nom collectif de C. Andre, G. Berthelot, C. Girault, G. Memmi, G. Roucairol, J. Sifakis, R. Valette, G. Vidal-Naquet. "Réseaux de Petri : Théorie et Pratique. Tome 1 : Théorie et Analyse ; Tome 2 : Modélisation et Applications". Editions Masson, septembre 1983.

[111] R. David, H. Alla. "Du Grafctet aux Réseaux de Petri", Edition Hermès, Paris, 1992.

[112] S. LAFTIT, J.M. PROTH and X.L. XIE, "Performance Analysis of Stochastic Jop-Shop Systems Using Event Graphs", ECC 91, Grenoble, France, July 2-5 1991, pp. 1484-1489.

[113] Wael KHANSA," RESEAUX DE PETRI P-TEMPORELS CONTRIBUTION A L'ETUDE DES SYSTEMES A EVENEMENTS DISCRETS" Thèse Présentée devant L'université de SAVOIE, pour l'obtention du titre de Docteurs. Spécialité : ELECTRONIQUE - ELECTROTECHNIQUE – AUTOMATIQUE 1997.

[114] Patrice BONHOMME," RESEAUX DE PETRI P-TEMPORELS:CONTRIBUTIONS A LA COMMANDE ROBUSTE", Thèse Présentée devant L'université SAVOIE, pour l'obtention du titre de Docteurs. Spécialité : Spécialité : ELECTRONIQUE - ELECTROTECHNIQUE – AUTOMATIQUE 2001.

- [115] Jensen Kurt. Coloured petri nets: Basic concepts, analysis methods and practical use, volume 1. Springer, 1997.
- [116] Valk R, Petri nets as token objects: An introduction to elementary object nets, In Jörg Desel and Manuel Silva (Eds.), Application and Theory of Petri Nets, volume 1420 of LNCS, pages 1–25, June 1998.
- [117] Valk R, Concurrency in communicating object Petri nets, in G. Agha, F. De Cindio, and G. Rozenberg (Eds.), Concurrent Object-Oriented Programming and Petri Nets, volume 2001 of LNCS, Springer Verlag, Berlin, 2001.
- [118] Yacine KISSOUM, " TEST DES SYSTEMES MULTI-AGENTS", Thèse Présentée devant L'université de MENTOURI CONSTANTINE, pour l'obtention du titre de Docteurs Science en Informatique 2010.
- [119] Kummer O, Simulating synchronous channels and net instances, in J. Desel, P. Kemper, E. Kindler, and A. Oberweis (Eds.), Forschungsbericht Nr. 694: 5. Workshop Algorithmen und Werkzeuge für Petrinetze, pp. 73–78. University of Hamburg, Computer Science Department, 1998.
- [120] Kummer O, Wienberg F and Duvigneau M, Renew – User Guide, University of Hamburg, Computer Science Department, Vogt-Kölln Str. 30, 22527 Hamburg, Deutschland, 1.6 edition, 2002
- [121] Köhler M and Rölke H, Towards a Unified Approach for Modeling and Verification of Multi-agent Systems, In Daniel Moldt (Ed.): Workshop on Modelling of Objects, Components, and Agents (MOCA'01), pages 85- 104, August 2001.
- [122] Köhler M and Rölke H, Mobile object net systems: Concurrency and mobility, in Proceedings of the International Workshop on Concurrency, Specification, and Programming (CS&P 2002), Berlin, 2002.
- [123] Christensen S, and Hansen ND, Colored Petri nets extended with channels for synchronous communication, In Robert Valette (Ed.), Application and Theory of Petri Nets 1994, Proc. of 15th Intern. Conf. Zaragoza, Spain, June 1994, LNCS, pages 159–178, Springer Verlag, Berlin.
- [124] Köhler M, Moldt D and Rölke H, Modelling Mobility and Mobile Agents Using Nets within Nets, ICATPN 2003, LNCS 2679, pp. 121–139, Springer-Verlag Berlin 2003
- [125] FIPA Agent Management Support for Mobility Specification, 30. June 2000. Available at <http://www.fipa.org/specs/fipa00087/>.
- [126] Cabri G, Leonardi L, and Zambonelli F, Weak and strong mobility in mobile agent applications, in Proceedings of the 2nd International Conference and Exhibition on The Practical Application of Java (PA JAVA 2000), Manchester (UK), April 2000.

- [127] http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/complete_db.html
- [128] Dianxiang Xu and Yi Deng, "Modeling Mobile Agent Systems with High Level Petri Nets", Proc. of the IEEE International Conference on Systems, Man, and Cybernetics (SMC'00), pp. 3177-3182, Nashville, October 2000.
- [129] Weyns D., Holvoet T. "A Colored Petri Net for a Multi-Agent Application", Proceeding of MOCA'2002. Organised by the CPN group at the Department of Computer Science, University of Aarhus, and Denmark.2002.
- [130] Michael Köhler, Marcel Martens, and Heiko Rolke, "Modelling Social Behaviour with Petri net based Multi-Agent Systems", In Proceedings of the Workshop MASHO'03 at the KI 2003, 2003.
- [131] J.L Koning, G.François et Y.Demazea. "Formalization and pre-validation for interaction protocols in multi agent systems", In 13th European Conference on Artificial Intelligence, Brighton, 1998.
- [132] A.El fallah seghrouchni,S.Haddad,H.Mzouzi , "Protocol Engineering for multiagent Interaction",9th European Workshop on Modeling Autonomous Agents in a Multi-agents Word MAAMAw99 ,LNCS 1647 ,Espagne juillet 1999.
- [133] COST S.R., CHEN Y., FININ T., LABROU Y. et PENG Y. Modeling agent conversations with colored Petri nets. In Agents'99: Working Notes of the Workshop on Specifying and Implementing Conversation Policies. 1999. Seattle (USA).
- [134] B.Mazouzi,"Ingenierie des protocoles d'interaction : des systems distributes aux systyèmes multi-agents ",Theses de Doctorat,Université Paris IX-Dauphine.2001
- [135] Kissoum Y, Sahnoun Z and Barkaoui K,: Model-based testing approach for mobile agents using the paradigm of reference net, 2010.
- [136] W.Prenninger, M.ElRamly and M.Horstmann,: Model based testing of reactive systems, in Springer Berlin/Heidelberg Eds, LNCS 3472, 2005, 439–461.
- [137] FIPA agent communication languages specifications, Available at www.fipa.org/repository.
- [138] Duvigneau M, Moldt D, and Rölke H, Concurrent architecture for a multi-agent platform, In Proceedings of the Workshop on Agent Oriented Software Engineering (AOSE'02), volume 2585 of Lecture Notes in Computer Science, Springer Verlag, Berlin, 2003.
- [139] P.Gauthier : Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente Thèse en Informatique Institut de Recherche en Informatique de Toulouse (IRIT) Université Paul Sabatier (UPS) 10 décembre 2004.
- [140] R.Vaughan, K.Stoy, G.Sukhatme and M.Matric,Go ahead make my day : Robot conflict resolution by aggressive competition, Dans Proceedings of the 6th International Conference on Simulation of Adaptive Behaviour, 2000

[141] M.Kolp, P.Giorgini, J.Mylopoulos, Organizational Multi-Agent Architectures: A Mobile Robot Example AAMAS'02, July 15-19, 2002, Bologna, Italy. 2002.