Une approche formelle basée BRS pour la spécification et la vérification des architectures des systèmes Multi-Agents

Ahmed Taki Eddine Dib

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Constantine 2 - Abdelhamid Mehri
Faculté des Nouvelles Technologies de l'Information et la Communication
Département de Technologies des Logiciels et des Systèmes d'Information

Année : 2017
N° d'ordre : 2017/-/DR.LMD/UC2
Série : 2017/-/DR.LMD/NTIC

T H È S E

Pour l'obtention du diplôme de Docteur en Informatique

Option : Ingénierie des systèmes d'information

Présentée et soutenue par : Ahmed Taki Eddine DIB

Une approche formelle basée BRS pour la spécification et la vérification des architectures des systèmes multi-agents

Dirigée par : Pr. Zaidi SAHNOUN

Devant le jury composé de :

Pr. Mohamed BATOUCHE	Université Constantine 2 - Abdelhamid Mehri	Président
Pr. Zaidi SAHNOUN	Université Constantine 2 - Abdelhamid Mehri	Directeur
Pr. Okba KAZAR	Université Mohamed Khider - Biskra	Examinateur
Pr. Faiza BELALA	Université Constantine 2 - Abdelhamid Mehri	Examinateur
Dr. Laid KAHLOUL	Université Mohamed Khider - Biskra	Examinateur
Pr. Kamel BARKAOUI	Conservatoire National des Arts et Metiers - Paris	Invité

Soutenue le - Septembre 2017

"Ne donnez pas ce que vous-même n'accepteriez pour vous qu'en fermant les yeux".

Le Coran, II, 267

Remerciements

Au terme de ce long parcours qu'est le doctorat, je tiens à adresser mes remerciements à l'ensemble des personnes qui par leurs contributions ont permis l'aboutissement de ce travail.

Je tiens, tout d'abord, à remercier mes deux directeurs de thèse pour la confiance qu'ils m'ont témoignée, et pour le climat de sérénité qu'ils ont entretenu durant ces années. Ma reconnaissance à Monsieur Zaidi SAHNOUN, professeur à l'Université Constantine 2 - Abdelhamid Mehri, pour avoir accepté de diriger mes travaux de recherche. Ses conseils, sa disponibilité et sa patience depuis mes premiers pas jusqu'à l'intégration totale dans la recherche m'ont été très précieux. J'adresse également mes vifs remerciements à Monsieur Kamel BARKAOUI, professeur du Conservatoire Nation des Arts et Metiers (CNAM-Paris) pour sa contribution au sein de l'encadrement de cette recherche dont il a largement participé à son évolution.

Je tiens à remercier Monsieur Mohamed Chawki BATOUCHE, professeur à l'Université Constantine 2 - Abdelhamid Mehri, pour m'avoir fait l'honneur d'accepter de présider le jury de ma thèse. Je suis particulièrement sensible, par ailleurs, au grand honneur dont m'ont gratifié Professeur Okba KAZAR de l'Université Mohamed Khider - Biskra, Monsieur Laid KAHLOUL, maître de conférences à de l'Université Mohamed Khider - Biskra et Madame Faiza BELALA, professeur à l'Université Constantine 2 - Abdelhamid Mehri, pour l'intérêt qu'ils ont bien voulu porter à mes travaux en acceptant d'en être les rapporteurs, qu'ils trouvent ici l'expression de ma profonde reconnaissance.

Ma gratitude infinie va aussi à ma famille et surtout mes parents qui m'ont soutenu tout au long de mes études. Je les remercie pour leurs encouragements, leur dévouement et leur soutien inconditionnel durant toutes ces années.

دَعْوَىٰهُمْ فِيهَا سُبْحَانَكَ اللَّهُمَّ وَتَحِيَّتُهُمْ فِيهَا سَلَامٌ ۚ
وَآخِرُ دَعْوَىٰهُمْ أَنِ الْحَمْدُ لِلَّهِ رَبِّ الْعَالَمِينَ

Résumé :

Le paradigme multi-agent (SMA) peut être particulièrement adapté à la construction de systèmes distribués, ouverts et concurrents. Il est reconnu, comme étant un moyen excellent et adéquat pour développer une variété d'applications industrielles de grande envergure dans différents domaines, tels que : la gestion des réseaux distribués, l'informatique en grille, les services web intelligents, etc. Ces systèmes doivent souvent opérer dans des environnements dynamiques et faire face au défi de l'évolution constante de leurs exigences ; donc ils doivent être flexibles, robustes et capables de s'adapter à leurs environnements. Ceci, rend difficile la conception et le développement des systèmes multi-agents reconfigurables.

Dans ce travail, nous nous basons sur les méthodes formelles pour gérer la complexité des aspects liés à la conception des architectures de SMA. Plus précisément, nous proposons une approche de modélisation formelle basée sur les systèmes réactifs bigraphiques (BRS) pour la spécification et la vérification des architectures des systèmes multi-agents et leur reconfiguration. Les BRS présentent une solution prometteuse et efficace pour le développement des SMA sûrs et de qualité. En outre, ils permettent la vérification des propriétés pertinentes des SMA à l'aide d'outils de vérification de modèles bigraphiques.

Mots clés : Méthodes formelles, modélisation, systèmes réactifs bigraphiques, reconfiguration, systèmes multi-agents, vérification formelle.

Abstract :

Multi-Agent System (MAS) paradigm, can be particularly suitable to build distributed, open and concurrent systems. In fact, it has shown to be an excellent and adequate way to develop a variety of large and complex industrial applications across different domains, such as distributed networks management, grid computing, smart web services, etc. Indeed, these systems must be designed to meet increasingly new requirements. Therefore, they must be flexible, robust and capable to adapt to their environments. However, the diversity of the basics, on one hand, and the complexity of the concepts related to agents, on the other hand, make it difficult to conceive and develop reconfigurable multi-agent systems.

In this work, we rely on formal methods to manage the complexity of these aspects and reason on the design of MAS architectures. Precisely, we propose a formal modelling approach based on Bigraphical Reactive Systems (BRS) for the specification of multi-agent system architectures and their reconfiguration. BRS present a promising and effective solution for the development of high quality and safe MAS at reasonable cost and time span. In addition, they enable the verification of MAS relevant properties using a bigraphical model checking tool named BigMC.

Keywords : Formal methods, modelling, bigraphical reactive systems, reconfiguration, Multi-Agent System, behaviour, formal verification.

ملخص:

نموذج النظم المتعددة الوكلاء، يمكن أن يكون مناسبا بشكل خاص لبناء الأنظمة الموزعة، المفتوحة والمتزامنة. في الواقع، فقد تبين أنها طريقة ممتازة وكافية لتطوير مجموعة متنوعة من التطبيقات الصناعية الكبيرة والمعقدة وذلك في مجالات مختلفة، مثل إدارة الشبكات الموزعة، والحوسبة الشبكية، وخدمات الويب الذكية، وما إلى ذلك .أكيد أنه، يجب تصميم هذه الأنظمة تلبية لمتطلبات جديدة على نحو متزايد .ولذلك، يجب أن تكون مرنة، قوية وقادرة على التكيف مع بيئاتها .ومع ذلك، فإن تنوع الأساسيات، من جهة، وتعقيد المفاهيم المتعلقة بالعوامل، من ناحية أخرى، يجعل من الصعب تصور وتطوير أنظمة متعددة الوجوه قابلة لإعادة التشكيل.

في هذا العمل، نعتمد على الطرق الرسمية لإدارة تعقيد تصميم معماريات النظم متعددة الوكلاء .على وجه التحديد، نقترح نهج النمذجة الرسمية على أساس النظم التفاعلية البيغرافية (برس) لتحديد معمارية نظم متعددة العوامل وإعادة تشكيلها .تقدم النظم التفاعلية البيغرافية، حلا واعد ا وفعالا لتطوير نظم متعددة الوكلاء آمنة وذات جودة عالية، في وقت وبتكلفة معقولين .وبالإضافة إلى ذلك، فإنها تمكن من التحقق من خصائص النظم متعددة الوكلاء باستخدام أداة فحص.

الكلمات المفتاحية: أنظمة متعددة الوكلاء؛ إعادة تشكيل؛ مواصفات رسمية؛ التحقق الرسمي؛ النظم التفاعلية البيغرافية .

Liste des abréviations

ACL	Agent Communication Language
ADEPT	Advanced Decision Environment for Process Task
ADL	Architecture Description Language
ADLMAS	Architecture Description Language for MAS
AGR	Agent-Groupe-Rôle
AOSE	Agent-Oriented Software Engineering
BDI	Belief-Desire-Intention
BigMC	Bigraphical Model Checker
BigraphER	Bigraph Evaluator and Rewriting
BiLog	Bigraphical logics
BPEL	Business Process Execution Language
BPL	Bigraphical programming language
BPL-Tool	Bigraphical Programming Language project Tool
BPMN	Business Process Model and Notation
BRS	Bigraphical Reactive Systems
CXM	Communicating X-Machines
DB	Data Base
FIPA	Foundation for Intelligent Physical Agents
GL	Génie Logiciel
GPS	Global Positioning System
MAS	Multi-Agent System
IA	Intelligence Artificielle
IAD	l'intelligence artificielle distribuée
JADE	Java Agent DEvelopment Framework
KQML	Knowledge Query and Manipulation Language
OMAS	Open Multi-Agent System
OperA	Organisations per agent
SNMP	Simple Network Management Protocol
SMA	Systéme Multi-Agent
UML	Unified modeling language
XHTML	Extensible hyperText markup language
XML	Extensible markup language

Table des matières

Table des figures

Liste des tableaux

Listings

Chapitre 1

Introduction générale

Sommaire

Les caractéristiques et les attentes des nouvelles applications d'entreprise, tels que le e-business, la gestion des connaissances, les services Web et l'informatique en nuage, ont profondément modifié l'ingénierie des systèmes d'information. La plupart des solutions conçues, pour ces types d'applications sont de fait, concurrentes et distribuées. De plus, elles ont tendance à être ouvertes et dynamiques (car il existe une évolution organisationnelle et opérationnelle de l'environnement où de nouveaux composants peuvent être ajoutés, modifiés ou supprimés à tout moment). Il est également important que de tels systèmes soient sûrs, étant donné qu'ils sont souvent utilisés dans la gestion et le stockage d'informations sensibles, telles que les informations médicales, financières etc...

Compte tenu de ces besoins, de nombreux chercheurs sont en quête de paradigmes leur permettant de concevoir et de mettre en œuvre des systèmes d'information qui peuvent fonctionner de manière effective dans de telles conditions. En effet, la plupart de ces problèmes qui sont de nature complexe, sont résolus en utilisant des environnements distribués. De plus, il existe actuellement une forte pression pour concevoir et développer ces systèmes dans un laps de temps limité.

Une approche habituellement utilisée pour accélérer le développement des systèmes distribués, consiste à réutiliser des composants logiciels déjà développés. En conséquence, un grand système distribué peut être développé à travers (i) l'identification de composants logiciels réutilisables, (ii) la personnalisation et l'intégration de ces composants avec les composants logiciels nouvellement développés, ceci afin de répondre aux nouvelles exigences du système en cours de développement.

Pour une meilleure gestion, du développement de logiciels complexes dans des environnements distribués, et concurrents ; l'utilisation des systèmes multi-

agents (SMA) est préconisé pour construire les systèmes d'information des entreprises d'aujourd'hui.

1.1 Contexte et Problématique

Les systèmes multi-agents (SMA) sont reconnus comme un domaine majeur de l'intelligence artificielle distribuée, et prédit pour qu'ils soient le futur paradigme pour le développement des nouveaux systèmes d'information, car, ils offrent aux développeurs une manière souple pour structurer des systèmes autour d'éléments autonomes et communicants. En effet les SMA sont des systèmes adaptatifs et flexibles capables de s'adapter aux changements de leurs environnements, où des agents peuvent être ajoutés ou supprimés au moment de l'exécution, [Jiao 2003].

Le domaine des systèmes multi-agents a fait l'objet d'études approfondies lors de cette dernière décennie. Un SMA peut être défini comme un système, décentralisé et complexe, composé d'agents communicants de façon asynchrone les uns avec les autres. Ces agents sont des programmes informatiques, qui agissent de manière autonome pour le compte d'une personne ou d'une organisation, tout en coordonnant leurs activités. Les systèmes multi-agents ont été jugés comme une solution efficace aux problèmes informatiques décentralisés, situés dans des environnements dynamiques, ouverts et ayant des capacités d'intelligence tels que : l'apprentissage [Haynes 1998], la négociation [Kraus 1995, Rosenschein 1998], le comportement social [Sandhlom 1997, Shehory 2014b, Shoham 1992, Demazeau 1990], les activités mentales (croyances, désirs, intentions [Rao 1995] et le raisonnement [Halpern 1997]). En effet, il y a un nombre considérable d'applications réelles qui ont été déployées avec succès dans divers secteurs d'application telles que : les télécommunications, la gestion des grilles et la robotique [Müller 2014]. De plus, le domaine des systèmes multi-agents est soutenu par un grand nombre de chercheurs et de praticiens avec des conférences réussies telles que AAMAS, IAT, MATES, et PAAMS, et avec des journaux comme JAAMAS, AAAI, AAIJ, et KER [Müller 2014].

Toutefois, l'utilisation généralisée des SMA est actuellement freinée par le manque de méthodes pour assurer leur fiabilité. Le changement fréquent dans les attentes des acteurs économiques ainsi que la croissance continuelle de la complexité des systèmes, des technologies et des organisations, à tout cela s'ajoute l'aspect dynamique de l'environnement d'aujourd'hui, font du développement des systèmes multi-agents, une tâche d'ingénierie complexe. En effet, les développeurs sont sollicités pour fournir des architectures qui doivent prendre en compte les exigences les plus complexes. Un problème critique dans la conception et la construction de tout système d'information complexe, de-

meure son architecture, c'est-à-dire sa structure organisationnelle définie en tant qu'une collection de composants interactifs. Assurément, une conception architecturale rigoureuse peut garantir qu'un système répond à des exigences clés telles que la performance, la fiabilité, la portabilité, l'évolutivité et l'interopérabilité [Shaw 1996]. Pour aider les développeurs à spécifier des architectures dédiées pour les systèmes d'information, les langages de description d'architecture sont utilisés. Un langage de description d'architecture (ADL) fournit une syntaxe concrète pour spécifier des abstractions architecturales dans une notation descriptive. Les abstractions architecturales concernent la structure des composants du système, leurs comportements et leurs interrelations. Un ensemble de langages formels d'architecture (ADL) tels que Darwin, Rapide, Dynamic-Wright [Allen 1998] et π-ADL [Oquendo 2004] ont été proposés pour la représentation et l'analyse des architectures logicielles. Cependant, ces ADL ne sont pas adaptés pour représenter pleinement les caractéristiques des architectures SMA telles que les interactions complexes, la perception et les reconfigurations dynamiques.

En effet, il existe beaucoup de difficultés pour pouvoir concevoir et analyser la structure et les comportements de SMA. Dans un SMA critique, l'exécution incorrecte de ces activités peut avoir des conséquences irréversibles et par conséquent dévastatrices. Toutefois, assurer l'exactitude des interactions complexes est une question difficile en raison des erreurs causées par les déconnexions d'agents, l'allocation dynamique de rôles et l'autonomie du comportement de l'agent. Pour relever ces défis, nous avons besoin d'approches de modélisation formelle pour soutenir la conception et la mise en œuvre de SMA, et assurer que les systèmes développés soient robustes, fiables, vérifiables et efficaces [Xu 2003]. Le manque de rigueur est reconnu comme l'un des principaux facteurs qui entrave l'adoption à grande échelle de la technologie multi-agents [Brazier 1997]. Une approche rigoureuse à haut niveau d'abstraction et orientée vers une conception de l'architecture de SMA permet de détecter et éliminer les erreurs de conception au début du cycle de développement, et ainsi réduire de manière sensible le coût de développement.

Au cours de ces dernières années, certains travaux de recherches menés sur la modélisation et l'analyse de système multi-agent montrent que la plupart de ces approches sont : ad-hoc, complexes et non fiables. Ces efforts souffrent d'un manque d'approches systématiques fondées sur des méthodologies de développement génie logiciel. Néanmoins, de nouvelles approches à caractère académique sont développées. Ces approches proposent des solutions abstraites mais non-génériques ; les formalismes utilisés ne sont pas dédiés à la modélisation des systèmes multi-agents et offrent une seule "vue " du système qui n'est pas suffisante.

1.2 Objectifs et Contributions

La problématique que nous posons est donc celle de **la formalisation des propriétés statiques et dynamiques des architectures de SMA à travers un modèle unique et générique**, applicable à tous types de systèmes, totalement ou partiellement spécifiés. Plus précisément, Ce travail de thèse s'inscrit dans le cadre de la spécification et la vérification des architectures de système multi-agent ; nous mettons en avant l'importance de proposer une approche architecturale de SMA. Néanmoins, proposer une approche pour la conception d'architectures de système multi-agent n'est pas une tâche simple et le succès d'une solution dépend de nombreux facteurs allant des concepts, (c'est-à-dire les concepts sont-ils assez simples à adopter par les développeurs ?) ; à la technologie (la technologie est-elle prête à supporter la mise en œuvre des concepts proposés ?) ; à la communauté (existe-t-il une communauté active soutenant une solution particulière ?) ; à l'expérience de l'utilisateur (c'est-à-dire est-ce que la solution est facile à utiliser par les utilisateurs ?) etc...

En conséquence, dans ce travail, nous proposons une approche abstraite qui met l'accent sur la séparation des préoccupations entre les différents niveaux conceptuels et œuvre pour une meilleure réutilisation du modèle. Ainsi que la couverture de concepts telles que : l'autonomie, la concurrence et la capacité de prise de décisions qui représentent les principales propriétés de l'approche agent, où les agents peuvent coordonner et négocier les uns avec les autres dans un environnement distribué.

Nous sommes fermement convaincus qu'une solution efficace à ce problème se situe quelque part entre les systèmes multi-agents et les ADL. Plus précisément, nous pensons que la solution optimale devrait utiliser des concepts du génie logiciel à travers les langages de description d'architectures pour concevoir et guider le développement des architectures de systèmes multi-agents. De plus, l'utilisation des bigraphes comme base théorique et formelle représente un outil de modélisation graphique et mathématique, à la fois simple et très expressive dans la représentation de la structure et des comportements dynamiques de système. Un avantage notable de l'utilisation des bigraphes est leur approche modulaire qui permet la spécification d'un système logiciel complexe à l'aide de concepts (1) "Mobile Locality" de distribution spatiale des entités physiques ou logicielles, et (2) "Mobile Connectivity" d'interaction entre l'ensemble de ces entités. Plus important encore, les bigraphes permettent l'analyse formelle de l'architecture SMA à travers des techniques bien établies, tels que la simulation, la détection de blocage, l'analyse de l'accessibilité et le model-checking.

En effet, l'objectif principal de la thèse est de fournir au développeur un

modèle générique et exhaustive (supportant les concepts de base des architectures de SMA) tout en restant raisonnablement analysables (qui aide à réduire la complexité de modélisation et d'analyse des systèmes). Concrètement, nous nous sommes basés sur les systèmes réactifs bigraphiques pour fournir une approche formelle permettant de modéliser et vérifier les aspects structurels et comportementaux des architectures de SMA. Pour ce faire, un nombre de contributions, a été proposé :

- La proposition du model BRS-MAS, qui est un modèle architectural bigraphique flexible, abstrait et générique. Ce modèle est flexible car il offre au concepteur la liberté de choix lors de la conception de son SMA, c.-à-d. le concepteur peut, construire son SMA en suivant une approche top-down ou bottom-up. Abstrait, par les différents niveaux conceptuels qu'il propose et générique car il permet de modéliser plusieurs types d'architectures de SMA (par exemple : les architectures modulaires, hiérarchiques, etc...).
- La proposition du modèle BDI-MAS qui est une instance de notre approche générique BRS-MAS appliquée sur des agents de type BDI. Une architecture en deux couches est suivie pour décrire notre SMA. Donc, un système multi-agent est spécifié à la fois au niveau individuel (Agent) et social (SMA) ce qui simplifie considérablement la lisibilité et donc l'analyse des architectures SMA. De plus, un ensemble de règles de réaction est proposé pour modéliser le comportement de l'SMA sur les deux niveaux, agent et social.
- La vérification ainsi que la validation de la faisabilité de l'approche présentée sont effectuées à travers l'utilisation d'outils de vérification sur un système de commerce électronique.

1.3 Organisation du manuscrit

Cette thèse est organisée en deux parties distinctes : la partie I présente les concepts fondamentaux, des SMA, et de la théorie des bigraphes ; La partie II étudie leur adoption comme outil de modélisation pratique pour les architectures de SMA. Les chapitres de ce manuscrit sont organisés de la manière suivante :

Dans la première partie, à la suite de l'introduction, nous présentons dans le chapitre 2 l'état de l'art, suivant les axes définis ci-dessous : les méthodes de conception, les modèles organisationnels, les modèles architecturaux et les modèles de reconfiguration des SMA. Le chapitre 3 propose une description des différents concepts liés aux systèmes réactifs bigraphiques, nécessaires

pour comprendre les contributions proposées dans la suite de cette thèse.

Dans une deuxième partie, nous exposons la contribution apportée par notre travail [Dib 2016, Dib 2015, Dib 2014]. Elle s'organise comme suit :
- Dans le chapitre 4, nous exposons la formalisation et la sémantique de notre modèle architectural. En présentant le modèle BRS-MAS.
- Dans le chapitre 5, nous proposons une méthode de conception des architectures de SMA fondée sur des agents de type BDI.
- Dans le chapitre 6, nous étudions l'applicabilité de nos propositions à travers la vérification de propriétés sur un système de commerce électronique.

Nous terminons ce manuscrit par une conclusion générale ainsi qu'une discussion sur les résultats de notre travail de recherche. Pour finalement conclure par dégager les voies ou les directions de recherche à entreprendre dans le futur.

Chapitre 2

Les Systèmes Multi-Agent

Sommaire

2.1 Introduction

Les agents et les systèmes multi-agents (SMA) sont une branche de l'Intelligence Artificielle (IA) qui tentent de combiner l'IA, les systèmes distribués et l'ingénierie logicielle dans une seule et même discipline. Depuis plus de trois décennies, les agents et les SMA ont été étudiés, mis en œuvre et évalués.

Des efforts considérables ont été investis dans la recherche théorique et pratique afin de faciliter le transfert des agents de la science vers l'ingénierie et des laboratoires vers le terrain. Pour faciliter l'ingénierie, la communauté du génie logiciel orienté agent a fourni des méthodes, des méthodologies et des outils pour appuyer le développement des agents et des SMA. Plusieurs

spécifications, standards ont été délivrées par la FIPA (Foundation for Intelligent Physical Agents) visant à faciliter le développement et le déploiement des agents et des SMA dans la pratique.

Les agents et le SMA introduisent des concepts et des abstractions dont la combinaison apporte une nouvelle approche pour la conception et la mise en œuvre de systèmes logiciels. Dans ce chapitre, nous présentons les principes fondamentaux de l'agent et des SMA.

2.2 Définitions

Le terme "agent" est probablement l'un des mots les plus controversés dans l'ensemble du domaine de recherche. Nwana a déclaré : « nous avons autant de chances de convenir d'une définition consensuelle pour le mot agent de même qu'ont les chercheurs en IA pour donner une définition à l'intelligence artificielle elle-même » [Nwana 1996]. Différentes définitions pour le concept agent existent, chacune ayant des influences différentes (inspiré de l'intelligence artificielle, des systèmes distribués, des sciences sociales et du génie logiciel). ([Shoham 1993, Franklin 1997, Poslad 2000]).

Auteurs & Année	Définition en français
[Shoham 1993]	« Une entité dont l'état est considéré constitué de composantes mentales tels que les croyances, les capacités, les choix et l'engagement. Ce qui rend n'importe quel matériel ou logiciel un agent ».
[Wooldridge 2009]	« Un agent est un système informatique qui est situé dans un environnement donné et qui est capable d'agir de manière autonome dans cet environnement afin d'atteindre ses objectifs de conception ».
[Russell 1995]	« Un agent est tout ce qui peut être considéré comme percevant son environnement par des capteurs et agissant sur cet environnement à travers des effecteurs ».
[Ferber 1995]	« Une entité réelle ou virtuelle, opérant dans un environnement, et qui est capable de percevoir et d'agir sur cet environnement, elle peut communiquer avec d'autres agents, qui manifestent un comportement autonome, qui peut être vu comme la conséquence de ces connaissances, interactions avec d'autres agents et des objectifs qu'elle poursuit. »
[Luck 2005]	« Un agent est un système informatique capable d'action autonome flexible dans des domaines dynamiques, imprévisibles, typiquement multi-agents ».

TABLE 2.1 – Définitions d'un agent

2.3 Propriétés d'un agent

Un agent logiciel est une entité logicielle qui exécute des tâches pour le compte d'une autre entité, qu'il s'agisse d'un logiciel, d'un matériel ou d'une entité humaine. Cette interprétation du terme agent est largement acceptée dans le contexte des systèmes logiciels. Cependant, cela laisse beaucoup de liberté quant à la classification des agents. Les dimensions communes selon lesquelles une telle classification peut être effectuée sont l'autonomie et l'intelligence. Autrement dit, on peut examiner si un agent est autonome dans son activité et si les traitements et actions qu'il effectue présentent une intelligence.

Avec de telles dimensions à l'esprit, un agent dans sa forme la plus basique n'est ni autonome, ni intelligent et par conséquent, il peut être assimilé à un composant logiciel. Exemple, il peut exécuter des tâches prédéfinies telles que la collecte et la transmission de données, comme dans le cas des agents SNMP (Simple Network Management Protocol) [Harrington 2002] (qui sont utilisés pour la gestion du réseau). Le fait que les tâches soient prédéfinies laisse peu de liberté d'action ; donc, l'autonomie de l'agent est plutôt limitée. D'autre part le fait que l'agent recueille simplement et transmet les données ne laisse aucune place à une manipulation intelligente ; de ce fait, l'agent n'a pas besoin d'intelligence.

Suivant ces dimensions, des agents plus sophistiqués peuvent présenter une autonomie ou de l'intelligence, ou les deux à la fois. Par exemple, les agents BDI (Croyance, Désir, Intention) [Rao 1995] sont des agents équipés de couches logicielles spécialement conçues pour le raisonnement intelligent et l'action. Ces derniers entretiennent et manipulent des plans et les données pertinentes aux plans, ensuite ils exécutent les plans adéquats vis-à-vis des changements de l'environnement pour atteindre leurs objectifs. Un tel comportement intelligent est basé sur des concepts tels que la croyance, le désir, l'intention et le but, qui sont tous mis en œuvre comme des artefacts logiciels au sein des agents. Ainsi, les agents BDI présentent à la fois intelligence et autonomie.

Une autre dimension importante de l'agent est la dimension sociale. La sociabilité désigne la capacité d'un agent à s'engager dans une interaction et la collaboration avec d'autres agents et entités non-agents. Par exemple, un agent peut avoir besoin d'exécuter une tâche qui ne peut être effectuée que d'une manière collaborative. Pour collaborer, un agent doit être capable de comprendre les agents qui l'entourent. Il peut également avoir besoin de négocier, de coordonner et de partager des ressources. Dans certains cas, les agents peuvent avoir besoin de prendre part dans un système plus grand composé d'agents multiples, appelé système multi-agent.

Pour faciliter le développement d'agents qui présentent de telles dimensions (et d'autres), il est nécessaire de spécifier les dimensions, les concepts sous-jacents et les constructions logicielles nécessaires à cet effet. Dans ce chapitre, nous nous proposons de les présenter brièvement.

2.4 Dimensions de l'agent

Il existe de nombreuses dimensions pour les agents. Pourtant, il n'y a aucun ensemble de dimensions qui soit largement accepté comme ensemble fondamental afin de définir ces derniers [Shehory 2014a]. Néanmoins, nous nous référons ici à un ensemble de base que nous estimons essentiel pour la définition et le développement d'agents logiciels. Ceux-ci incluent l'autonomie, l'intelligence, la socialité, et la mobilité, sur lesquelles nous élaborons dans ce chapitre.

L'autonomie

L'autonomie apparaît parmi les propriétés les plus importantes et les plus distinctives. L'autonomie se réfère à la capacité d'un agent à effectuer des calculs et des actions non supervisés et à poursuivre ses objectifs sans être explicitement programmé ou instruit pour le faire. L'autonomie se réfère en outre à l'encapsulation de données et de fonctionnalités au sein de l'agent. Cet aspect de l'autonomie est toutefois aussi présent dans les objets tels que définis dans les paradigmes orientés-objet et n'est donc pas unique aux agents. Un agent autonome est supposé avoir plein contrôle sur son état interne et ses comportements. Pour permettre une telle autonomie, un agent doit être équipé de composants qui prennent en charge l'autonomie. Une composante importante de l'autonomie est la présence d'un module d'état interne. Un tel module (d'état interne) habituellement détient et maintient l'état de l'agent dans son environnement tel qu'il est perçu et interprété par l'agent lui-même. Par exemple, un agent peut croire que son emplacement physique se trouve à une certaine coordonnée (x, y) dans un plan. Indépendamment de ce qui est sa vraie position, son état interne devrait contenir cette information et la mettre à jour. Un module d'état interne de ce type facilite l'autonomie car il permet à l'agent d'agir sur son état sans avoir besoin d'une supervision externe. Avoir un état interne au sein de l'agent est très important pour la mise en œuvre des capacités d'intelligence artificielle, comme nous le verrons plus loin dans ce chapitre. De plus, les agents montrent de l'autonomie en mettant en œuvre des comportements. Un comportement est généralement une activité qui comprend plusieurs actions élémentaires. Il est communément admis qu'un comportement est initié et contrôlé par l'agent lui-même, sans intervention

externe. Certains comportements peuvent être itératifs ou continus, tandis que d'autres ne sont exercés qu'une seule fois. Quoi qu'il en soit, les comportements permettent à un agent de poursuivre ses objectifs d'une manière autonome.

Dans de nombreux cas, l'autonomie est également associée à la proactivité. Un agent peut réagir soit en réponse à un évènement interne mais aussi externe dans le cas où ce dernier est réactif. Cependant, il peut également initier des actions, par exemple, en prévision des états et des événements de l'environnement et des autres agents avec lesquels il coexiste. Dans ce dernier cas, il est proactif. La réactivité et la proactivité peuvent coexister dans le même agent. Les deux peuvent être présentes dans un agent autonome. Pourtant, la proactivité est plus communément associée à l'autonomie, car elle montre le comportement de l'agent, qui est considéré comme indépendant des déclencheurs externes. En effet, la proactivité suggère un niveau plus élevé d'autonomie de l'agent.

Il existe aussi d'autres facteurs et indicateurs de l'autonomie des agents ; toutefois, on peut construire un agent autonome basé sur les propriétés mentionnées ci-dessus. Pour ce faire, les propriétés architecturales de l'agent doivent inclure un état interne, des comportements et des moyens de proactivité. En général, les architectures d'agents comprennent en effet de telles constructions.

Intelligence

Les agents logiciels n'ont pas besoin d'être intelligents ; cependant, l'intelligence et la notion d'agent sont fréquemment associées l'une à l'autre. En effet, la notion d'agents intelligents est utilisée non seulement dans les systèmes à base d'agents, mais aussi dans l'intelligence artificielle de manière générale. La raison derrière cette association entre intelligence et agent provient probablement du fait qu'un agent est commis à agir pour le compte d'autre agents. Dans de nombreux cas, une telle action nécessite de l'intelligence. A cet effet, les agents intelligents sont généralement, ceux qui raisonnent pour servir les autres et agissent en conséquence.

L'intelligence n'a jamais été simple à définir et la tâche ne devient pas plus facile en passant à l'intelligence artificielle. Ce chapitre n'a pas l'intention de définir ces notions ; or, il vise à présenter certains des principaux éléments associés à l'incorporation du raisonnement et du comportement intelligent dans les agents logiciels. De tels éléments comportent à leur tour des exigences architecturales et techniques qui sont d'intérêt dans cette thèse.

L'intelligence des agents peut exiger des capacités tels que l'apprentissage, le raisonnement, la planification et la prise de décision. Ces derniers doivent à leur tour permettre à un agent de prendre de meilleures décisions et de

se comporter de façon rationnelle. Ces capacités doivent également lui permettre d'être orienté vers ses objectifs. Autrement dit, un agent peut avoir des finalités abstraites pour la réalisation de ces dernières, il construit des plans, raisonne pour choisir le plus adéquat et l'exécute en conséquence. Les résultats de ses actions peuvent servir à l'agent de feed-back pour l'apprentissage, afin d'améliorer le raisonnement et les actions futures dans le but d'atteindre ses objectifs.

Pour faciliter ces capacités, un agent peut avoir besoin de certains composants logiciels spécialisés. Plusieurs approches architecturales ont été conçues pour supporter l'intelligence des agents. L'approche BDI, mentionnée plus haut, comprend une composante de croyances qui contient des faits considérés comme vrais par l'agent. Il comprend également des plans que l'agent construit et peut manipuler (appelés désirs dans l'approche BDI). Pour ce faire, la segmentation des plans et la planification sont nécessaires. De plus, on exige un mécanisme de sélection des plans. Il s'agit généralement d'un mécanisme de filtrage, de raffinement et éventuellement de prise de décision, qui permet à l'agent de choisir un seul plan d'action (ce que l'on appelle une intention). Le BDI n'est qu'un exemple d'une flore d'agents intelligents orienté but.

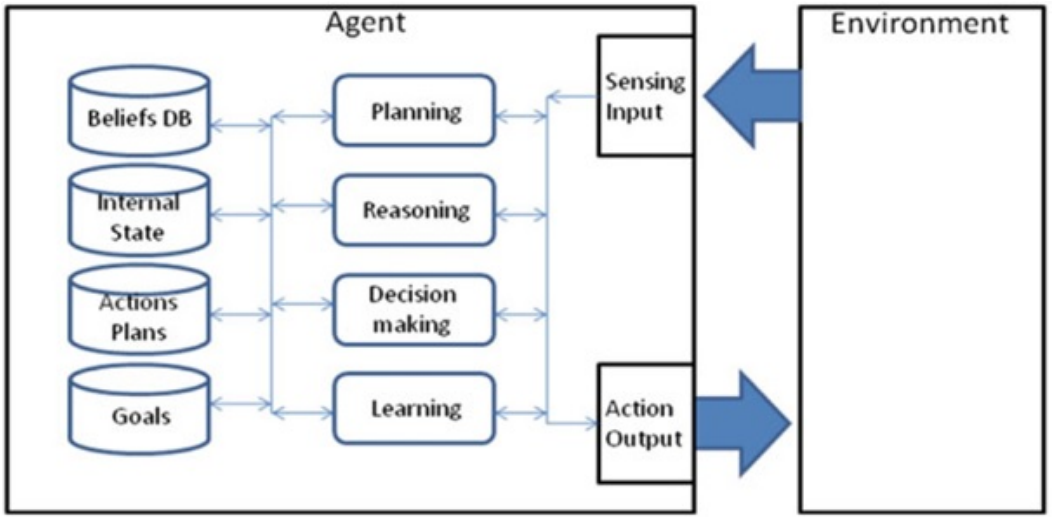


FIGURE 2.1 – Architecture d'un agent BDI [Shehory 2014a]

D'un point de vue architectural, les agents orienté but ont une architecture similaire à celle de la figure. 2.1. L'agent interagit avec l'environnement en recevant une entrée à travers un capteur (ou autre) et agit sur l'environnement par l'intermédiaire d'actionneurs. Les entrées sont stockées dans une mémoire interne, en l'occurrence, une base de données de croyances (DB), qui comprend

des faits concernant l'environnement.

L'agent peut détenir un ensemble d'actions ou de fragments à partir desquels il peut élaborer des plans. Il implémente également une composante de raisonnement qui est utilisée pour prendre des décisions concernant les meilleurs plans et actions et mettre à jour les faits dans la base de croyances. Le raisonnement peut reposer soit sur une base de règles, un système d'inférence, un arbre de décision et bien d'autres alternatives. Un agent orienté objectif implémente un module d'état interne (voir aussi l'autonomie ci-dessus). Il s'agit d'une banque de données, qui peut faire partie de la base des faits (ou croyances). Les objectifs sont habituellement stockés et servent à la planification et à la prise de décision. Dans certains agents intelligents, un élément d'apprentissage peut également être présent.

Notons que les détails spécifiques de chaque composant et leur interaction ont besoin de raffinement architectural, ce qui peut avoir un impact significatif sur la fonctionnalité et l'efficacité de l'agent. Des détails d'architectures internes d'agents sont largement présents dans l'état de l'art, y compris des manuels tels que [Wooldridge 2013, Wooldridge 1998]. Notons également que les agents intelligents et orienté buts résident généralement dans un environnement multi-agent, où ils doivent interagir intelligemment pour atteindre leurs objectifs. Dans la figure. 2.1, nous n'avons pas inclus les constructions logicielles qui soutiennent explicitement cette interaction.

Sociabilité

Dans de nombreux cas, un agent est situé dans un environnement multi-agent. Pour atteindre ses objectifs, l'agent peut avoir besoin d'interagir, de coordonner, de collaborer ou d'être en concurrence avec d'autres agents. Par exemple, un agent peut avoir besoin de former une équipe pour effectuer une tâche de sauvetage dans une mission de récupération après un sinistre. Pour ce faire, l'architecture de l'agent doit inclure des composants favorisant la sociabilité. Au niveau infrastructure, la communication est nécessaire. Un composant de communication doit permettre l'envoi, la réception et le traitement des messages. Il doit également prendre en charge des langages de communication spécifiques, des formats de message et des protocoles (par exemple, ACL FIPA [Specification 2017]). L'analyse et la compréhension du message est aussi un besoin fondamental pour toute communication significative. Dans ce qui suit nous fournissons plus de détails sur la communication des agents.

En plus de la communication, une interaction significative nécessite généralement que l'agent ait un modèle des autres agents ou qu'il garde au moins certaines données les concernant, à savoir leurs informations de contact, leurs capacités ou éventuellement des journaux d'interaction passée avec eux.

Pour ce faire, l'architecture de l'agent doit inclure des modèles de lui-même et des autres agents. La composante de raisonnement doit comprendre des moyens pour raisonner sur les agents et interagir avec eux. Des extensions similaires sont également nécessaires pour les modules de planification, de prise de décision, et d'apprentissage. Exemple, les plans partagés [Grosz 1996], la prise de décision conjointe, et l'adaptation multi-agent et l'apprentissage [Panait 2005].

D'autres capacités peuvent être nécessaires pour faciliter les comportements sociaux. Par exemple, les mécanismes de collaboration, de formation d'équipes et de coalitions [Shehory 1998, Tambe 1997], ainsi que les comportements stratégiques dans des scénarios concurrentiels ; tous permettent la sociabilité. Un exemple du besoin d'un tel mécanisme est le cas où un agent a besoin d'acheter certains biens pour le compte d'un utilisateur humain alors qu'un rabais est disponible, qui est basé sur le nombre d'articles achetés. Dans un tel cas, il peut être avantageux pour l'agent de former un groupe d'acheteurs pour obtenir un rabais. À cette fin, un mécanisme approprié et des stratégies à adopter, sont nécessaires. Les constructions logicielles qui favorisent la compréhension et la participation à de tels mécanismes sont nécessaires dans le cadre d'un agent socialement capable.

Les agents qui interagissent dans un environnement concurrentiel peuvent avoir besoin de négocier et éventuellement utiliser des stratégies et de l'argumentation dans le cadre de cette négociation. Par conséquent, ils doivent être équipés de modules de négociation et d'argumentation pour faciliter ces capacités. Il existe de nombreuses façons de mettre en œuvre de tels modules. De tels détails sont hors de la portée de ce chapitre. Une référence importante sur la négociation et l'interaction des agents est présenté dans le livre « Rules of Encounter » [Rosenschein 1994].

Mobilité

Certains agents et systèmes multi-agents présentent une mobilité. Autrement dit, les agents peuvent être en mesure de modifier leur emplacement logique ou physique. L'une des manières de manifester cette mobilité est que l'agent se déplace de son environnement d'exécution actuel (l'hôte sur lequel il s'exécute) vers un autre environnement d'exécution. Une autre incarnation de la mobilité se trouve dans les cas où les agents résident sur des appareils mobiles tels que les smartphones. Dans de tels cas, même si l'agent ne quitte jamais son hôte, le périphérique d'hébergement peut lui-même se déplacer, ce qui change l'emplacement et les conditions environnementales. En conséquence, l'agent qui s'exécute sur l'appareil se déplacé et ainsi modifie ses conditions environnementales. Les deux types de mobilité exigent que les

agents soient conçus pour la mobilité et soient équipés de modules logiciels appropriés pour la gérer.

La mobilité dans laquelle les agents passent d'un environnement d'exécution à un autre nécessite généralement plus d'infrastructure que les autres types de mobilité. Nous en discutons (brièvement) ici. Un agent qui se déplace vers un autre hôte (physique ou logique) doit enregistrer son état avant de se déplacer, générer une copie de lui-même sur l'hôte cible, arrêter son exécution sur l'hôte source et reprendre l'exécution sur l'hôte cible. Un ensemble spécial de comportements de mobilité devrait donc être incorporé dans de tels agents. Pourtant, il ne suffit pas que l'agent soit équipé de constructions logicielles nécessaires pour la mobilité. Il est indispensable que les hôtes sur lesquels des agents mobiles peuvent s'exécuter disposent d'une infrastructure de support pour la mobilité telle qu'une station d'accueil. Cette dernière, représente un environnement d'exécution qui fournit à l'agent mobile les ressources dont il a besoin pour son exécution, mais aussi protège l'hôte cible contre la surexploitation et les attaques externes.

Autres dimensions

Il y a beaucoup d'autres propriétés de l'agent qui affectent leur ingénierie. Comme nous l'avons déjà dit, nous n'avons pas l'intention de toutes les couvrir. Une propriété importante de certains agents est leur capacité à s'adapter aux changements au niveau de leurs objectifs et de leurs conditions environnementales. L'adaptation est en effet considérée comme une dimension de l'agent. Toutefois, dans de nombreux cas, elle ne nécessite pas de module d'adaptation spéciale. Par exemple, l'adaptation peut être traitée par des capacités d'apprentissage, de planification et de raisonnement.

Un autre aspect de l'agent est son interaction avec les utilisateurs humains. Étant donné que dans de nombreux cas, les agents effectuent des tâches pour le compte d'utilisateurs humains : la qualité de l'interaction avec l'humain est donc d'une grande importance. Cette interaction doit être supportée par des composantes d'interaction homme-machine. L'interaction homme-machine est un vaste domaine de recherche et de pratique qui s'étend au-delà des agents, des systèmes multi-agents et de l'IA.

Les agents peuvent avoir un corps physique, comme dans le cas des robots. Les propriétés physiques des agents introduisent un ensemble d'exigences qui ne sont pas présentes dans les agents logiciels. Ceux-ci sont bien pris en compte dans le domaine de la robotique. Encore une fois, nous renvoyons le lecteur à ce domaine pour plus de détails. Certains autres aspects telles que les émotions [Dastani 2014], la conscience de soi, la sécurité, la gestion des ressources ainsi que d'autre ensembles de composants logiciels peuvent être incorporés dans

les agents.
Néanmoins, il est important de noter que les dimensions introduites ici ne soient pas toutes ou partiellement regroupées dans un même agent.

2.5 Architectures d'agents

L'agent d'un point de vue externe est vu comme une entité autonome, réactive, pro-active et sociale qui interagit avec d'autres agents (SMA) et avec son environnement. D'un point de vue interne, c'est une entité logicielle ou matérielle capable de prendre des décisions et de gérer ses relations avec d'autres agents. Elle est également capable de contrôler ses actions et ses perceptions. Les architectures d'agent traitent la conception et la création de systèmes informatiques qui satisfont aux propriétés de l'agent (proposées par les théoriciens de l'agent) [Wooldridge 2000a]. "Une architecture d'agent est essentiellement une carte des composants internes d'un agent - constitué de structures de données, d'opérations et de flux de contrôle entre ces structures» [Wooldridge 1995]. On identifie les classes suivantes d'architectures d'agent :

- Architectures délibératives (ou symboliques)
- Architectures réactives (ou comportementales ou situées)
- Architectures hybrides

L'architecture délibérative la plus populaire et la plus utilisé est l'architecture BDI (Belief Desire Intention) [Georgeff 1998]. Selon le modèle BDI, un agent maintient une base de connaissances sur son environnement appelé croyances. L'ensemble des désirs représente les états possibles du monde que l'agent veut atteindre. Les intentions représentent un sous-ensemble de désirs non conflictuels. Les architectures d'agents varient des architectures réactives telles que l'architecture de Brook (subsomption) [Brooks 1991] aux architectures hybrides tel que INTERRAP [Müller 2011].
Les systèmes à base d'agents se concentrent sur les composants interactifs et dynamiques. Chacun de ces composants à son propre thread de contrôle et est engagé dans des protocoles de coordination complexes. L'accent est mis sur l'interaction entre composants plus que sur la structure interne de ces derniers [Jennings 2000].

2.6 Modélisation d'agents

"L'ingénierie logicielle orienté agents" (AOSE) consiste à appliquer les principes de l'ingénierie logicielle et de l'intelligence artificielle pour l'analyse, la conception et la mise en œuvre des systèmes logiciels. L'ingénierie logicielle

orientée agents est interprétée de deux manières différentes. La première vise à soutenir le développement d'un système à base d'agents, tandis que la seconde est plus révolutionnaire et vise à soutenir le développement de systèmes complexes où la notion d'agent est utilisée. S'il apparaît que la communauté AOSE a fait des progrès considérables dans la première vision, la seconde n'a pas beaucoup progressé. Ceci est souligné par Ricci et Santi [Ricci 2011].
Au cours des deux dernières décennies, le domaine AOSE a évolué pour aborder de nombreux sujets. Le domaine comprend les thèmes de haut niveau : méthodologies, techniques de modélisation, implémentations de framework, langages de programmation d'agents et communication d'agents.

2.7 Applications

Il y a une exploration continue des types d'applications pour lesquelles les agents sont une approche de développement adéquate et précieuse. Ci-dessous, nous listons les champs pour lesquels le paradigme orienté agent a été étudié ou examiné, ou est considéré comme prometteur. Nous avons divisé la liste en sous-listes orientées sur les agents, se réfèrant à la manière dont la technologie de l'agent est incorporée avec d'autres technologies mais aussi une sous-liste qui se réfère aux domaines d'application dans lesquels des agents ont été trouvés utiles. Les domaines orienté agent sont les suivants [Müller 2014] :

- Les grilles de calcul orienté agent
- Agents et services (alignement des agents avec le développement de logiciels orientés services)
- Agents pour systèmes auto-adaptatifs
- L'Intégration de logiciels orientés agents dans les processus d'affaires existants et la réingénierie des processus d'affaires
- Simulation multi-agent

Les domaines d'applications sont les suivants :

- Les systèmes auto-organisés
- L'ingénierie sociale
- Le commerce électronique
- Les services touristiques
- Les jeux

2.8 La Communication des agents

Un aspect important des agents est leur capacité à communiquer. La communication peut être exercée de différentes façons. Dans la littérature, la communication de l'agent a été abordée de divers points de vue telles que la performance, l'évolutivité, la fiabilité, la sécurité et la confidentialité.
Les différents niveaux d'abstraction influent également sur l'étude de la communication entre agents. En conséquence, plusieurs sujets liés à la communication de l'agent se retrouvent dans la recherche et la pratique, notamment :

- Les langues de communication de l'agent (ACL), y compris la sémantique et la pragmatique
- L'argumentation
- Les engagements en communication
- Les infrastructures de communication
- La coordination et la coopération
- Les jeux de dialogue
- La communication homme-agent
- L'intégration des protocoles d'interaction au sein des agents
- L'interopérabilité
- Les conversations multi-parties
- L'application du traitement du langage naturel à la communication
- La négociation
- Les ontologies et communication
- La réutilisation dans la communication.

2.9 Qu'est-ce qu'un système multi-agent ?

Les Systèmes Multi-Agents (SMA) ont résulté du fait de la nécessité de construire des systèmes distribués complexes et coopératifs dans le domaine de l'intelligence artificielle [Ferber 1995]. Un SMA contient un ou des ensembles d'agents qui interagissent les uns avec les autres dans un environnement commun et formant éventuellement une organisation. Un exemple typique d'un système multi-agent est présenté dans la figure 2.2 où nous pouvons voir que les agents perçoivent et agissent sur leur environnement. Une autre définition intéressante des SMA est comme suit :
Selon [Cossentino 2005] « Un SMA est un système logiciel composé de multiples points de contrôle indépendants et encapsulés (c'est-à-dire les agents) interagissant dans un contexte applicatif spécifique ».

Ferber a défini un système Multi Agent comme une population d'agents autonomes qui interagissent. Il se concentre sur les deux aspects de l'organisation et l'interaction [Ferber 1995]. Demazeau affirme qu'une approche SMA consiste en quatre modules essentiels : Agent, Environnement, Interaction, Organisation (AEIO) [Demazeau 1995, Hübner 2002a]. Représentant :

- **A** définit un agent avec ses capacités de raisonnement internes comme éléments de base du SMA.
- **E** représente l'environnement où l'agent interagit.
- **I** désigne l'interaction (elle peut être entre les agents et entre les agents et leur environnement.)
- **O** désigne l'organisation qui fournit une structure au SMA. Elle est basée soit sur les fonctionnalités, le comportement ou l'interaction entre agents.

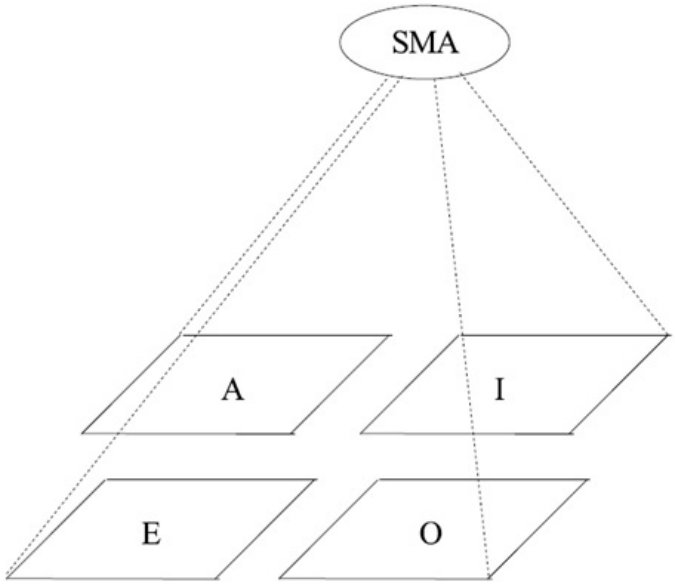


FIGURE 2.2 – Système multi-agent [Wooldridge 2009]

Dans les OMAS (Open Multi-Agent System), la notion de rôle est également importante, car elle permet de distinguer les participants d'une interaction et de définir les comportements et les capacités des agents. Un agent avec les modules A et E est appelé agent autonome, avec les modules A, E, I : un agent communicatif et avec les quatre modules : un agent social. Il existe un manque de normalisation dans la classification des SMA, qui peut changer en fonction du contexte. Toutefois, nous pouvons distinguer les deux catégories principales des SMA : individuels et collectifs. Il semble difficile de

définir les frontières entre ces types de SMA, chaque catégorie peut contenir des agents cognitifs et réactifs, ce qui apporte une dimension fonctionnelle ou décisionnelle aux SMA.
Les systèmes multi-agents organisationnels sont collectifs de ce fait, un problème est divisé et réparti entre ces agents. D'autre part, il existe des types de SMA centrés agent. Ces types d'agents individuels peuvent prendre leurs décisions de manière indépendante lors de la résolution de certains problèmes. Selon Ferber, ces deux types de SMA peuvent être visualisés à partir de deux vues : interne et externe. Il en résulte quatre cas possibles (voir le tableau 2.2) : la vue individuelle interne considère les états mentaux et l'architecture d'un agent. La vue individuelle externe définit le comportement de l'agent. La vue interne des SMA collectifs définit leur ontologie, leurs connaissances communes et leurs normes sociales, alors que la vue externe représente l'organisation de ces derniers.

	Interne	**Externe**
Individuel	États mentaux et architecture des agents	Comportement de l'agent
Collectif	Ontologie, connaissances communes et Les normes sociales	Organisation et institutions

TABLE 2.2 – Vue intérieure et extérieure des SMA [Aboud 2012]

2.10 Modélisation des systèmes multi-agents

Nous sommes conscients qu'un modèle conceptuel simple est une condition essentielle pour le succès des agents en tant que paradigme de développement[Lacomme 2011]. Dans cette section, nous faisons un état de l'art orienté vers la représentation des SMA et de leur dynamique. Nous organisons et analysons les modèles existants des systèmes multi-agents provenant de milieux différents. En effet, nous nous intéressons aux méthodes de développement pour différentes raisons :

- L'identification des concepts clés utilisés lors de la modélisation de SMA.
- L'analyse de la conception de ces approches et leurs orientations : SMA ouverts, émergents, auto-adaptatifs ;
- L'analyse des prérequis architecturaux imposés au niveau de chaque méthode
- L'évaluation des SMA créés, si possible au niveau de chaque approche.

A partir de ces informations nous pouvons identifier les besoins et les problèmes existants dans l'état de l'art du paradigme SMA, ce qui nous permet

d'avoir une vue des problèmes soulevés lors de la conception des SMA, et de positionner notre modèle sur l'axe de conception.

2.10.1 Méthodes de conception des SMA

AGR

AGR (Agent, Groupe, Rôle) [Ferber 2003] est l'évolution du modèle AALAADIN [Ferber 1998]. Dans AGR, le groupe et le rôle sont les concepts de modélisation primitifs. Un agent est une entité active et communicante, jouant un ou plusieurs rôles, au sein d'un ou de plusieurs groupes. Aucune contrainte n'est imposée sur l'architecture interne et les capacités mentales d'un agent. Selon les auteurs, « un agent peut être aussi réactif qu'une fourmi ou autant intelligent que l'homme ». Un groupe est un ensemble d'agents qui partagent certaines caractéristiques communes. Un groupe est le contexte d'un ensemble d'activités et il est utilisé pour partitionner les organisations. Un agent peut participer en même temps dans un ou plusieurs groupes. Ces agents peuvent communiquer si et seulement s'ils appartiennent au même groupe. Un rôle est la représentation abstraite d'une position fonctionnelle (comportement attendu) d'un agent dans un groupe. Les rôles sont locaux aux groupes, et un agent doit demander un rôle. Le principal atout du modèle AGR est sa simplicité. Cependant, ce modèle souffre d'un environnement de conception.

OperA

OperA (Organisations per agent) [Dignum 2004a] est un cadre pour la spécification des systèmes multi-agents, composé de trois modèles interdépendants :

- Le modèle organisationnel (OM) : décrit la structure organisationnelle de la société au moyen de rôles et d'interactions.
- Le modèle social (SM) : spécifie comment les agents individuels choisissent leurs rôles ;
- Le modèle d'interaction (IM) : décrit l'interaction possible entre les agents.

Une caractéristique principale d'OperA est qu'il propose des constructions pour représenter à la fois les préoccupations des organisations d'agents (dans l'OM) et les intérêts individuels des agents (dans le SM et IM).
Le modèle organisationnel d'OperA, est utilisé pour spécifier une organisation d'agents selon quatre structures : sociale, interactionnelle, normative et communicative.

Dans le cadre de la structure sociale : les objectifs, les groupes et les dépendances entre les rôles sont définis. Les rôles identifient les activités et les services nécessaires pour atteindre les objectifs sociaux et permettent d'abstraire les individus qui les réalisent. Les rôles sont décrits par des objectifs (ce qu'un acteur devrait atteindre), des normes (comment un acteur doit se comporter), des droits (capacités que les acteurs reçoivent lors de l'adoption du rôle) et le type de rôle (rôle institutionnel ou externe). Les groupes fournissent le moyen de se référer collectivement à un ensemble de rôles. Les groupes sont utilisés pour spécifier les normes adoptées pour tous les rôles du groupe. Les dépendances entre les rôles décrivent leurs liaisons lors de la réalisation d'un objectif. Il existe trois types de dépendance de rôle, liés aux mécanismes : hiérarchiques (délégation), de marché (enchères) et de coordination générale (demande).

La structure d'interaction, définit comment l'activité principale d'une organisation d'agent devrait se dérouler. Cette définition est basée sur des scènes, scripts, transitions de scène et de relations d'évolution des rôles. Les scènes sont des représentations d'interactions spécifiques qui impliquent des acteurs. Un script de scène est décrit par ses acteurs (rôles ou groupes), les normes de scène (comportement attendu des acteurs dans une scène) et un modèle d'interaction. Les transitions de scènes sont utilisées pour coordonner et définir l'ordre et la synchronisation des scènes.

Les relations d'évolution des rôles précisent les contraintes à vérifier par les agents lorsque ces derniers se déplacent d'une scène à l'autre tout en respectant les transitions définies.
La structure normative de l'OM d'OperA regroupe toutes les normes qui sont définies lors de la spécification des rôles, des groupes et des scripts de scène. Enfin, la structure communicative décrit l'ensemble des performatives et des concepts utilisés dans la structure d'interaction. Bien que ce modèle offre une séparation plus claire des préoccupations (organisationnelles, sociales et d'interactions), il est encore trop complexe pour être utilisé par les développeurs.

ISLANDER

ISLANDER [Esteva 2002] est l'un des premiers cadres de conception pour les organisations et les institutions multi-agents. Il spécifie un SMA en termes de règles et de normes organisationnelles. Pour interpréter et exécuter ces spécifications, une plate-forme d'exécution, appelée AMELI, a été développée par [Esteva 2004]. Cette plate-forme met en œuvre une infrastructure qui, d'une part, facilite la participation de l'agent dans l'environnement institutionnel, et soutient l'aspect de communication et, d'autre part, impose l'application des règles et des normes institutionnelles spécifiées. L'aspect clé d'ISLANDER / AMELI est que les normes ne peuvent jamais être violées par les agents. En

d'autres termes, les systèmes programmés via ISLANDER / AMELI utilisent uniquement la régimentation afin de garantir que les normes sont effectivement suivies. Les normes dans [Esteva 2004, García-Camino 2005, da Silva 2008] sont liées à des actions que les agents devraient ou ne devraient pas effectuer. Dans ces approches, les questions liées à l'expression de normes de plus haut niveau sont ignorées. Ces normes de haut niveau fournissent une description déclarative d'un état du système plutôt que de préciser comment elles devraient l'établir.

OMNI

OMNI [Dignum 2004b]. Il s'agit d'une extension récente de OperA, qui combine la méthodologie OperA avec le cadre HARMONIA [Vázquez-Salceda 2003]. Ce cadre traduit des normes d'un niveau abstrait (dans lequel les statuts et valeurs d'organisation sont définis) à un niveau procédural (où les normes sociales sont implémentées). Dans OMNI, les points forts des deux approches sont unifiées, mais il manque encore des outils appropriés pour l'analyse et la conception. L'outil AMELI/ISLANDER [Esteva 2004] peut être utilisé pour la description des normes, des scènes et des transitions, par contre le modèle organisationnel OperA doit être complété à la main. Ce modèle offre un équilibre entre les besoins de l'organisation, et l'autonomie de l'agent. Il existe trois dimensions essentielles pour ce modèle : organisationnelle, normative et ontologique [Dignum 2004b]. La figure 2.3 représente les différents niveaux et dimensions d'OMNI.

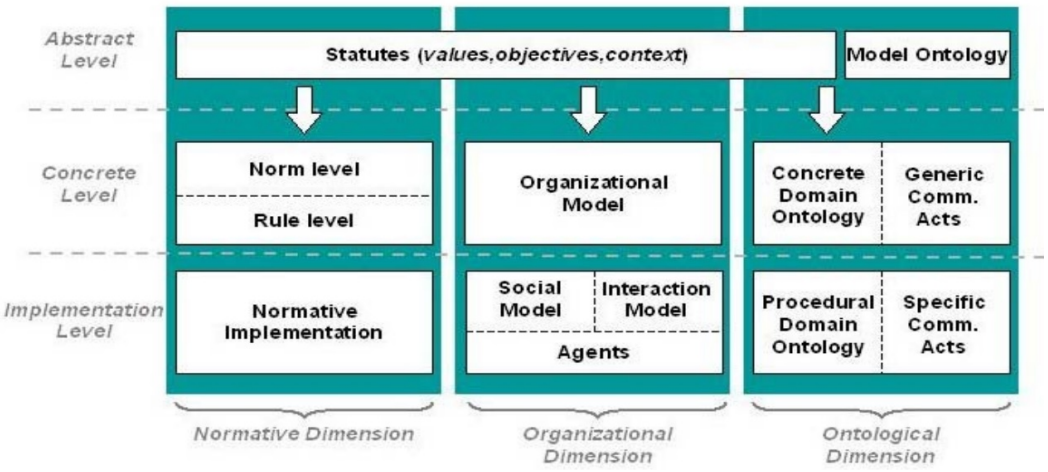


FIGURE 2.3 – Niveaux et dimensions d'OMNI [Dignum 2004b]

Modèle d'organisation pour les systèmes multi-agent (MOISE / MOISE +)

MOISE [Hannoun 2000] est un modèle organisationnel qui combine des architectures d'agents et d'organisations. Le modèle MOISE se compose de deux niveaux importants : (1) structure organisationnelle (OS) et (2) entité organisationnelle (OE). La OS est un ensemble de rôles et de groupes qui participent à la définition de la structure du système, indépendamment des agents qu'elle inclut. La population des agents qui obéit aux contraintes de OS est l'OE. Ces deux niveaux facilitent la représentation de la conception. Néanmoins, si le contrôle dynamique de l'organisation ou la sémantique exacte d'une instance ou groupe de rôles n'est pas défini, nous pouvons voir l'OE comme une instance d'une OS pour un ensemble d'agents. La figure 2.4 présente l'OS et l'OE, on trouve dans l'OS les types de groupes avec leurs rôles liés aux schémas. Un schéma est un arbre de buts et de missions à atteindre. Au niveau OE, nous trouvons les instances de groupe et de schéma, avec les missions et les rôles liés aux agents.

MOISE + [Hübner 2002b] est une extension du modèle MOISE avec de nombreux apports, tels que le processus de réorganisation et l'apparence de l'héritage entre rôles. Les concepts principaux dans MOISE / MOISE + sont : les rôles et les lignes organisationnels, qui représentent la relation entre les rôles et les groupes. Ici, nous détaillons certains de ces concepts. L'organisation agit comme un système de règles qui contraint l'activité des agents(leurs actions individuelles possibles). Ces règles sont exprimées à l'aide de la notion de rôle, qui définit les comportements attendus d'un agent au sein de sa structure sociale.

Un groupe est défini comme une agrégation de rôles, c'est-à-dire un ensemble de règles cohérentes. Cette cohérence est défini par des liens spécifiés entre les rôles, ce qui exprime les possibles communications, connaissances attendues ou relations d'autorité entre les rôles.

L'agent est spécifié comme responsable d'une partie de l'ensemble des tâches de l'application, et possède ses propres ressources pour réaliser certaines actions.

Le rôle définit le comportement et les services fournis ou requis d'un agent. L'originalité du modèle MOISE pour ce concept est la capacité de considérer le rôle comme un ensemble de missions, que l'agent doit suivre.

La mission est un sous-concept lié au rôle, un agent qui joue certains rôles doit archiver certaines missions. Ces missions répondent aux contraintes et comportements pour accomplir une tâche. Elles contiennent l'autorisation pour les quatre éléments de tout comportement (objectif, plan, action et ressource). Les lignes organisationnelles, structurent les échanges sociaux entre les rôles, cette ligne relationnelle a des rôles source et cible et est marquée

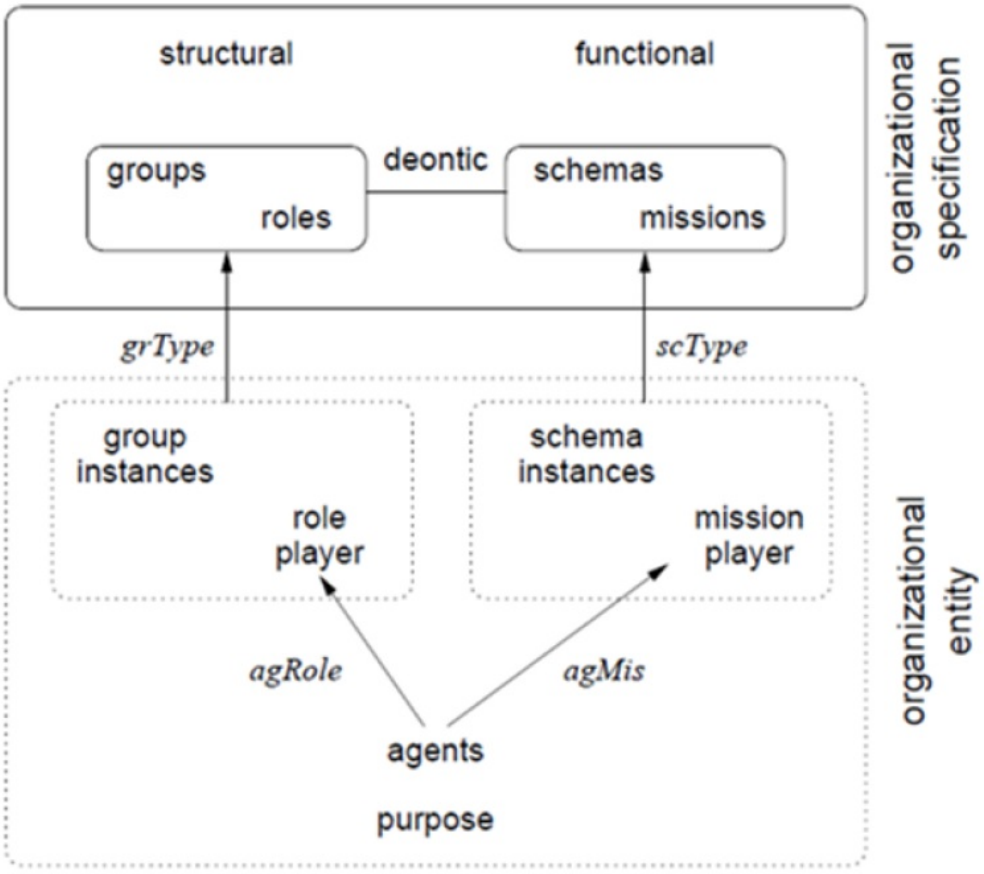


FIGURE 2.4 – La spécification du niveau organisationnel et entité [Hannoun 2000]

par N éléments (ensemble de contraintes, ensemble de missions du rôle source et cible, ce qui permet de définir le contexte d'une mission). Dans MOISE, une ligne d'organisation peut être de type : ligne de communication, ligne d'autorité et ligne de connaissance. Les lignes de communication structurent les échanges d'informations ; et définissent également les protocoles d'interaction, tandis que la ligne d'autorité définit le contrôle et la séquence d'un rôle à l'autre. La ligne d'accointance définit la vue que l'agent a sur les autres agents de l'organisation.

Dans cette approche, nous trouvons que le concept d'interaction est explicite entre les rôles de l'organisation ce qui permet de définir des protocoles. Néanmoins le concept de service est représenté implicitement par les notions de rôle, de comportement et de mission de l'agent.

2.10.2 Modèles organisationnels

Gaia

La méthodologie Gaia est apparue dans [Wooldridge 2000a] pour l'analyse et la conception de systèmes multi-agents fermés. Toutefois, elle a été étendue dans [Zambonelli 2003] afin de prendre en charge les SMA ouverts. La méthodologie Gaia permet de construire une organisation / société de SMA. Cette dernière se compose d'un ensemble de rôles assignés à des agents qui définissent les schémas d'interactions.
L'organisation désigne l'abstraction organisationnelle tels que l'environnement, ses entités et les ressources qui sont utilisées chaque à fois que les agents interagissent pour réaliser un objectif organisationnel. Les rôles et les interactions sont détaillés dans des modèles dédiés. L'organisation définit également des règles organisationnelles qui décrivent les fonctionnalités et les capacités requises par une organisation. Elle s'appuie sur deux niveaux de modélisation, qui partent de concepts abstraits correspondant à l'étape d'analyse, qui permet la spécification des modèles de rôle et d'interaction. Pour aboutir à un niveau concret correspondant à l'étape de conception, qui définit les modèles d'agent, de service et d'accointance.

- **Le modèle de rôle** décrit les différents rôles d'un système. Un rôle dans Gaia est une description abstraite d'une fonctionnalité. Il est défini par trois éléments :
  - Les responsabilités qui décrivent le rôle ainsi que ses fonctionnalités ;
  - Les autorisations ou permissions accordées à un rôle ;
  - Les protocoles qui symbolisent les relations entre les rôles. Ces derniers sont définis dans le modèle d'interaction.

- **Le modèle d'interaction** est composé d'un protocole qui définit les interactions inter-rôles. Les protocoles tiennent compte des modèles d'interaction, et sont définis par les initiateurs, les interlocuteurs, les flux entrants, les flux sortants et une description textuelle.
- **Le modèle d'agent** consiste à identifier les types d'agents et les instances d'agents dans les phases d'exécution. Le type d'agent est défini par l'ensemble des rôles qu'il peut jouer.
- **Le modèle de service** décrit les services fournis par chaque type d'agent, et l'activité associée à chaque rôle. Un service correspond à une fonction dans Gaia.
- **Le modèle d'accointance** définit simplement l'accointance qui existe entre les liens de communication des différents types d'agents, sous forme d'un graphe orienté.

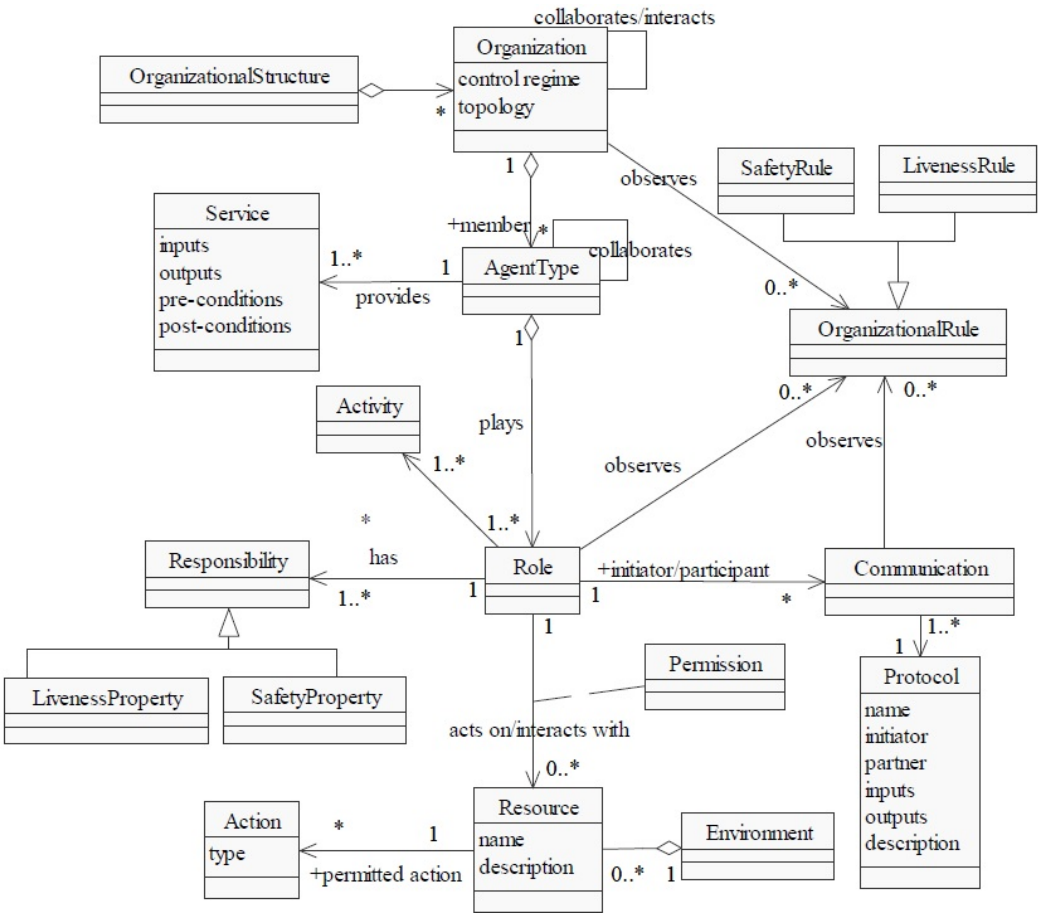


FIGURE 2.5 – Méta-model Gaia [Jennings 2001]

La figure 2.5 montre qu'une organisation dans Gaia est une agrégation

d'agents qui fait partie d'une structure organisationnelle. L'agent fournit des services et joue des rôles à travers ses interactions. Le concept de service est explicite dans cette méthodologie, il est considéré comme un bloc d'activité cohérent dans lequel un agent peut s'engager. Le concept d'interaction est également explicite, et il est défini par un protocole. Cette méthodologie n'apporte pas de détail sur la phase de mise en œuvre de la conception résultante.

2.10.2.1 INGENIAS

MESSAGE [Caire 2001] et INGENIAS [Gomez-Sanz 2002]. Identifient tous les cas d'utilisation dans le système, puis ils spécifient un modèle organisationnel, où les groupes, les membres, les flux de travail et les objectifs organisationnels sont détaillés. Ainsi, plusieurs flux de travail, et les acteurs (agents et rôles) participant à ces flux de travail, sont définis pour chaque cas d'utilisation. Les acteurs sont ensuite regroupés selon leur fonctionnalité ou selon les ressources dont ils ont besoin. Dans ces méthodes, les groupes sont identifiés, mais aucune conception organisationnelle contenant des intervenants humains n'est envisagée. Les normes sociales ne sont pas explicitement modélisées (elles sont supposées être implicitement dans la structure organisationnelle) et les dynamiques organisationnelles ne sont pas considérées (alors, comment les agents peuvent-ils rejoindre ou quitter le système ? comment peuvent-ils former dynamiquement des groupes ? etc...).

2.10.2.2 PASSI

L'acronyme de processus pour la spécification et la mise en œuvre de sociétés d'agents (PASSI) [Cossentino 2005] couvre toutes les phases du développement à partir de l'exigence au code sans oublier le test. Il étend, les concepts UML pour s'adapter aux conceptions des agents, et utilise la plate-forme FIPA pour leur implémentation. Cette méthodologie contient trois domaines (voir figure 2.6).

- **Le domaine du problème**, qui est lié à la phase d'exigence et d'analyse, où nous trouvons les informations liées au contexte, comme les scénarios, les ressources, les exigences et l'ontologie ;
- **le domaine de l'agentification** définit les concepts du modèle d'agent qui sont liés aux exigences qui, à leur tour sont liées aux concepts du domaine de la solution ;
- **le domaine de la solution** ou de la mise en œuvre de l'agent dans la plate-forme FIPA.

Le noyau du méta-modèle de cette méthodologie se situe au niveau du domaine de l'agentification. Nous pouvons y trouver les concepts d'agent, de

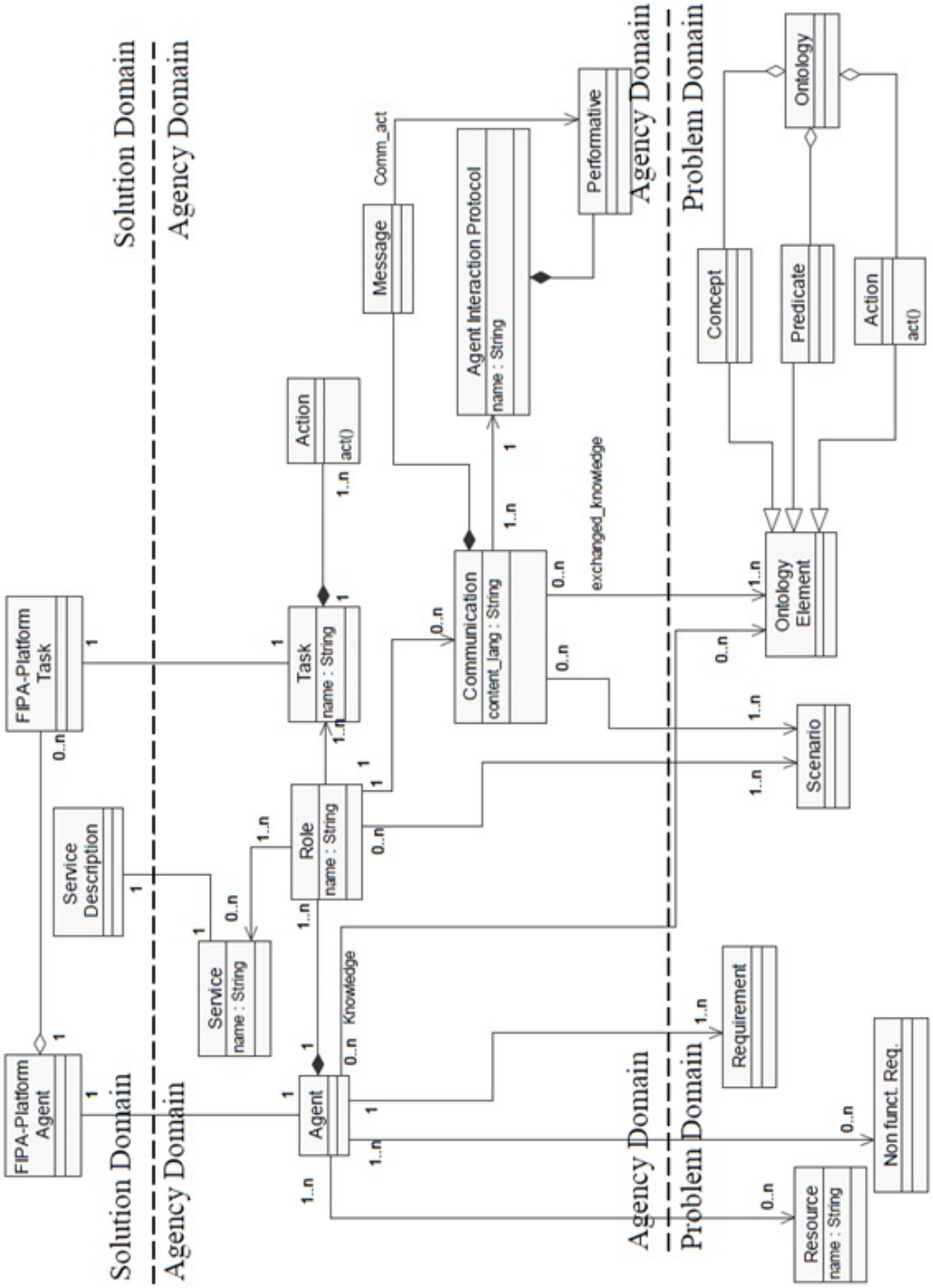


FIGURE 2.6 – Méta-model PASSI [Jennings 2001]

rôle, de service, de but, de tâche, de communication et d'interaction d'agent. Le service dans ce modèle a la même signification que dans le modèle Gaia, il est associé à chaque rôle d'agent. L'interaction est un modèle de conversation utilisé pour effectuer certaines tâches, il s'agit de modèles de dialogue basés sur l'envoi de messages.

2.10.2.3 Tropos

La méthodologie Tropos [Bresciani 2004] couvre également toutes les phases d'un processus de développement logiciel. Elle se concentre, sur les interactions des agents et leurs environnements.
L'originalité de cette méthodologie est l'utilisation du concept d'acteur comme une généralisation de l'agent [Bresciani 2004]. Un acteur peut être physique ou un agent logiciel. Un acteur peut atteindre ses objectifs en acceptant un plan et / ou en utilisant les ressources de l'environnement. Il existe une relation de dépendance entre les agents afin de satisfaire leur propre objectif ou d'accéder à une ressource (voir la figure 2.7).
Le concept de service n'existe pas explicitement dans cette méthodologie, bien qu'elle contienne des modèles sociaux qui prennent en charge les aspects intentionnels.

2.10.3 Modèles architecturaux pour SMA

Dans les approches architecturales, plusieurs contributions proposent différentes descriptions architecturales formelles et semi-formelles qui mettent l'accent sur la conception de systèmes multi-agents.
Dans [Park 2005], DMAS une approche semi-formelle centrée sur l'architecture est proposée pour développer des SMA. Elle prend en charge les phases les plus importantes du développement de logiciels, et permet de mettre en avant différentes propriétés du système en particulier celles liées à la coordination et à l'autonomie des agents. La conception est organisée en trois phases. La première phase (analyse des problèmes) : une approche axée sur les objectifs et utilisée pour l'analyse du problème, ensuite les agents sont mappés avec les objectifs du système. La deuxième et troisième phases (modélisation d'agents et conception SMA) définissent, la structure interne de chaque agent et son assemblage en utilisant UML (Unified Modeling Language) et ADL (Architecture Description Language).
En outre, deux ADLs dédiés pour la description d'architecture SMA sont proposés : SKwyRL-ADL [Faulkner 2003] et ADLMAS [Yu 2006]. Le SKwyRL-ADL est basé sur la logique de premier ordre. Cette approche identifie les bases d'un langage de description architecturale (ADL) pour spécifier des architectures de systèmes multi-agents. Un ensemble de concepts architecturaux basé

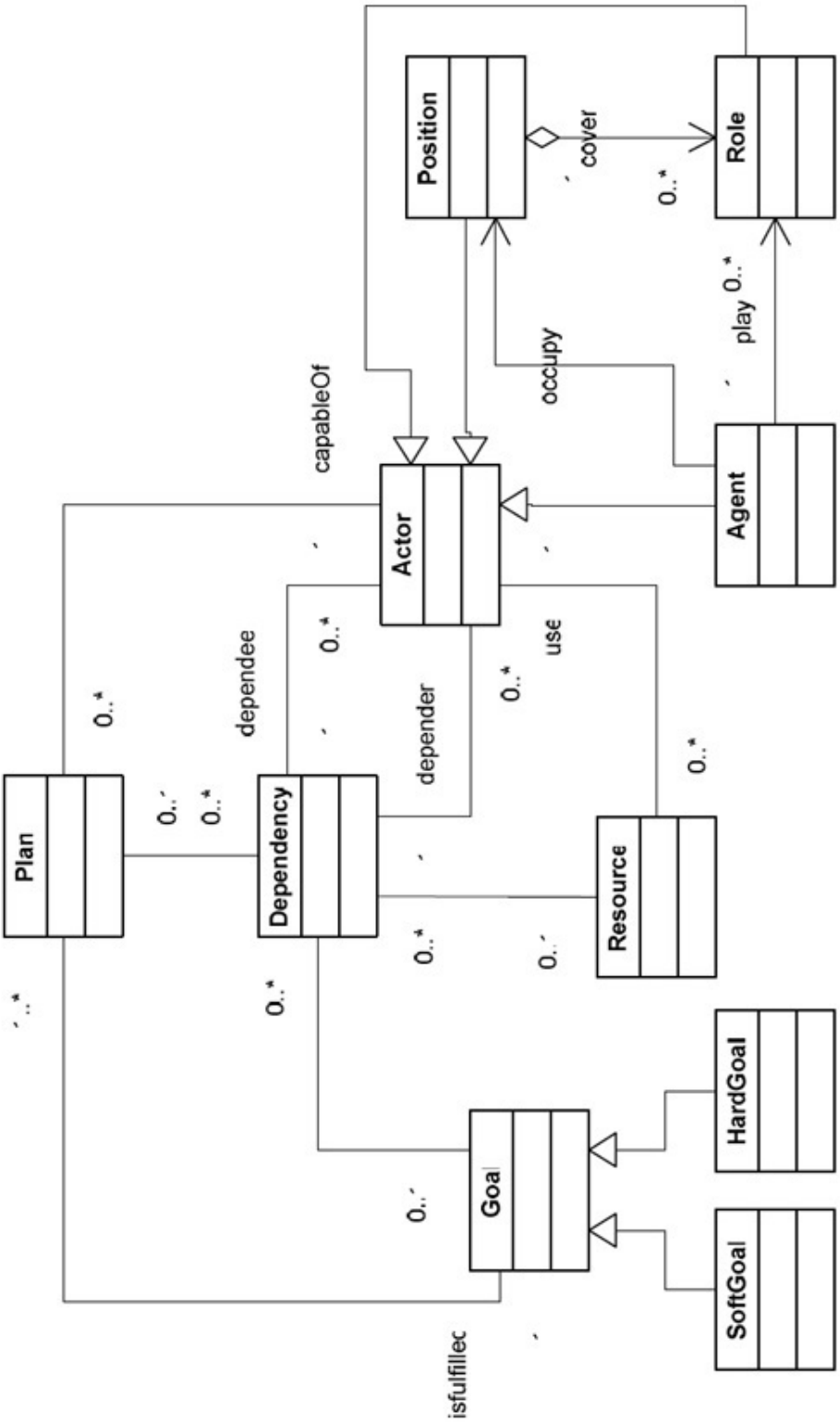


FIGURE 2.7 – Méta-modèle Tropos [Jennings 2001]

principalement sur le modèle d'agent BDI et les ADL classiques est proposé. La modélisation d'un SMA peut être traitée sur deux niveaux : interne et global. Le modèle interne capture les états et le comportement potentiel de l'agent, tandis que le modèle global est utilisé pour décrire l'interaction entre les agents qui composent l'architecture SMA. Dans ADLMAS [Yu 2006], les réseaux de Pétri orientés-objet ont été adoptés pour représenter les aspects de concurrence, de synchronisation et de distribution des SMA au niveau de l'agent et du SMA. SKwyRL-ADL [Faulkner 2003] et ADLMAS [Yu 2006] traitent de la modélisation au niveau micro (agent) et macro (SMA). Cependant, ces deux ADL offrent moins d'abstraction et ne permettent pas l'hétérogénéité parmi les agents qui constituent le SMA, car ils se basent sur une seule architecture d'agent (agent BDI). De plus, l'architecture SMA est fixée lors de la phase de conception et aucun mécanisme d'évolution architecturale n'est prévu pour permettre la reconfiguration du SMA.

2.10.4 Modèles de reconfiguration des SMA

Dans les modèles de reconfiguration, [Kefalas 2005] propose un cadre de modélisation formel pour le développement de systèmes multi-agents avec une structure dynamique. Cette approche est une extension des « communicating X-machines » (CXM). Un ensemble d'opérations de reconfiguration est défini et utilisé pour reconfigurer le SMA. Cependant, les auteurs n'ont pas fourni de solutions pour la vérification des systèmes CXM. Dans [Seghrouchni 2004], un modèle hiérarchique pour la construction de systèmes adaptatifs et à grande échelle est proposé. Cette approche est basée sur le modèle de systèmes ambiant hiérarchique, est implémentée à l'aide du langage de programmation Claim. Ce dernier permet une reconfiguration flexible du système en définissant trois primitives (de mobilité, d'héritage et de création/ suppression dynamique des agents). L'approche proposée est soutenue par une plate-forme distribuée spécifique appelée SyMPA, qui offre les mécanismes nécessaires pour la reconfiguration dynamique du SMA. Bien que ces deux approches fournissent des méthodes de conception efficace, elles manquent d'abstraction lors de la conception vu qu'elles sont orientées vers l'implémentation du SMA et liées à une plateforme de développement comme dans [Seghrouchni 2004]. De plus, certains concepts de SMA ont été négligés tels que le schéma d'interaction entre les agents. A noter que, les modèles comportementaux dans [Kefalas 2005] sont trop complexes pour être vérifiables.

De manière différente, de tous les travaux connexes cités ci-dessus nous notons une approche de modélisation bigraphique présentée par [Mansutti 2014]. Les auteurs ont adopté les systèmes réactifs bigraphiques pour la conception et le prototypage de systèmes multi-agents. Cette approche a une perspective

différente de la nôtre. Elle se concentre principalement sur la partie structurelle du SMA. En outre, aucun détail sur l'évolution de l'architecture SMA n'est donné et aucune vérification n'a été effectuée sur les spécifications obtenues.

2.11 Comparaison

Dans cette section, nous fournissons une courte comparaison entre les méthodes de conception, les modèles organisationnels, les modèles architecturaux et les modèles de reconfiguration des SMA. Dans le tableau 2.3 le symbole "Non" fait référence à un concept non disponible, alors que "Oui" se réfère à la présence explicite du concept ou en utilisant "limité" pour spécifier que le concept est partiellement couvert.
Dans notre analyse comparative des différentes approches de conception de SMA répertoriées sur les 4 groupes identifiés plus haut, nous avons essayé de déterminer des critères de comparaison qui nous semblent essentiels dans la description et la conception des SMA. Ces critères couvrent les aspects structurels, dynamiques et l'aspect lié à l'évaluation et la vérification des spécifications conceptuelles résultantes de chaque approche. Les critères sur lesquels nous avons pu dresser le tableau 2.3 sont :

- La possibilité de décrire les organisations ouvertes,
- La capacité à spécifier la dynamique du système,
- L'indépendance de la conception organisationnelle vis-à-vis des choix d'architecture interne des agents,
- La possibilité d'évaluation des systèmes en cours de conception et enfin la comparaison des choix de conception entre eux.

En ce qui concerne l'aspect structurel, on peut retenir que les concepts explicites les plus partagés, entre les modèles étudiés sont : les agents, les rôles, les organisations et les interactions. D'autres concepts comme le groupe, le service, le but, la tâche, les ressources et les capacités sont explicites dans certains modèles et sont implicites ou non disponibles dans d'autres. Bien que variables, ces concepts sont sémantiquement proches. Nous notons que la majorité des approches présentées dans l'état de l'art ne traitent pas de l'aspect de reconfiguration dynamique dans les SMA. La dynamique organisationnelle de ces solutions est habituellement limitée à une organisation souvent rigide (préétablie), où la fonction principale des agents participants est d'effectuer un rôle spécifique. Par conséquent, elles restreignent l'autonomie de l'agent et la conception d'architectures de systèmes multi-agents reconfigurables, exceptés les modèles de reconfiguration présentés précédemment, et qui permettent de gérer la reconfiguration des spécifications architecturales. Néanmoins, ces

	Ouverture	Dynamique	Indépendance	Évaluation
Modèles organisationnels				
AGR [Ferber 2003]	**Non**	**Non**	**Oui**	**Non**
OperA [Dignum 2004a]	**Limité**	**Non**	**Oui**	**Non**
ISLANDER [Esteva 2002]	**Limité**	**Non**	**Oui**	**Non**
OMNI [Dignum 2004b]	**Oui**	**Non**	**Oui**	**Non**
MOISE+ [Hübner 2002b]	**Oui**	**Non**	**Oui**	**Non**
Méthodes de conception de SMA				
GAIA [Wooldridge 2000a]	**Non**	**Non**	**Oui**	**Non**
INGENIAS [Gomez-Sanz 2002]	**Oui**	**Oui**	**Oui**	**Oui**
PASSI [Cossentino 2005]	**Non**	**Non**	**Oui**	**Non**
TROPOS [Bresciani 2004]	**Non**	**Non**	**Oui**	**Oui**
Modèles architecturaux pour SMA				
DMAS [Park 2005]	**Limité**	**Non**	**Oui**	**Non**
SKWYRL-ADL [Faulkner 2003]	**Non**	**Non**	**Non**	**Oui**
ADLMAS [Yu 2006]	**Non**	**Oui**	**Non**	**Oui**
Modèles de reconfiguration				
CXM [Kefalas 2005]	**Oui**	**Oui**	**Oui**	**Non**
HIMALAYA [Seghrouchni 2004]	**Oui**	**Oui**	**Non**	**Non**

TABLE 2.3 – Modèles de conception et d'analyse de SMA

spécifications sont souvent complexes. Nous déduisons que pour construire un modèle pour la conception à la fois statique et dynamique de SMA, il faut penser à revoir les éléments structurels de base ainsi que leurs interconnexions. Nous remarquons également que l'aspect d'évaluation et de vérification est absent ou non pris en charge car soit (1) les approches proposées ne définissent pas de cadre suffisamment formel pour la mise en place de la vérification de spécification ou (2) l'approche est tellement générique que les spécifications résultantes sont complexes et donc elles ne peuvent être vérifiées (3) cela peut être aussi dû à l'absence d'outils dédiés à la vérification.

2.12 Conclusion

Dans ce chapitre, nous avons défini les concepts de base liés aux agents et SMA. Nous avons commencé le chapitre en donnant les définitions existantes du mot agent, ensuite nous avons identifié ses propriétés, les architectures existantes, les moyens de communication et les différents domaines de son application. La partie la plus importante dans ce chapitre traite de la modélisation des SMA. Dans cette section, nous avons présenté les différentes approches de modélisation et de conception de SMA, répertoriées en 4 groupes : les méthodes de conception, les modèles organisationnels, les modèles architecturaux et les modèles de reconfiguration de SMA. Dans chaque approche présentée, nous avons mis en avant les concepts clés utilisés lors de la modélisation de SMA, leurs orientations (SMA ouverts, émergents, auto-adaptatifs, etc.), les prérequis architecturaux imposés au niveau de chaque méthode et enfin l'évaluation des SMA créés. Ce travail nous a permis de dresser le tableau comparatif (Tableau 2.3) entre les différentes approches. De plus, nous avons pu identifier les points forts et les limites de chaque approche ce qui nous servira de base lors de la proposition de notre approche de conception dans les chapitres qui suivent.

Chapitre 3

Fondements formels

Sommaire

3.1 Introduction

Au cours des cinq dernières décennies, les chercheurs et les développeurs de logiciels ont créé des abstractions qui les aident lors de la conception de systèmes (qu'ils soient distribués ou non), pour permettre la conception de l'espace du problème, indépendamment de l'environnement informatique sous-jacent, en fournissant un modèle du système avant de l'implémenter.

La conception des modèles, aide en général à obtenir une image plus claire des propriétés du système telles que la disposition de ses composants et leur communication. On peut également spécifier des contraintes sur la manière qu'ont les composants pour communiquer entre eux, et les actions qui ne sont pas autorisées dans le système. Pour ce faire, l'utilisation d'un langage formel offre un haut niveau d'abstraction, permettant d'exposer des énoncés de manière précise et sans ambiguïté ; en proposant des modèles formels suffisamment expressifs et mathématiquement analysables. En effet, une méthode formelle est une approche de conception de logiciels informatiques qui permet d'établir avec certitude, s'il existe des effets secondaires, des états logiques imprévus, des incohérences, des conditions impossibles et d'autres problèmes

au niveau du modèle conçu. Typiquement, les méthodes formelles utilisent une combinaison de logique de premier ordre, de logique d'ordre supérieur, de la théorie des ensembles et bien d'autres.

Dans ce travail de thèse, nous nous intéressons aux systèmes réactifs bigraphiques pour la modélisation et la vérification des architectures des systèmes multi-agents. Les systèmes réactifs bigraphiques (BRS) ont été initialement introduits par [Milner 2006] afin de fournir un modèle formel intuitif et entièrement graphique, capable de représenter à la fois la localité et la connectivité des systèmes informatiques ubiquitaires et distribués, ce qui est très proche des concepts des SMA. En outre, les BRS fournissent l'unification des calculs de processus existants pour la concurrence et la mobilité (comme le $\pi-$calcul, les réseaux de Petri, Λ Calcul, etc.) de manière plus simple [Milner 2008].

Ce chapitre présente les concepts fondamentaux de la théorie des bigraphes, en insistant sur les notions qui seront exploitées lors de la présentation de notre contribution. Dans la section 3.2, nous décrivons informellement les notations graphiques représentant un bigraphe et nous donnons ensuite les définitions formelles de ces constituants, à savoir le graphe de places et le graphe de liens. La section 3.2.5 est consacrée à la définition des termes algébriques et des opérations fondamentales sur les bigraphes, en particulier la composition et la juxtaposition. Aussi, la section 3.2.6 traite de la dynamique des bigraphes à travers les règles de réactions. Quelques extensions et applications du formalisme de base (les bigraphes) sont données dans la section 3.2.7. L'ensemble des outils pratiques d'édition et d'exécution des bigraphes, est présenté dans la section 3.2.8. Les définitions données dans ce chapitre sont tirées de [Conforti 2006, Milner 2006, Milner 2008, Milner 2009, Cherfia 2016, Sahli 2017, Benzadri 2016] pour les systèmes réactifs bigraphiques.

3.2 Les systèmes réactifs bigraphiques

Les systèmes réactifs bigraphiques (BRS) [Milner 2008] sont un métamodèle flexible et expressif pour les systèmes informatiques ubiquitaires et distribués. Les états du système sont représentés par des bigraphes, qui sont des structures composites, décrivant à la fois la localité et la connectivité logique des composants (éventuellement imbriqués) d'un système. Comme dans la réécriture des graphes, le comportement dynamique d'un système est défini par un ensemble de règles de réaction (paramétriques). Enfin, il convient de noter qu'un bigraphe est doté d'une représentation graphique et une spécification algébrique équivalente.

3.2.1 Anatomie des bigraphes et forme graphique

Graphiquement un bigraphe (représenté dans la figure 4.1) est contenu dans une racine (aussi appelé région), représentée par des rectangles en pointillé. Dans un bigraphe, le concept de racine permet de décrire l'emplacement des parties du système qui ne sont pas nécessairement adjacentes. Les entités et les composants (réels ou virtuels) d'un système sont exprimés par des nœuds v_i, qui sont dénotés par des cercles, ovales, triangles ou par d'autres formes graphiques. Les interactions entre ces entités (nœuds) sont réalisées à travers des hyper-arcs e_i. On peut également remarquer que les nœuds peuvent être imbriqués les uns dans les autres (par exemple, v1 est imbriqué dans v0). Cette imbrication permet de décrire leurs emplacements spatiaux. Un nœud peut être doté de ports (point noirs), représentant des points de liaison entre les nœuds et les arcs. Chaque nœud a un contrôle (type), qui indique le nombre de ports qu'il peut avoir et permet de déterminer s'il est atomique (il ne peut pas contenir d'autres nœuds) ou non atomique. Dans ce dernier cas, il peut être soit actif soit passif. Les carrés gris sont appelés des sites : ils représentent des emplacements logiques dans lesquels une racine ou un nœud peut être insérés. Les nœuds, les sites et les régions sont appelés les places d'un bigraphe. L'ensemble des contrôles d'un bigraphe est appelé la signature du bigraphe. Un bigraphe peut aussi avoir des noms internes et des noms externes ("inner-names" et "outer-names") ; ces derniers, précisent les liens potentiels avec d'autres bigraphes. Une telle interconnexion n'est possible que si le nom extérieur d'un bigraphe ou d'une racine correspond au nom intérieur d'un autre bigraphe.

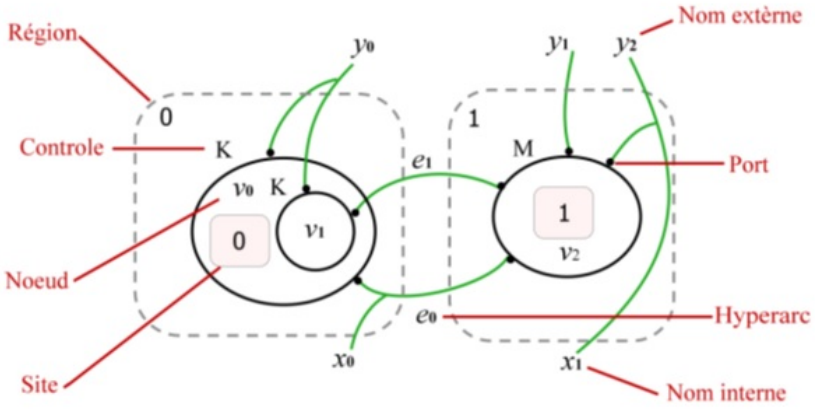


FIGURE 3.1 – Anatomie des bigraphes [Cherfia 2016]

3.2.2 Définitions formelles

Concrètement, un bigraphe est la combinaison de deux structures indépendantes : le graphe de places et le graphe de liens. Le graphe de places représente la répartition géographique des entités du système, alors que le graphe de liens est un hypergraphe qui représente les interconnexions entre ces entités. Dans ce qui suit, nous commençons par la définition formelle de la signature d'un bigraphe, ensuite nous fournissons les définitions respectives d'un bigraphe, d'un graphe de places et d'un graphe de liens.
Formellement, un bigraphe tel qu'il est introduit par [Milner 2009], est comme suit :

Définition 3.1 (signature). La signature d'un bigraphe B prend la forme $(\mathcal{K}, ar)$. Elle comporte un ensemble $\mathcal{K}$ dont les éléments sont des contrôles et une fonction de transformation $ar : \mathcal{K} \to \mathbb{N}$, qui associe une arité à chaque contrôle. La signature $\mathcal{K}$ assigne à chaque nœud d'un bigraphe B un contrôle, dont les arités indiquent le nombre de ports des nœuds. Une signature pour le bigraphe B de l'exemple présenté dans la Figure 3.1 est donnée par $\mathcal{K} = \{N : 2, L : 2, M : 4\}$.

Définition 3.2 (bigraphe). Formellement, un bigraphe prend la forme :

$$G = (V_G, E_G, ctrl_G, G^P, G^L) : I \to J$$

- V_G est un ensemble fini de nœuds qui peuvent être imbriqués.
- E_G est un ensemble fini d'hyper-arcs.
- $ctrl_G : V_G \to \mathcal{K}$ est une fonction de transformation qui associe à chaque nœud $v_i \in V_G$ un contrôle $k \in \mathcal{K}$ indiquant le nombre de ports. La signature $\mathcal{K}$ est un ensemble fini de contrôles.
- $G^P = (V_G, ctrl_G, prnt_G) : m \to n$ est le graphe de places associé à G. m et n représentent respectivement le nombre de sites et de régions. $prnt_G : m \uplus V_G \to V_G \uplus n$ est une fonction de parenté qui associe à chaque nœud ou site son parent hiérarchique.
- $G^L = (V_G, E_G, ctrl_G, link_G) : X \to Y$ est le graphe de liens de G, où $link : X \uplus P \to E_G \uplus Y$ est une fonction de transformation. X, Y et P représente respectivement l'ensemble de noms internes, l'ensemble de noms externes et l'ensemble de ports de G.
- $I = \langle m, X \rangle$ et $J = \langle n, Y \rangle$ représentent respectivement les interfaces internes et externes du bigraphe G. Elles représentent la capacité du bigraphe G d'interagir avec l'environnement extérieur.

3.2.3 Graphe de places

Le graphe de places permet la spécification de la notion de localité (imbrication de nœuds) dans un bigraphe. Il est constitué d'une forêt composée de plusieurs arbres (comme l'illustre la figure 3.2), dont la racine est toujours une région à laquelle sont rattachées hiérarchiquement plusieurs feuilles (des nœuds ou des sites). La figure 3.2 décrit le graphe de places du bigraphe présenté dans la figure 3.1.

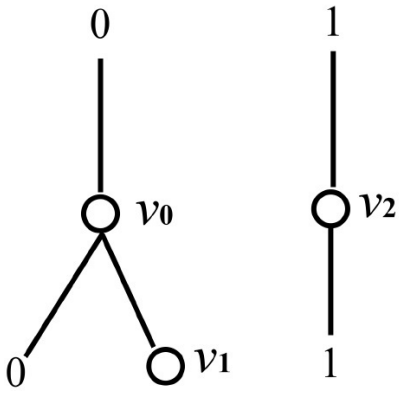


FIGURE 3.2 – Graphe de places

Définition 3.3 (graphe de places). Un graphe de places est défini formellement par :

$$G^P = (V_G, ctrl_G, prnt_G) : m \to n$$

- V_G est un ensemble fini de nœuds.
- $ctrl_G : V_G \to \mathcal{K}$ est une fonction de transformation associant un contrôle à chaque nœud du bigraphe G.
- $prnt_G : m \uplus V_G \to V_G \uplus n$ est une fonction de parenté associant à chaque nœud ou site son parent hiérarchique. La notation $m \uplus V_G$ est utilisée pour indiquer que les deux ensembles sont conservés disjoints.
- m et n représentent respectivement le nombre de sites et de régions du bigraphe G. La notation $m \to n$ permet d'enregistrer les interfaces du graphe de places, où m et n représentent respectivement les interfaces internes et externes du graphe de places.

3.2.4 Graphe de liens

Un graphe de liens permet de décrire la notion de connectivité (interaction entre les nœuds) d'un bigraphe. La figure 3.3 décrit le graphe de liens du bigraphe présenté dans la figure 3.1.

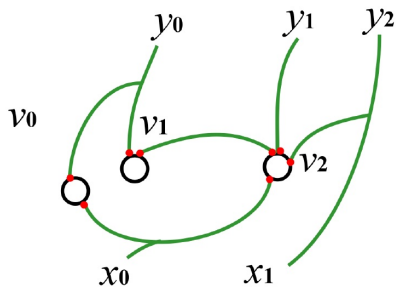


FIGURE 3.3 – Graphe de liens

Définition 3.4 (graphe de liens). Un graphe de liens est défini formellement par :

$$G^L = (V_G, E_G, ctrl_G, link_G) : X \to Y$$

- V_G est un ensemble fini de nœuds.
- E_G est un ensemble fini d'hyper-arcs.
- $ctrl_G : V_G \to \mathcal{K}$ est une fonction de contrôle.
- $link : X \uplus P \to E_G \uplus Y$ est une fonction de transformation montrant le flux de données des noms internes X ou des ports P vers les noms externes Y ou les hyper-arcs E.

3.2.5 Opérations sur les bigraphes

En plus de leur représentation graphique (figure 3.1), les bigraphes sont dotés d'une spécification algébrique. Le tableau 3.1 résume les termes algébriques de base du langage bigraphique. Dans cette section nous allons voir les différentes opérations définies sur les bigraphes [Milner 2008]. En effet, on peut construire des bigraphes complexes à partir de bigraphes élémentaires en utilisant les opérations de composition et de produit tensoriel.

Définition 3.5 (produit parallèle).

Produit parallèle des graphes de places : Si $F_i = (V_i, ctrl_i, prnt_i) : m_i \to n_i$, $(i = 0, 1)$ sont deux graphes de places avec des supports disjoints, leur produit parallèle $F_0 || F_1 : m_0 + m_1 \to n_0 + n_1$ est donné par :

$$F_0 || F_1 \stackrel{\text{def}}{=} F_0 \otimes F_1.$$

Produit parallèle des graphes de liens : Si $F_i = (V_i, E_i, ctrl_i, link_i) : X_i \to Y_i$, $(i = 0, 1)$ sont deux graphes de liens avec des supports disjoints

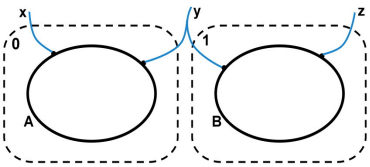
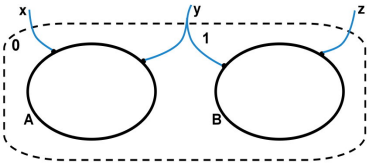
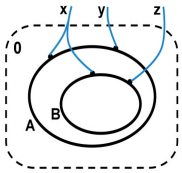
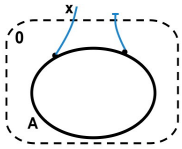
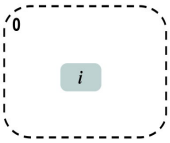
Description	Forme algébrique	Forme graphique
Produit parallèle	$A_x y \| \| B_y z$	
Fusion	$A_x y \| B_y z$	
Imbrication	$A_x y.B_y z$	
Clôture de nom	$/z A_x z$	
Identité (bigraphe élémentaire)	id_i	
Région vide	1	
Site numéroté i	d_i	

TABLE 3.1 – Langage de termes bigraphiques [Sahli 2017]

et $link_0 \cup link_1$ est une fonction de transformation, leur produit parallèle $F_0||F_1 : X_0 \cup X_1 \to Y_0 \cup Y_1$ est donné par :

$$F_0||F_1 \stackrel{\text{def}}{=} (V_0 \uplus V_1, E_0 \uplus E_1, ctrl_0 \uplus ctrl_1, link_O \cup link_1).$$

Produit parallèle des bigraphes : Si $F_i : I_i \to J_i$, $(i = 0, 1)$ sont deux bigraphes avec des supports disjoints, leur produit parallèle $F_0||F_1 : \langle m_0 + m_1, X_0 \cup X_1 \rangle \to \langle n_0 + n_1, Y_0 \cup Y_1 \rangle$ est donné par :

$$F_0||F_1 \stackrel{\text{def}}{=} \langle F_0^P||F_1^P, F_0^L||F_1^L \rangle.$$

Définition 3.6 (fusion des bigraphes). Étant donné deux bigraphes $G_i : \langle m_i, X_i \rangle \to \langle n_i, Y_i \rangle$ avec $(i = 0, 1)$, supposons que $G_0||G_1$ est défini, leur fusion est donnée par :

$$G_0|G_1 \stackrel{\text{def}}{=} (merge_{n_0+n_1} \otimes id_{Y_0 \cup Y_1}) \circ (G_0||G_1)$$

Avec $G_0|G_1 : \langle m_0 + m_1, X_0 \cup X_1 \rangle \to \langle 1, Y_0 \cup Y_1 \rangle$.

Définition 3.7 (imbrication des bigraphes). Étant donné deux bigraphes $F : I \to \langle m, X \rangle$ et $G : m \to \langle n, Y \rangle$, leur imbrication $G.F : I \to \langle n, X \cup Y \rangle$ est définie par :

$$G.F \stackrel{\text{def}}{=} (G||id_x) \circ F.$$

3.2.5.1 Produit Tensoriel

Cette opération importante, est définie entre deux bigraphes ayant des interfaces disjointes. Elle est fondamentale car, elle permet la composition des bigraphes, et également appelée composition horizontale, et notée par $\otimes$. Cette opération consiste à mettre côte à côte les régions des bigraphes, et à réaliser l'union de leurs graphes de liens (en joignant les liens ouverts communs). Plus précisément, on dit que deux bigraphes $F_i : \langle m_i, X_i \rangle \to \langle n_i, Y_i \rangle$, avec $(i = 0, 1)$ ayant des interfaces disjointes si $X_0 \cap X_1 = \emptyset$ et $Y_0 \cap Y_1 = \emptyset$. Le produit tensoriel sur les interfaces $I = \langle m, X \rangle$ et $J = \langle n, Y \rangle$ est défini par $I \otimes J = \langle m + n, X \uplus Y \rangle$. Un exemple de l'utilisation de l'opération du produit tensoriel sur deux bigraphes disjoints, respectivement G (Figure 3.4a) et B (Figure 3.4b) est représenté par la figure 3.4 [Sahli 2017].

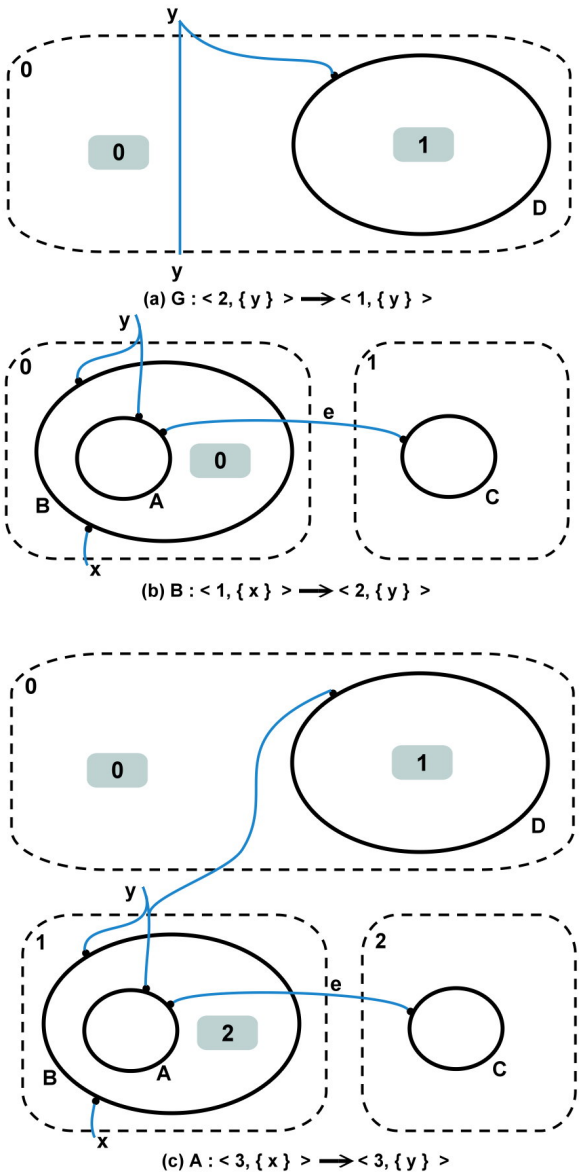


FIGURE 3.4 – Produit tensoriel des bigraphes : $A = G \otimes B$ [Sahli 2017]

Définition 3.8 (produit tensoriel des graphes de places). Si $F_i = (V_i, ctrl_i, prnt_i) : m_i \to n_i$, $(i = 0, 1)$ sont deux graphes de places disjoints, leur produit tensoriel $F_0 \otimes F_1 : m_0 + m_1 \to n_0 + n_1$ est donné par :

$$F_0 \otimes F_1 \stackrel{\text{def}}{=} (V_0 \uplus V_1, ctrl_0 \uplus ctrl_1, prnt_O \uplus prnt'_1)$$

où $prnt'_1(m_0 + i) = n_0 + j$ lorsque $prnt_1(i) = j$.

Définition 3.9 (produit tensoriel des graphes de liens). Si $F_i = (V_i, E_i, ctrl_i, link_i) : X_i \to Y_i$, $(i = 0, 1)$ sont deux graphes de liens disjoints, leur produit tensoriel $F_0 \otimes F_1 : X_0 \uplus X_1 \to Y_1 \uplus Y_2$ est donné par :

$$F_0 \otimes F_1 \stackrel{\text{def}}{=} (V_0 \uplus V_1, E_0 \uplus E_1, ctrl_0 \uplus ctrl_1, link_O \uplus link_1).$$

Définition 3.10 (produit tensoriel des bigraphes). Si $F_i : \langle m_i, X_i \rangle \to \langle n_i, Y_i \rangle$, $(i = 0, 1)$ sont deux bigraphes avec des supports disjoints, leur produit tensoriel $F_0 \otimes F_1 : \langle m_0 + m_1, X_0 \uplus X_1 \rangle \to \langle n_0 + n_1, Y_0 \uplus Y_1 \rangle$ est donné par :

$$F_0 \otimes F_1 \stackrel{\text{def}}{=} \left\langle F_0^P \otimes F_1^P, F_0^L \otimes F_1^L \right\rangle.$$

3.2.5.2 Composition

On peut voir l'opération de composition comme l'insertion d'un bigraphe dans un autre. Exemple : le résultat de la composition de deux bigraphes B et G est un nouveau bigraphe A (voir figure 3.5). Pour cela, il est impératif que le nom externe du bigraphe B soit égal à celui du bigraphe G. Algébriquement l'opération de composition est notée comme suit : G ° B. Plus particulièrement, quand un bigraphe est inséré dans un autre, ces noms externes sont attachés aux noms internes, correspondant au bigraphe hôte, et ces régions sont insérées au niveau des sites du bigraphe hôte. Dans l'exemple, la racine en B contenant le nœud C est fusionnée avec le site présent à l'intérieur du nœud D du bigraphe G. En outre, le nœud D est lié aux nœuds du contrôle A et B à travers le nom commun y. Parfois, le bigraphe G est considéré comme le contexte ou l'environnement pour le bigraphe B. Nous définissons maintenant une composition séparée pour les graphes de place concrets et les graphiques de liens. La composition de deux bigraphes est définie séparément pour la composition des graphes de places et des graphes de liens ; comme le précise la figure 3.5 [Milner 2009] :

Définition 3.11 (composition des graphes de places). Si $F : k \to m$ et $G : m \to n$ sont deux graphes de places avec des supports disjoints, leur composite $G \circ F = (V, ctrl, prnt) : k \to n$ à l'ensemble de nœuds $V = V_F \uplus V_G$ et la fonction contrôle $ctrl = ctrl_F \uplus ctrl_G$. Sa fonction de parenté $prnt$ est définie par : Si $w \in k \uplus V$ est un site ou un nœud dans $G \circ F$ alors

$$prnt(w) \stackrel{\text{def}}{=} \begin{cases} prnt_F(w) & \text{si } w \in k \uplus V_F \text{ et } prnt_F(w) \in V_F, \\ prnt_G(j) & \text{si } w \in k \uplus V_F \text{ et } prnt_F(w) = j \in m, \\ prnt_G(w) & \text{si } w \in V_G. \end{cases}$$

Définition 3.12 (composition des graphes de liens). Si $F : X \to Y$ et $G : Y \to Z$ sont deux graphes de liens avec des supports disjoints, leur composite $G \circ F = (V, E, ctrl, link) : X \to Z$ à un ensemble de nœuds $V = V_F \uplus V_G$, un ensemble d'hyper-arcs $E = E_F \uplus E_G$ et une fonction de contrôle $ctrl = ctrl_F \uplus ctrl_G$. Sa fonction $link$ est définie par : Si $q \in X \uplus P_F \uplus P_G$ est un point de $G \circ F$ alors

$$link(q) \stackrel{\text{def}}{=} \begin{cases} link_F(q) & \text{si } q \in X \uplus P_F \text{ et } link_F(q) \in E_F, \\ link_G(y) & \text{si } q \in X \uplus P_F \text{ et } link_F(w) = y \in Y, \\ link_G(q) & \text{si } q \in P_G. \end{cases}$$

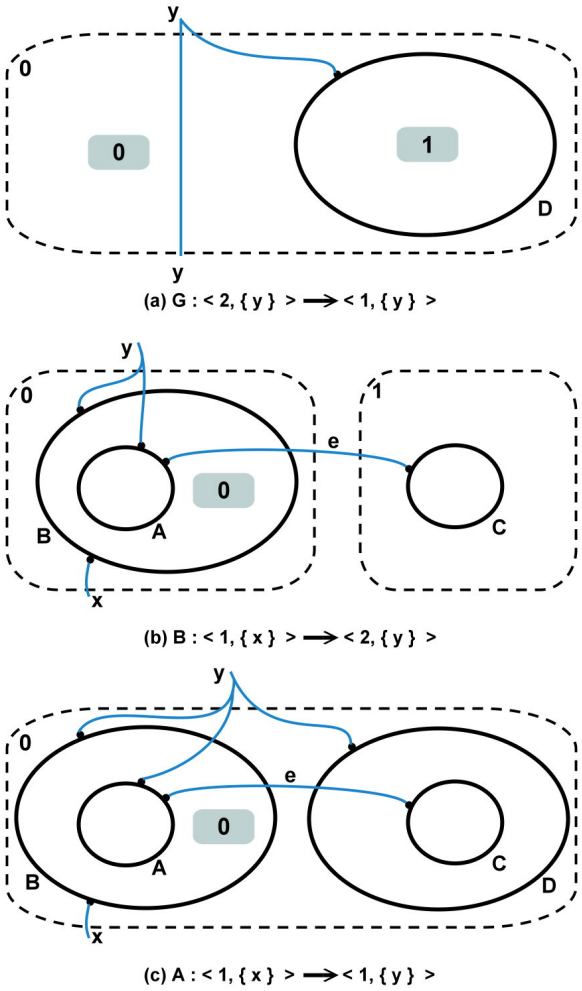


FIGURE 3.5 – Composition des bigraphes $A = G \circ B$ [Sahli 2017]

Définition 3.13 (composition des bigraphes). Si $F : \langle k, X\rangle \to \langle m, Y\rangle$ et $G : \langle m, Y\rangle \to \langle n, Z\rangle$ sont deux bigraphes avec des supports disjoints, leur composite est donné par : $G \circ F \stackrel{\text{def}}{=} (G^P \circ F^P, G^L \circ F^L) : \langle k, X\rangle \to \langle n, Z\rangle$

3.2.6 Aspect dynamique des bigraphes

La dynamique structurelle des BRS (Bigraphical Reactive System) est exprimée par une catégorie de bigraphes et un ensemble de règles de réactions. En effet, les BRS sont dotés de moyens qui leur permettent de reconfigurer leurs structures. Ces reconfigurations sont régies par un ensemble de règles de réactions. Un rapprochement sémantique peut être fait, avec les règles génériques dédiées pour la réécriture de graphes. Toutefois, les règles de réactions bigraphiques diffèrent des règles de réécriture habituelle car elles sont paramétrées. Simplement, une règle de réaction $R \rightarrow R'$ est une paire de bigraphes <redex, reactum>. Le redex représente la pré-condition pour le déclenchement de la règle, appelé aussi bigraphe contextuel. Il représente le pattern à modifier. Le résultat de l'application de la règle de réaction est un bigraphe appelé reactum. La reconfiguration d'un bigraphe est effectuée de manière distincte sur le graphe de liens, en ajoutant ou en supprimant des liens, et sur le graphe de place en déplaçant, ajoutant et supprimant des nœuds. Un exemple d'application d'une règle de réaction bigraphique est représenté dans la figure 3.6. Cette règle modélise la dynamique comportementale d'un utilisateur (nœud de contrôle P) qui veut utiliser un ordinateur (nœud de contrôle PC) se situant dans une salle (nœud de contrôle R). Cette règle de réaction peut être représentée algébriquement par : $P|R.(PC) \rightarrow R.(P_e|PC_e)$.

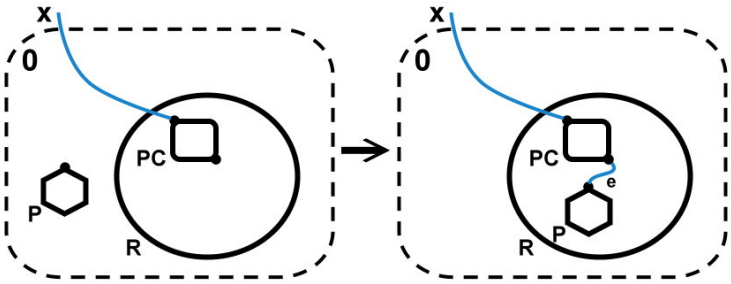


FIGURE 3.6 – Exemple d'une règle de réaction bigraphique [Sahli 2017]

Définition 3.14 (règle de réaction). Une règle de réaction prend la forme : $R = (R : m \rightarrow J, R' : m \rightarrow J, \eta)$ et est généralement écrite $R \rightarrow R'$, où $R : m \rightarrow J$ est un bigraphe appelé redex (pattern à changer), $R' : m \rightarrow J$ est également un bigraphe appelé reactum (nouveau bigraphe changé) et l'instanciation $\eta : m' \rightarrow m$ est une transformation d'ordinaux.

Après avoir défini formellement un bigraphe et une règle de réaction, nous pouvons maintenant donner la définition d'un système réactif bigraphique (BRS).

Définition 3.15 (système réactif bigraphique). Un système réactif bigraphique (BRS) consiste en une paire (B, R), où B est un ensemble de bigraphes et R est un ensemble de règles de réaction définies sur B. La relation de réaction d'un BRS est représentée par $\rightarrow$.

3.2.7 Extensions des Bigraphes

Depuis l'avènement des bigraphes [Milner 2006]. Plusieurs chercheurs ont contribué à faire évoluer ce nouveau formalisme, qui diffère des formalismes traditionnels tels que la notation Z, les réseaux de Petri, la méthode B et l'algèbre de processus, par son aspect graphique et sa capacité de représenter à la fois la localité et la connectivité des systèmes informatiques ubiquitaires et distribués. A cet effet, un nombre de propositions ont vu le jour sous forme d'extension de la définition standard des bigraphes, tels que défini par Milner. Ces extensions et raffinements ont été dirigés par le besoin de leurs initiateurs lors de l'application des bigraphes sur un domaine bien précis. On note quatre extensions majeures. Dans ce qui suit, nous présentons chaque notion introduite au niveau de chaque extension.

"Binding Bigraphs" [Damgaard 2006]. Cette extension des bigraphes, offre plus de flexibilité entre le graphe de places et le graphe de liens. La nouveauté de cette extension est qu'elle autorise l'assignement de noms communs au sein des racines et sites. L'apport de cette extension est constaté lors de la modélisation de protocole de communication sécurisé.

"Stochastic Bigraphs" [Krivine 2008]. Dans le but de modéliser le contexte des systèmes moléculaires utilisant des compartiments, ainsi que l'analyse stochastique des systèmes distribués de manière générale, cette extension a été proposée. Dans un BRS, la partie qui traite de la dynamique et de la reconfiguration du système est exprimée à l'aide de règles de réactions. Dans cette approche, une valeur stochastique a été attribuée à chaque règle de réaction. En conséquence, à partir de la séquence de l'ensemble des règles de réactions qui constituent l'espace d'état ; on peut tirer une chaîne de Markov à temps continu (CTMC). De ce fait, on peut obtenir le nombre possible d'occurrences d'une règle. Un tel taux représente le comportement stochastique de cette règle de réaction.

"Directed Bigraphs" [Grohmann 2007]. Dans cette troisième extension, le concept ajouté est identifié au niveau du graphe de liens. Plus particulièrement, un sens a été attribué aux arêtes, ceci afin de préciser le sens du flux de demande des ressources. Les bigraphes dirigés se sont vu

appliquer comme une théorie générale à partir de laquelle, des systèmes de transitions peuvent être dérivés. L'apport de cette extension est constaté dans le domaine de la sécurité des systèmes informatiques (la cryptographie).

"Bigraphs with sharing" [Sevegnani 2015]. L'apport principal de cette extension réside dans le remplacement du simple graphe de place standard par un "DAG" (graphe orienté acyclique), ce qui permet d'introduire la possibilité pour un nœud d'hériter de plusieurs nœuds parents. Comme domaine d'application, cette approche a fait ses preuves dans le domaine des réseaux sans fil.

3.2.8 Outils autour des BRS

Selon Milner, un outil dédié pour les bigraphes doit permettre la spécification des modèles bigraphiques, la visualisation graphique et la vérification des spécifications bigraphiques. A cet effet un nombre d'outils dédiés pour les BRS ont été proposés :

Tool for Bigraphical Programming Languages. BPL [Højsgaard 2011], est un outil destiné pour modéliser et expérimenter les BRS a liaison abstraite. Les objectifs de l'outil BPL, sont (i) d'implémenter la théorie complète des BRS (ii) et d'avoir une correspondance étroite entre la théorie et sa mise en œuvre pour atteindre un degré de fiabilité et de confiance dans les spécifications résultantes. Pour atteindre ces objectifs, les développeurs de cet outil, se sont basés sur un algorithme de matching ainsi que sur le langage de terme et la forme normale des bigraphes à liaison abstraite, tels que développés par Damgaard et Birkeda. BPL Tool, permet la manipulation, la simulation et la visualisation des systèmes réactifs bigraphiques. Il a été utilisé dans plusieurs domaines on cite entre autres : les systèmes de téléphonie mobile [Højsgaard 2011], WS-BPEL, HomeBPEL [Bundgaard 2008] et les modèles platographiques [Elsborg 2009].

BigMC [Perrone 2012]. Bigraphical Model Checker est un model-checker dedié pour la verification de systèmes modelisés par des BRS. Il se base sur la technique de matching (définie dans BPL Tool). Algorithmiquement, cet outil permet d'explorer les différents états possibles d'un système réactif bigraphique, afin de vérifier une certaine propriété sur le système. Dans le cas où la propriété voulue n'est pas satisfaite (vérifiée), BigMC fournit un contre-exemple qui permet, de détecter la cause qui a permis d'arriver à l'état actuel (violation). La structure d'un modèle spécifié à l'aide de la syntaxe BigMc est la suivante :

Bigrapher [Sevegnani 2016]. est une implémentation du système réactif

```
# Comments
<control definitions>

<names>

<reaction rules>

<model definition>

<properties>

%check;
```

FIGURE 3.7 – Syntaxe du BigMC

bigraphique de Robin Milner (BRS) avec une extension qui prend en charge les bigraphes avec partage. Ses caractéristiques principales sont :

- Un soutien natif pour les bigraphes et les bigraphes avec le partage
- Un moteur de matching efficace basé sur SAT
- Des règles de réaction stochastiques
- Des priorités de la règle
- Des règles avec des cartes d'instanciation
- Des définitions fonctionnelles
- Une sortie graphique
- Une simulation (Stochastique)
- Une exploration exhaustive de l'espace d'état
- Une exploration vers un modèle checker probabiliste PRISM
- Une vérification de prédicat

3.3 Conclusion

Dans cette section, nous avons présenté la théorie des bigraphes tel que developpée par Milner, dans un détail suffisant pour permettre une bonne compréhension de nos travaux au chapitre 4. Nous avons commencé par la présentation et la définition de cette dernière et des systèmes réactifs bigraphiques (BRS) qui est un modèle graphique dédié à la modélisation de systèmes ubiquitaires et distribués, mettant en avant, deux aspects clés, qui sont

la connectivité et la localité. En effet, les BRS sont un méta-modèle qui peut être instancié avec une syntaxe particulière, spécifiée par une signature et une sémantique et dont la dynamique est mise en œuvre par un ensemble de règles de réactions. Ensuite, nous avons succinctement présenté certains développements de la théorie qui sont particulièrement pertinents. Enfin, nous avons terminé ce chapitre par la présentation d'un ensemble d'outils autour du développement et de la vérification des systèmes modélisés en utilisant les BRS.

Chapitre 4

BRS-MAS modèle générique pour la spécification des architectures des SMA

Sommaire

4.1 Introduction

Depuis son émergence dans les années 1980, le domaine des systèmes multi-agents (SMA) s'est imposé comme un domaine de recherche reconnu. Ce dernier combine les recherches de plusieurs domaines. On évoque plus particulièrement l'intelligence artificielle distribuée (IAD) et le génie logiciel (GL). Ces deux derniers constituent à leur tour le domaine de l'ingénierie logicielle orientée agent (Agent-Oriented Software Engineeringg), dont l'objectif est de fournir des méthodes pratiques pour développer des systèmes multi-agents. Les SMA ont été jugés comme une solution efficace aux problèmes informatiques décentralisés, situés dans des environnements dynamiques, ouverts et ayant des capacités d'intelligence tels que : l'apprentissage

[Haynes 1998], la négociation [Kraus 1995, Rosenschein 1998], le comportement social [Sandhlom 1997, Shehory 2014b, Shoham 1992, Demazeau 1990] et les activités mentales (croyances, désirs, intentions [Rao 1995], le raisonnement [Halpern 1997]). En effet, il y a un nombre considérable d'applications réelles qui ont été déployées avec succès dans divers secteurs d'application telles que : les télécommunications, la gestion des grilles, la robotique [Müller 2014]. De plus, le domaine des systèmes multi-agents est soutenu par un grand nombre de chercheurs et de praticiens à travers des conférences réussies : AAMAS, IAT, MATES, et PAAMS, et des journaux comme JAAMAS, AAAI, AAIJ, et KER [Müller 2014].

Les développeurs de systèmes informatiques ont souvent besoin de développer des composants logiciels qui peuvent déjà exister dans d'autres solutions. Dans de ce cas, on peut parler de développement par réutilisation de composants. Ceci permet un réel gain de temps lors de la construction de l'architecture d'un système logiciel [Allen 1998]. Une architecture logicielle offre une vue d'ensemble de la structure et de l'organisation d'un système logiciel ainsi que des propriétés non structurelles associées. Ce qui rend l'architecture logicielle, comme un support qui facilite la compréhension du modèle étudié.

La recherche dans le domaine des SMA a introduit une variété d'approches pour supporter la modélisation des différentes propriétés d'intelligence, d'autonomie et d'interaction spécifiques aux SMA. Néanmoins, ces solutions ont rarement traité la modélisation des SMA d'un point de vue architectural. On rapporte également l'importance des agents et des systèmes multi-agents en tant qu'architecture logicielle pour la construction des systèmes informatiques distribués [Shehory 2014b]. Nous sommes convaincus de l'apport des architectures logicielles pour la construction des SMA, car elles introduisent une simplicité et une vue d'ensemble du problème offrant ainsi une meilleure compréhension et une utilisation adéquate par le praticien. Bien que les systèmes et technologies multi-agents affichent des caractéristiques clés pour l'ingénierie des systèmes complexes, ils opèrent souvent dans des environnements dynamiques et sont confrontés à l'émergence de nouvelles exigences, rendant difficile la conception et le développement d'une architecture SMA reconfigurable.

Une architecture SMA reconfigurable peut être définie comme un ensemble flexible d'agents autonomes distribués, munis de moyens qui leur permettent d'adapter dynamiquement leur structure organisationnelle, en réponse à l'émergence de nouveaux changements environnementaux (par exemple, le comportement opérationnel peut évoluer ; les agents peuvent migrer). Afin de faire face aux défis croissants de l'adaptation et de la reconfiguration dans les SMA. Nous nous sommes inspirés à la fois des ADL, qui offrent un niveau d'abstraction adéquat (en considérant le schéma organisationnel comme

l'assemblage d'entités autonomes), et des bigraphes, qui apportent une dimension formelle, en permettant de spécifier la distribution physique et logique des agents sous forme d'une configuration hiérarchique. Par conséquent, on peut exprimer un large éventail d'organisations de SMA (par exemple, organisation modulaire, hiérarchique, etc.). De plus, l'aspect graphique des bigraphes permet d'avoir une représentation claire et intuitive des architectures de SMA. Plus précisément, nous nous basons sur les BRS pour gérer la complexité des aspects structuraux et comportementaux des architectures des SMA reconfigurables. Les BRS présentent une solution prometteuse et efficace pour le développement d'un SMA sûr et de qualité (à un coût et une durée raisonnables).
La modélisation d'un SMA reconfigurable implique deux dimensions : une dimension structurelle (ou statique) et une dimension dynamique. Dans ce chapitre, nous présentons notre approche architecturale BRS-MAS dédiée à la modélisation des architectures des SMA reconfigurables.

4.2 Exemple de motivation : « système de gestion de véhicules intelligents et d'aide à la conduite »

Tout comme dans les autres secteurs, l'adoption des systèmes multi-agents dans le domaine des systèmes sensibles au contexte, est de plus en plus importante. Effet, les SMA offrent une couverture et une prise en charge des concepts clés des systèmes sensibles au contexte : (1) la modélisation du contexte (l'environnement), ainsi que les changements possibles qui peuvent avoir lieu sur ce dernier, (2) la modélisation des entités (physiques ou logiques) et de leurs localisations, mobilité et leur interaction. De plus, les SMA offrent une dimension d'intelligence qui se manifeste par un raisonnement intelligent et une autonomie des entités qui constituent le système multi-agent. Les systèmes de gestion de véhicules intelligents et d'aide à la conduite sont une déclinaison applicative des systèmes sensibles au contexte qui sont omniprésent dans la vie de tous les jours.

Un système de gestion de véhicules intelligents et d'aide à la conduite (voir figure 4.1) est conçu pour prévenir les accidents de circulation. Une voiture intelligente comprend un certain nombre de capteurs (capteurs de vitesse, capteurs de pluie, GPS, etc.), des modules de traitement et des actionneurs (c.-à-d. le régulateur de vitesse, l'alarme d'avertissement, le système de stationnement automatique, etc.) qui sont utilisés pour collecter et interpréter les informations de l'environnement, du conducteur, du trafic et de la voiture elle-même. Ces informations sont utilisées afin de fournir les services d'assistance

les plus fiables en temps réel [Cherfia 2016].

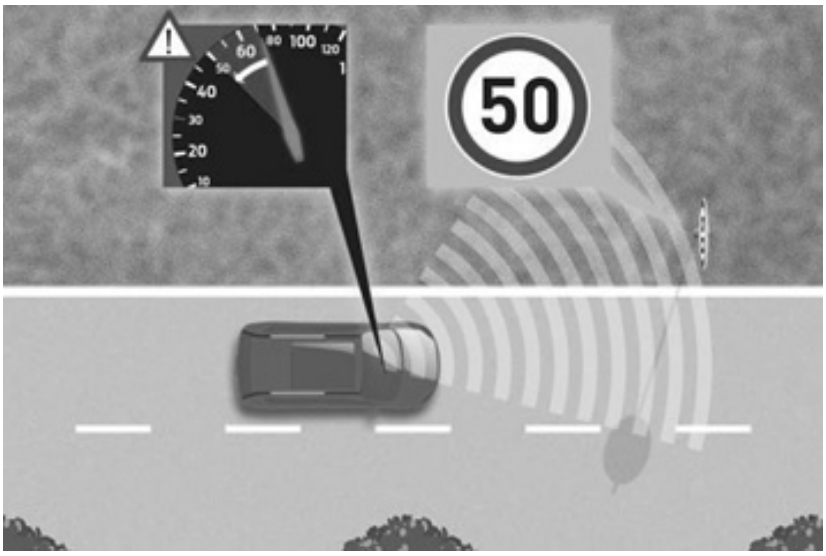


FIGURE 4.1 – Exemple de motivation

4.3 Principe de modélisation

L'utilisation des méthodes formelles offre suffisamment de souplesse et d'expressivité pour spécifier rigoureusement les aspects conceptuels des systèmes informatiques. Elles permettent une intégration précoce de la vérification au niveau du processus de conception. Par conséquent, on peut vérifier le système à partir de sa spécification sans avoir à l'implémenter.

La spécification formelle des architectures des systèmes multi-agents aide à obtenir une meilleure représentation et donc améliorer la qualité de la conception. Elle permet aussi, d'avoir une vision plus abstraite (et modulaire) de l'architecture des systèmes multi-agents. Dans ce qui suit, nous allons aborder la spécification des architectures des systèmes multi-agents en utilisant les BRS. Mais avant cela, nous devons tout d'abord identifier les concepts et éléments architecturaux qui constituent une architecture SMA. Ensuite, nous proposons un méta-modèle semi-formel qui regroupe les principaux concepts des SMA ainsi que les différentes dépendances qui les relient.

4.3.1 Éléments architecturaux de modélisation

Dans les systèmes multi-agents : les agents, l'environnement et les interactions représentent les concepts majeurs (voir figure 4.2).

- **Un agent** est un système informatique situé dans un environnement, il est flexible et autonome dans les actions qu'il entreprend afin de répondre à ces objectifs de conception [Jennings 2001].

  Cette définition est basée sur les trois mots clés suivants :

  - **Situé :** signifie que l'agent peut recevoir des données sensorielles depuis son environnement, et agir sur ce dernier en effectuant des actions qui peuvent le modifier.
  - **Autonomie :** signifie que l'agent peut agir par lui-même sans intervention externe.
  - **Flexibilité :** est liée à la notion d'objectif et d'intelligence.

  Les agents sont considérés comme des granules de composants communicants, responsables de tâches précises. Selon leurs propriétés et leurs capacités les agents peuvent être classés dans l'une des trois principales catégories : réactive, cognitive, hybride.
- **L'environnement** représente l'espace dans lequel les agents se déplacent, autrement dit, tout ce qui n'est pas à l'intérieur d'un agent est considéré comme un environnement. On distingue trois types d'environnements : (1) Environnement virtuel (2) Environnement discret, et (3) Environnement continu représenté par une location physique.
- **L'interaction** est un changement, une réaction sur l'état d'un (ou plusieurs) agents causés par un (ou plusieurs) agents à un instant t. Il existe 3 types d'interactions au niveau d'un SMA : (1) interaction directe, (2) indirecte, (3) complexe / conditionnelle / collective.

Un système multi-agent est défini par l'émergence d'un comportement global généré par un ensemble d'interactions entre agents pour résoudre des problèmes qui sont au-delà des capacités de raisonnement d'un seul agent [Sycara 1998]. Un SMA peut être classé selon plusieurs critères : taille et nombre d'agents, mécanismes d'interaction, etc. Cependant, on distingue deux caractéristiques principales dans les SMA :

- **Les capacités cognitives** des agents précisent la capacité des agents, à planifier leurs actions, à raisonner sur les actions et les plans d'autres agents et à évaluer l'environnement.
- **L'organisation du SMA** définit les relations et les structures des communications entre les agents et le degré de coopération. Ces agents « intelligents » ont caractéristiques suivantes : (1) une représentation explicite de la connaissance, (2) la structuration de leurs états mentaux, et (3) des capacités cognitives (mise à jour des connaissances, la planification et l'autonomie).

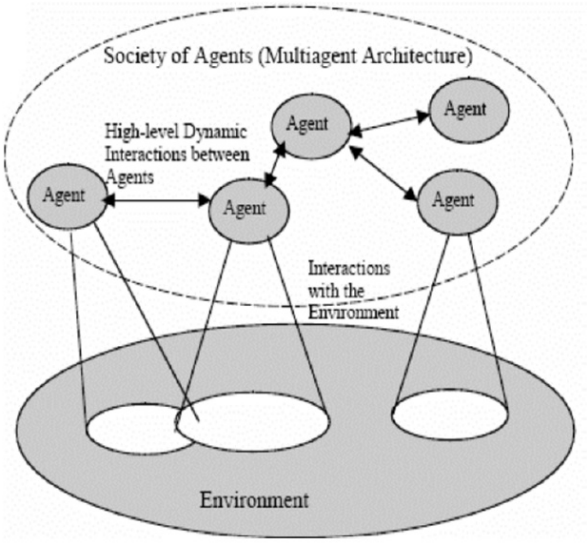


FIGURE 4.2 – Architecture d'un SMA [Ferber 1998]

Les agents au sein d'un SMA peuvent être repartis en groupes, ce qui leur permet d'être plus efficace et donc mieux coopérer pour atteindre leurs buts. Les organisations de SMA peuvent être étudiées sur trois niveaux [Rihawi 2014] (voir figure 4.3) :

- **Le niveau micro**(le niveau le plus bas) : on se focalise sur l'agent et les moyens d'interaction avec ce dernier.
- **Le niveau méso** (le niveau intermédiaire) : le SMA est vu comme des ensembles ou des groupes d'agents. Où chaque groupe partage un même intérêt, donc un but commun.
- **Le niveau macro** (le niveau le plus abstrait) : le comportement des agents est vu comme un comportement global unique.

De plus, on peut utiliser les niveaux : micro, méso et macro pour construire des SMA de deux manières différentes :

- **De haut en bas** (descendante ou top-down, approche orientée organisation) : on commence par specifier l'organisation globale puis on passe à la définition de la structure interne des agents au niveau micro.
- **De bas en haut** (ascendante ou bottom-up, approche émergente) : on commence à partir du niveau micro en définissant les capacités des agents pour aboutir au niveau macro.

Les systèmes multi-agents doivent souvent opérer dans des environnements dynamiques, et faire face au défi de l'évolution constante de leurs exigences ;

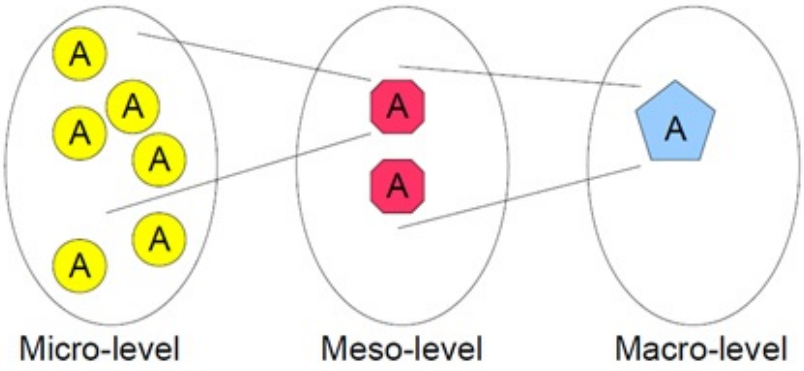


FIGURE 4.3 – les différents niveaux de conception [Navarro 2013]

Donc ils doivent être flexibles, robustes et capables de s'adapter à leurs environnements [Wooldridge 2009]. Par conséquent, le développement de systèmes multi-agents est une tâche d'ingénierie complexe. Pour relever ce défi, il est nécessaire d'élever le niveau d'abstraction des systèmes bien au-delà des lignes de code, et de les diviser en modules (c.-à-d. en sous-problèmes) [Parnas 1972] afin de gérer la complexité et faciliter le raisonnement. Dans le domaine du génie logiciel, l'architecture logicielle est une façon reconnue pour répondre à ces attentes. Elle vise à fournir des descriptions de haut niveau des systèmes en représentant, non seulement leurs structures logiques, mais aussi beaucoup d'autres aspects fonctionnels et non fonctionnels (par exemple, comportement, sécurité, etc,). Elle permet également de décrire l'ensemble des propriétés et des contraintes que le système doit respecter. Offrant une vision globale et une représentation de haut niveau de la structure et de l'organisation d'un système, l'architecture logicielle joue un rôle clé en tant que pivot entre les exigences d'un système et sa mise en œuvre. À un haut niveau d'abstraction, les langages de description d'architecture (ADL) ont été adoptés comme des outils formels pour décrire l'architecture de logiciels. De plus, peu importe si nous modélisons le comportement d'une organisation humaine, un écosystème complexe ou la dynamique d'un énorme marché : on peut s'attendre à trouver des modèles répétés, un schéma partagé, des lois communes qui font que ces systèmes paraissent semblables lorsqu'ils sont observés à partir d'un bon niveau d'abstraction. Par conséquent, nous sommes convaincus que le domaine du génie logiciel à travers les concepts d'architecture, (représenté par les langages de description d'architecture) ainsi que celui des langages formels apportent le degré d'abstraction adéquat pour concevoir des SMA plus sûrs et plus efficaces.

4.3.2 Méta-modèle d'un système multi-agent

L'ensemble des notions et des concepts d'un système multi-agent pris en considération dans notre modèle bigraphique est issu du méta-modèle illustré par la Figure 4.4. Ce méta-modèle cerne tous les éléments principaux, jugés nécessaires pour la modélisation des architectures de ce type de systèmes. Dans le méta-modèle de la Figure 4.4, les éléments architecturaux sont représentés sous la forme d'instances de SMA de différents types. Une instance peut être un agent, un groupe d'agent, ou toute une organisation, ces diverses instances possèdent des attributs similaires qui décrivent leurs identifiants et leurs états. Dans un système multi-agent, ces instances sont hébergées généralement au niveau d'une architecture où il existe une relation hiérarchique entre les différents types d'instances. Une architecture peut contenir des instances de type agent et groupe.

Nous pouvons distinguer trois types d'agents dans le méta-modèle, des agents réactifs, cognitifs et hybrides. Les agents de notre architecture SMA peuvent faire partie d'un groupe donné et ont un rôle bien déterminé au sein du groupe. L'interaction entre agents se fait à l'aide d'interfaces. On distingue trois types d'interaction : la coordination, la coopération et enfin la négociation, en suivant un protocole d'échange bien déterminé. Par conséquent, le méta-modèle défini, vise à capturer un ensemble "central" d'abstractions conceptuelles, y compris les relations et les contraintes qui sont fondamentales pour la description de toute architecture SMA, ce qui permet de simplifier considérablement le passage vers les systèmes réactifs bigraphiques. Néanmoins, contrairement aux méthodes formelles et plus spécifiquement les BRS, ce modèle semi-formel ne permet pas de définir la sémantique comportementale ou de réaliser des vérifications des propriétés des architectures de SMA.

4.4 Spécification formelle des architectures des systèmes multi-agents

Nous commençons cette section en présentant un ensemble de règles, qui définissent les correspondances entre la sémantique des SMA et des bigraphes (voir le tableau 4.1). Nous formalisons la structure organisationnelle d'un SMA par un bigraphe, où chaque agent est représenté par un nœud, dont les modules internes sont modélisés comme des nœuds imbriqués. On affecte à chaque nœud bigraphique un contrôle définissant son type, ses attributs et son nombre de ports. De plus, les agents impliqués dans une architecture SMA organisationnelle forment des groupes. Cela est formalisé à l'aide de racines, qui représentent la distribution logique ou physique des agents. Les

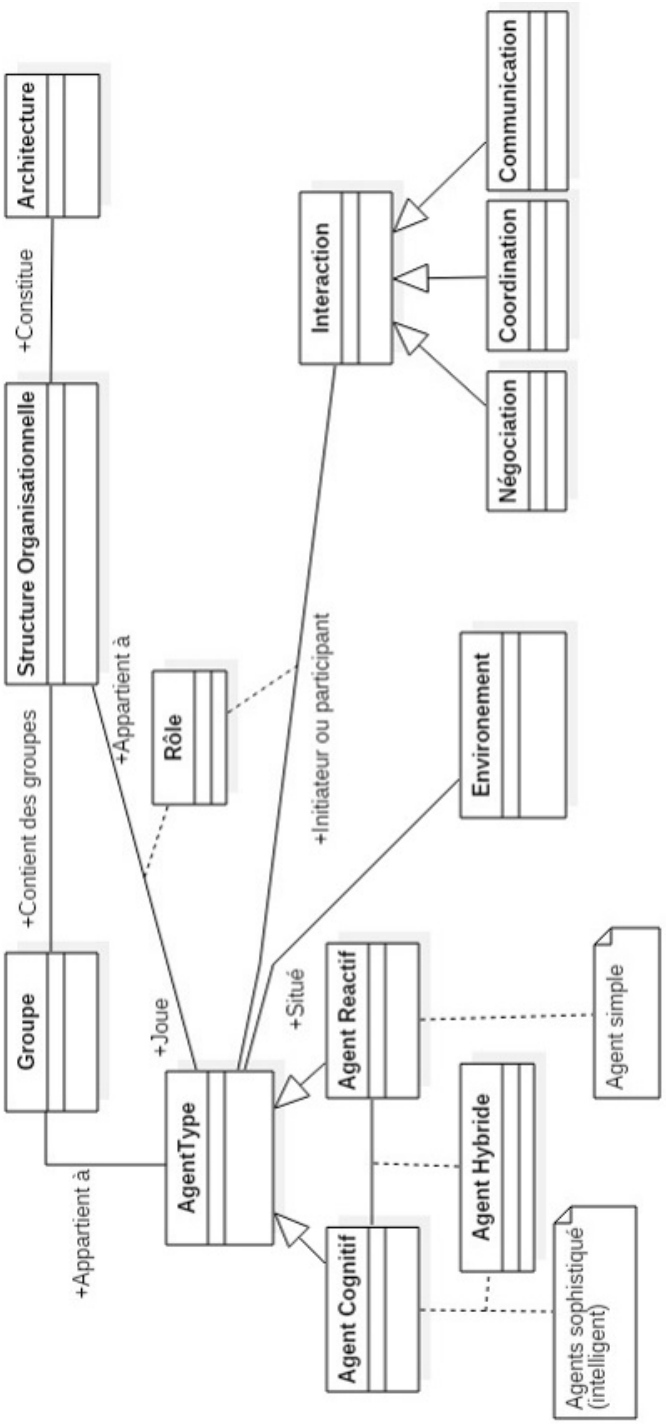


FIGURE 4.4 – Méta-modèle d'un système multi-agent

interactions entre agents sont modélisées par des liens ou des hyper-liens.

Les éléments d'architecture d'un SMA	Sémantique bigraphique
L'architecture SMA	Bigraphe : $G_{SMA} = (V_{SMA}, E_{SMA}, ctrl_{SMA}, G^P_{SMA}, G^L_{SMA}) : I \to J$
Agent	Nœud : $V_i \in V_{SMA}$
Interaction	Hyperlien : $e \in E_{SMA}$
La repartition logique ou physique de l'agent	Racine : ou n représente le nombre régions
Elements abstraits	Site : d_m ou m est le nombre de sites

TABLE 4.1 – Sémantique bigraphique des éléments d'un système

Définition 2. Un bigraphe G_{SMA} modélisant une architecture de système multi-agent sur une signature $\mathcal{K}$ prend la forme :

$$G_{SMA} = (V_{SMA}, E_{SMA}, ctrl_{SMA}, G^P_{SMA}, G^L_{SMA}) : I \to J$$

- V_{SMA} est un ensemble fini de nœuds d'agents, où chaque nœud définit un type d'agent. $V_{SMA} = SA \cup WA \cup AM$ est un ensemble de sous-ensembles, où :

  - SA est un ensemble fini d'agents simples qui ne sont ni autonomes ni intelligents (par exemple les agents SNMP (Simple Network Management Protocol) [Harrington 2002]). Ce type d'agents peut être utilisé pour spécifier des organisations SMA hiérarchisées (par exemple, SMA fédéré, telles que OAA [Martin 1999] ou [Genesereth 1994]) et également des organisations SMA de subsumption (par exemple ADEPT (Advanced Decision Environment for Process Task) [Norman 1997]). Un agent SA est généralement composé d'agents de type WA, et de modules de type AM (par exemple, des facilitateurs de communication, etc.), il prend formellement la forme suivante : $\forall x \in SA \exists x' \subset WA, \exists y \in AM / x = x' \cup y$

  - WA est un ensemble fini d'agents sophistiqués qui peuvent présenter une autonomie ou une intelligence. Exemple : croyance, désir, intention (BDI) agents [Rao 1995]. Nous utilisons ce type d'agents pour spécifier des organisations de SMA plates et modulaires. Par exemple, SMA basé sur la plate-forme JADE [Bellifemine 2005], Archon [Wittig 1994], OSACA [Scalabrin 1996] et bien d'autres organisations ouvertes. Un agent WA est composé d'un ensemble de modules d'agent AM.

- AM est un ensemble fini de constructions ou de modules logiciels spécialisés d'agents, implémentant des capacités (tels que le raisonnement, la planification, la prise de décision, les modules d'apprentissage, etc).

- E_{SMA} est un ensemble fini de hyper-liens où chaque lien relie différents agents et modules.
- $\mathcal{K}$ est une signature étendue définie par un ensemble d'éléments appelés contrôles. Pour chaque contrôle, la signature fournit un ensemble fini de ports et un ensemble fini d'attributs, où : $port(k)$ désigne les noms des ports de contrôle, et l'attribut (k) désigne les attributs d'un contrôle. Un attribut prend la forme $< nom, type, valeur >$: les attributs sont utiles pour la description architecturale de SMA. Il peuvent aider à préciser le paradigme organisationnel adopté et le protocole de communication, utilisé entre les agents et de nombreux autres détails architecturaux. Par exemple, $< Role, Groupe, identificateur >, < protocole, spcification, null >, etc.$ Sur la base de ces attributs et en utilisant des outils, nous pouvons effectuer des analyses et des manipulations sur les spécifications architecturales des SMA. Enfin, une signature $\mathcal{K}$ détermine également les contrôles atomiques et non atomiques actifs ou passifs.
- $ctrl_{SMA} : V_{SMA} \rightarrow \mathcal{K}$ est une fonction de transformation qui associe à chaque nœud $v \in V_{SMA}$ un contrôle $k \in \mathcal{K}$.
- G^P_{SMA} et G^L_{SMA} représentent respectivement le graphe de places et le graphe de liens, avec $\mathcal{P}_{SMA}\{(v \mid i) \mid i \in ar\,(ctrl_{SMA}\,(v))\}$ qui représente l'ensemble de ports du bigraphe.
- $I = \langle m, X \rangle$ et $J = \langle n, Y \rangle$ représentent respectivement l'interface interne et l'interface externe de G_{MAS} ; où m représente le nombre de sites, X est l'ensemble de noms internes, n représente le nombre de régions et enfin Y est l'ensemble des noms externes.

Dans notre approche, une architecture d'un système multi-agent est formellement représentée par un bigraphe G_{MAS}. Les agents qui constituent l'architecture SMA sont représentés par l'ensemble V_{MAS} désignant les différents types d'agents.
On distingue deux types d'agents SA et WA respectivement, les agents sophistiqués (atomiques) et les agents composites ou simples. Ces agents peuvent être soit (1) actifs et peuvent évoluer dynamiquement en adaptant leur structure interne en réponse à de nouveaux besoins. Ou (2) passifs et donc leur structure interne est définie lors de la conception. Au sein de notre architecture, les agents communiquent entre eux à travers des ports. Une interaction est représentée par un arc si elle est locale, ou par un hyper-arc si elle distante.

Notre but dans ce travail de thèse, est de fournir aux concepteurs et développeurs de système multi-agent une approche générique et formelle qui permet la conception effective de SMA au niveau architectural. Cette approche peut être raffinée et donc adaptée selon le besoin pour décrire un type d'architecture de SMA bien précis (voir chapitre 5). L'adoption des bigraphes en tant que base formelle dans notre approche est essentiellement motivée par le rapprochement sémantique que nous avons identifié entre les concepts des SMA et BRS, telles que la localisation (physique ou logique), la connectivité dynamique et la représentation hiérarchique des agents. Enfin, les bigraphes apportent le niveau d'abstraction requis pour une représentation sur plusieurs niveaux (micro-méso-macro) des architectures de SMA.

4.5 Description structurelle du modèle BRS-MAS

Dans notre approche BRS-MAS, un système multi-agent est structurellement spécifié suivant un ensemble de principes fondamentaux que nous organisons en deux niveaux d'abstraction : le niveau social (ou SMA) et le niveau interne (ou agent). En effet, notre modèle BRS-MAS permet à un concepteur SMA de spécifier une architecture de système multi-agent, sous deux perspectives de modélisation. Une perspective de haut en bas (top-down) où le SMA est spécifié au niveau social comme une configuration hiérarchique d'agents qui interagissent entre eux, indépendamment de leur structure interne. De plus, le concepteur peut également suivre une perspective de bas en haut (bottom-up), où la modélisation est centrée sur la structure interne de l'agent (c'est-à-dire les états mentaux et les capacités de l'agent).
Au niveau social, le concepteur SMA se concentre sur la définition de la structure sociale du SMA. Le modèle BRS-MAS fournit une notation explicite pour décrire la structure des architectures SMA. Par conséquent, l'aspect structurel au niveau social est exprimé à l'aide de deux graphes : Le graphe de places représentant la topologie organisationnelle en termes de concepts tels que les groupes et les rôles et le graphe de liens représentant les liens et les interconnexions entre ces entités. Ces deux graphes combinés forment le bigraphe G_{MAS}, représenté sur la figure 4.5, ce dernier décrit une organisation composée de six agents modélisés par les nœuds de AG1 à AG6, divisés physiquement en deux groupes représentés par la racine 0 et 1. Les différentes interactions entre ces agents (nœuds) sont modélisées par des liens (e1-e4). Dans le premier groupe (racine 0), nous avons une hiérarchie d'agents (nœuds) représentée par AG2 qui est composée de AG3 et AG4. Les sites désignés de 0 à 7 représentent des éléments qui ont été abstraits à ce niveau de la conception.

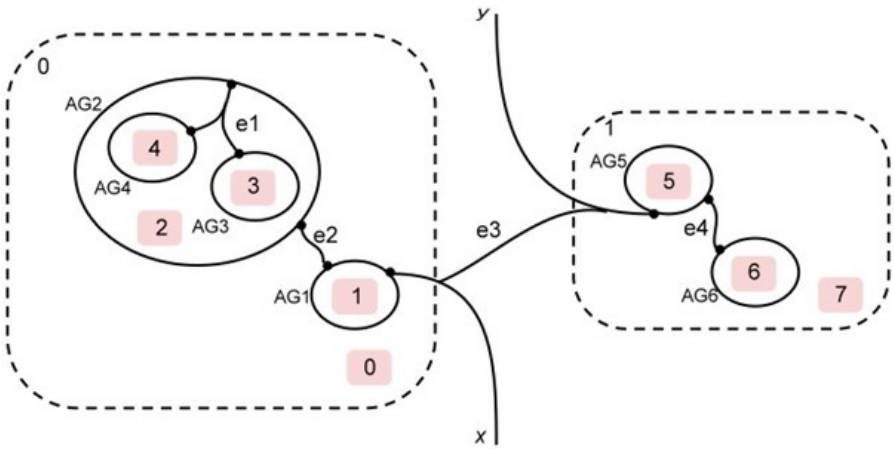


FIGURE 4.5 – Modèle bigraphique de la configuration BRS-MAS.

Suivant les deux perspectives (top-down ou bottom-up), nous pouvons construire un système, où les blocs élémentaire de construction (agents) peuvent avoir le même contrôle, et donc une structure interne identique, ou différents contrôles et par conséquent des structures internes différentes. Dans les deux cas, le modèle permet cette abstraction et hétérogénéité grâce à la notion d'interface abstraite, représentée par les interfaces entrantes et sortantes. En effet, les agents de notre système doivent avoir des interfaces structurellement identiques pour communiquer entre eux, et par conséquent, travailler ensemble pour résoudre un objectif local ou global.
Dans ce qui suit nous allons utiliser un exemple plus concret pour mieux illustrer la partie structurelle de notre approche BRS-MAS. Comme exemple, nous allons modéliser le système de gestion de véhicule intelligent et d'aide à la conduite décrit dans la section 4.2. Cependant nous n'allons pas nous attarder sur la structure interne des agents de notre SMA car nous verrons cet aspect plus en détail dans le prochain chapitre.

4.6 Exemple :

La partie structurelle du système de gestion du trafic de voiture intelligente, est modélisée par le bigraphe CAS représenté sur la figure 4.6. Comme vous pouvez le voir, la racine 0 représente l'environnement réel de notre système actuel. Elle contient un ensemble interconnectés de nœuds et plusieurs sites (carré gris). Le site (0-site), est utilisé pour représenter des parties du contexte qui ont été abstraites. Les nœuds de type R représentent les parcelles routières. Le nœud du type C modélise la voiture intelligente tandis que le site (5-Site) représente les composants internes du nœud C (tels que l'unité

de gestion de la vitesse, l'unité de gestion des notifications, le moteur, etc., ...) qui ont été abstraits à ce stade de la modélisation. Le reste des nœuds représente le conducteur, le panneau de signalisation etc ...
Dans cette section, nous démontrons la capacité de notre approche BRS à modéliser les systèmes contextuels comme indiqué dans la figure 4.6.

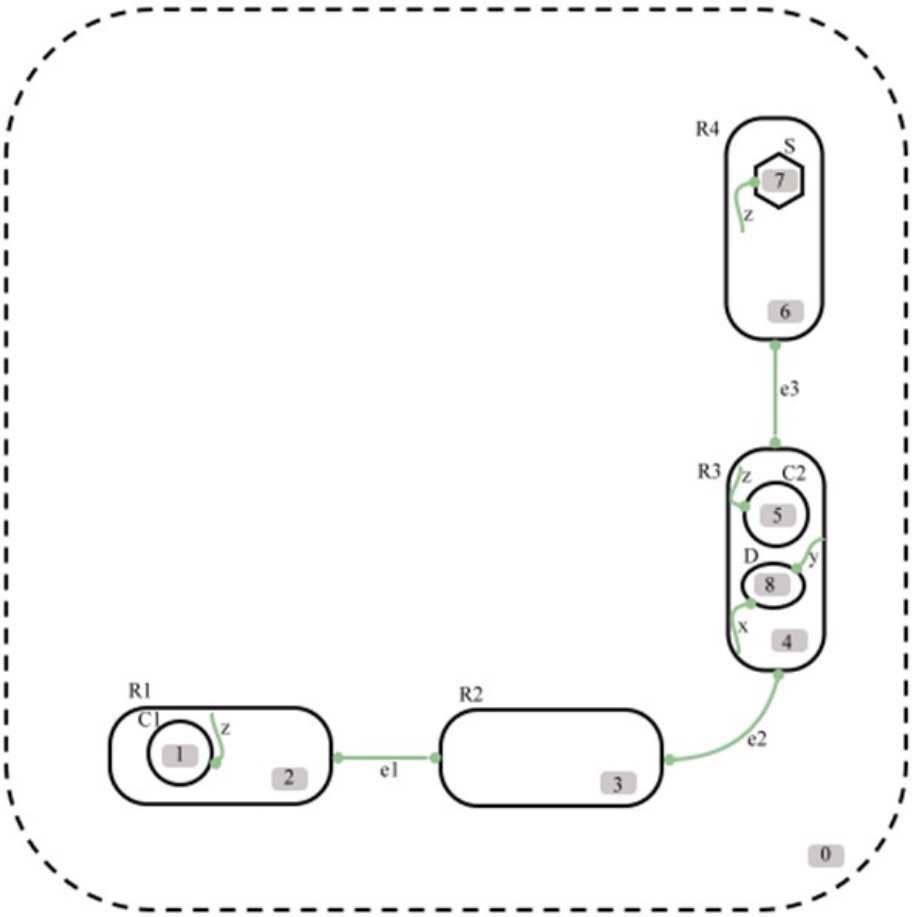


FIGURE 4.6 – Modèle bigraphique du système de gestion de véhicule intelligent et d'aide à la conduite .

L'expression algébrique du modèle est comme suit :

$$R1_{e1}.(C1_z.(d_1)|d_2)|R2_{e1e2}.(d_3)|$$
$$R3_{e2e3}.(C2_z.(D_{xy}.(d_8)|d_5)|d_4)|R4_{e3}.(S_z.(d_7)|d_6)|d_0$$

$CAS_0 = (\{R1, R2, R3, R4, C1, C2, D, S, AG1, AG2, AG3\},$
$\{e1, e2\}, ctrl_{BDI}, G^P_{BDI}, G^L_{BDI}) :$

$\langle 8, Z\rangle \rightarrow \langle 1, \varnothing\rangle \, Where :$
$K = \{R, C, S\}$

$port(C2) = \{InputInterface\}$,
$attribute(C2) = \{< ActualSpeed, double, 60 >\}$
$port(S) = \{trafficSignInputOutput, \}$,
$attribute(S) = \{< SpeedLimit, double, 50 >\}$
$port(R) = \{Leftjoint, Rightjoint\}$,
$attribute(R) = \{< ParcelState, string, good >, < ParcelLengh, double, 50 >\}$

4.7 Spécification de la reconfiguration architecturale BRS-MAS

Dans la section précédente, nous avons présenté les aspects structurels de notre modèle d'architecture BRS-MAS. Néanmoins, nous n'avons pas abordé les questions relatives à l'évolution et à la dynamique de l'architecture en terme de reconfiguration. La reconfiguration d'une architecture SMA est déclenchée en réponse à l'émergence de nouveaux changements au niveau de l'environnement ou au niveau des objectifs organisationnels. Ainsi, dans les deux cas, le SMA doit se reconfigurer en adaptant sa structure organisationnelle pour atteindre ses objectifs. Dans notre approche BRS, nous affichons un niveau élevé de reconfiguration. Par, la spécification de constructions architecturales flexibles pour les architectures SMA. Notre principale contribution repose sur la définition d'un ensemble de règles de réaction exprimant la reconfiguration d'une architecture SMA à deux niveaux distincts, à savoir le niveau social que nous verrons dans ce chapitre à travers l'exemple de motivation et le niveau agent que nous verrons plus en détail dans le chapitre 5. En effet, comme définie dans la section précédente, une configuration à un instant t obéit à la définition 2 :
$G_{SMA} = (V_{SMA}, E_{SMA}, ctrl_{SMA}, G^P_{SMA}, G^L_{SMA}) : I \rightarrow J$ est une configuration d'un SMA, alors que la reconfiguration de SMA vers SMA' est régie par des règles ou des meta-règles de réactions dont la forme générale est :
Meta-règle de réaction : $RL = (MAS, MAS', m' \rightarrow m)$

La reconfiguration d'un SMA au niveau social est représentée par l'évolution dynamique de l'architecture SMA en ajoutant ou en supprimant de nouveaux agents (ou un groupe d'agents), modifiant ainsi, drastiquement la topologie organisationnelle ou en effectuant une simple réorganisation de la topologie actuelle où les agents peuvent se déplacer d'un groupe à un autre. De plus, le modèle d'interaction organisationnelle peut changer dynamiquement en ajoutant ou en supprimant un ou plusieurs liens externes. De même, la reconfiguration interne de l'agent consiste à adapter son comportement interne en ajustant sa structure afin d'atteindre un objectif global ou local. Pour une meilleure compréhension, nous utilisons le bigraphe CAS_0 qui définit le

système de gestion de véhicule intelligent et d'aide à la conduite décrit dans la section 4.2.

4.8 Exemple

Dans ce qui suit, nous présentons un scénario décrivant la détection de panneaux de circulation qui illustre comment un système de voiture intelligente pourrait s'adapter en réponse aux changements d'un contexte environnemental. Scénario 1 : en premier lieu, le conducteur doit entrer dans la voiture. Puis, une fois à l'intérieur, il peut utiliser les composants de conduite (Accélérateur (Acc), frein (brake)) pour se déplacer entre les différentes parcelles de route et atteindre la destination.
Scénario 2 : la limite de vitesse peut changer, lorsqu'un panneau de limitation de vitesse est détecté par les capteurs de la voiture, comme indiqué sur la figure 8, si le panneau détecté indique une limitation de vitesse inferieure à la vitesse actuelle du véhicule, l'unité de gestion des notifications (NMU) de la voiture avertit le conducteur par une alerte sonore mais aussi visuelle en affichant un message-texte qui affiche la limite de vitesse indiquée par le panneau de limitation de vitesse ; simultanément, les composants de l'unité de gestion de la vitesse (SMU) ajustent automatiquement la limite de vitesse. Formellement, les scénarios susmentionnés sont modélisés par une suite de règles de réaction (voir tableau 6).

SEQUENCE DE RÈGLES DE RÉACTION

À partir du bigraphe CAS_0 représenté graphiquement dans la figure 8, qui représente l'état initial du système de gestion de véhicule intelligent et d'aide à la conduite. Sur le plan structurel, l'état final décrit dans le scénario (1) est représenté sur la figure 9. Cet état, représente le résultat de l'application de la séquence de règles de réaction SC_1 donnée par :

$$SC_1 = CAS_0 \overset{R11}{\rightarrow} CAS_1 \overset{R1}{\rightarrow} CAS_2 \overset{R7}{\rightarrow} CAS_3 \overset{R11}{\rightarrow} CAS_4 \overset{R8}{\rightarrow} CAS_5 \overset{R3}{\rightarrow} CAS_6$$

En effet, le plan d'exécution SC_1 représente les différents états du système décrit dans le scénario 1 du système de gestion du trafic. À partir du bigraphe CAS_0, la règle de réaction RL11 est déclenchée en raison de l'émergence du désir du conducteur d'entrer dans la voiture. Par conséquent, pour satisfaire ce désir, il faut que les règles de réactions RL1, RL7 soient déclenchées. Une fois exécutées, on peut dire que le conducteur a atteint son but avec succès (voir figure 10. (a,b)). À ce stade, le conducteur peut prendre contrôle du véhicule et donc se déplacer vers sa destination. Les règles de réaction RL8,

RL 1 Règle de réaction entrer dans la voiture (macro)	$R.(C.(d_1)\|D.(d_2)\|d_0) \rightarrow R.(C.(D.(d_2)\|d_1)\|d_0)$
RL 2 Règle de réaction sortir de la voiture	$R.(C.(D.(d_21)\|d_1)\|d_0) \rightarrow R.(C.(d_1)\|D.(d_2)\|d_0)$
RL 3 Règle de réaction avancer	$R.(C.(D.(d_3)\|d_2)\|d_0)\|R'.(d_1) \rightarrow R.(d_0)\|R'.(C.(D.(d_3)\|d_2)\|d_1)$
RL 4 Règle de réaction reculer	$R.(d_0)\|R'.(C.(D.(d_3)\|d_2)\|d_1)\|R.(C.(D.(d_3)\|d_2)\|d_0)\|R'.(d_1)$
RL5 Règle de réaction détection des signes de circulation	$R.(C_z.(D.(d_3)\|d_1)\|S_z.(d_2)\|d_0) \rightarrow R.(C_(ze1).(D.(d_3)\|d_1)\|S_(ze1).(d_2)\|d_0)$
RL6 Règle de réaction déconnexion du panneaux de signalisation	$R.(C_(ze1).(D.(d_3)\|d_1)\|S_(ze1).(d_2)\|d_0) \rightarrow R.(C_z.(D.(d_3)\|d_1)\|S_z.(d_2)\|d_0)$
RL7 Règle de réaction entrer dans la voiture (méso)	$D_{xy}.(d_1)\|C.(Acc_{ye2}.(d_2)\|Brake_{ye2}.(d_3)\|SMU_{ze1e2e3}.(d_4)\|$ $Engine_e1.(d_5)\|NMU_{xe3}.(d_6)\|d_0)$ $\rightarrow$ $C.(D_{xy}.(d_1)\|Acc_{ye2}.(d_2)\|Brake_{ye2}.(d_3)\|SMU_{ze1e2e3}.(d_4)\|$ $Engine_e1.(d_5)\|NMU_{xe3}.(d_6)\|d_0)$
RL8 Règle de réaction initier une accélération	$C.(D_{xy}.(d_1)\|Acc_(ye1).(d_2)\|Brake_{ye2}.(d_3)\|SMU_{ze1e2e3}.(d_4)\|$ $Engine_e1.(d_5)\|NMU_{xe3}.(d_6)\|d_0)$ $\rightarrow$ $C.(D_{xye4}.(d_1)\|Acc_{ye1e4}.(d_2)\|Brake_{ye2}.(d_3)\|SMU_{ze1e2e3}.(d_4)\|$ $Engine_e1.(d_5)\|NMU_{xe3}.(d_6)\|d_0)$
RL9 Règle de réaction freiner	$C.(D_{xye4}.(d_1)\|Acc_{ye1e4}.(d_2)\|Brake_{ye2}.(d_3)\|SMU_{ze1e2e3}.(d_4)\|$ $Engine_e1.(d_5)\|NMU_{xe3}.(d_6)\|d_0)$ $\rightarrow$ $C.(D_{xye5}.(d_1)\|Acc_{ye1}.(d_2)\|Brake_{ye2e5}.(d_3)\|SMU_{ze1e2e3}.(d_4)\|$ $Engine_e1.(d_5)\|NMU_{xe3}.(d_6)\|d_0)$
RL10 Règle de réaction notifier le conducteur	$C.(D_{xye4}.(d_1)\|Acc_{ye1e4}.(d_2)\|Brake_{ye2}.(d_3)\|SMU_{ze1e2e3}.(d_4)\|$ $Engine_e1.(d_5)\|NMU_{xe3}.(d_6)\|d_0)\|S_z$ $\rightarrow$ $C.(D_{xye4e5}.(d_1)\|Acc_{ye1e4}.(d_2)\|Brake_{ye2}.(d_3)\|SMU_{ze1e2e3}.(d_4)\|$ $Engine_e1.(d_5)\|NMU_{xe3e5}.(d_6)\|d_0)\|S_(z)$
RL11 Règle de réaction résolution d'un but interne	$AG_{xy}.(B_e1.(K\|d_2)\|G.(D1\|d_4)\|I_e1.(P\|d_3)\|d_1)\|d_0$ $\rightarrow$ $AG_{xy}.(B_e1.(K\|K1\|d_2)\|G.(D1_e2\|d_4)\|I_e1.(P_e2\|d_3)\|d_1)\|d_0$

TABLE 4.2 – Règles de réaction modélisant les interactions front-end/back-end

RL3 sont exécutées pour permettre un tel comportement. Le bigraphe CAS_6 (voir figure 9 (c)) représente le dernier état du plan d'exécution SC_1 et donc le résultat de la reconfiguration du système.

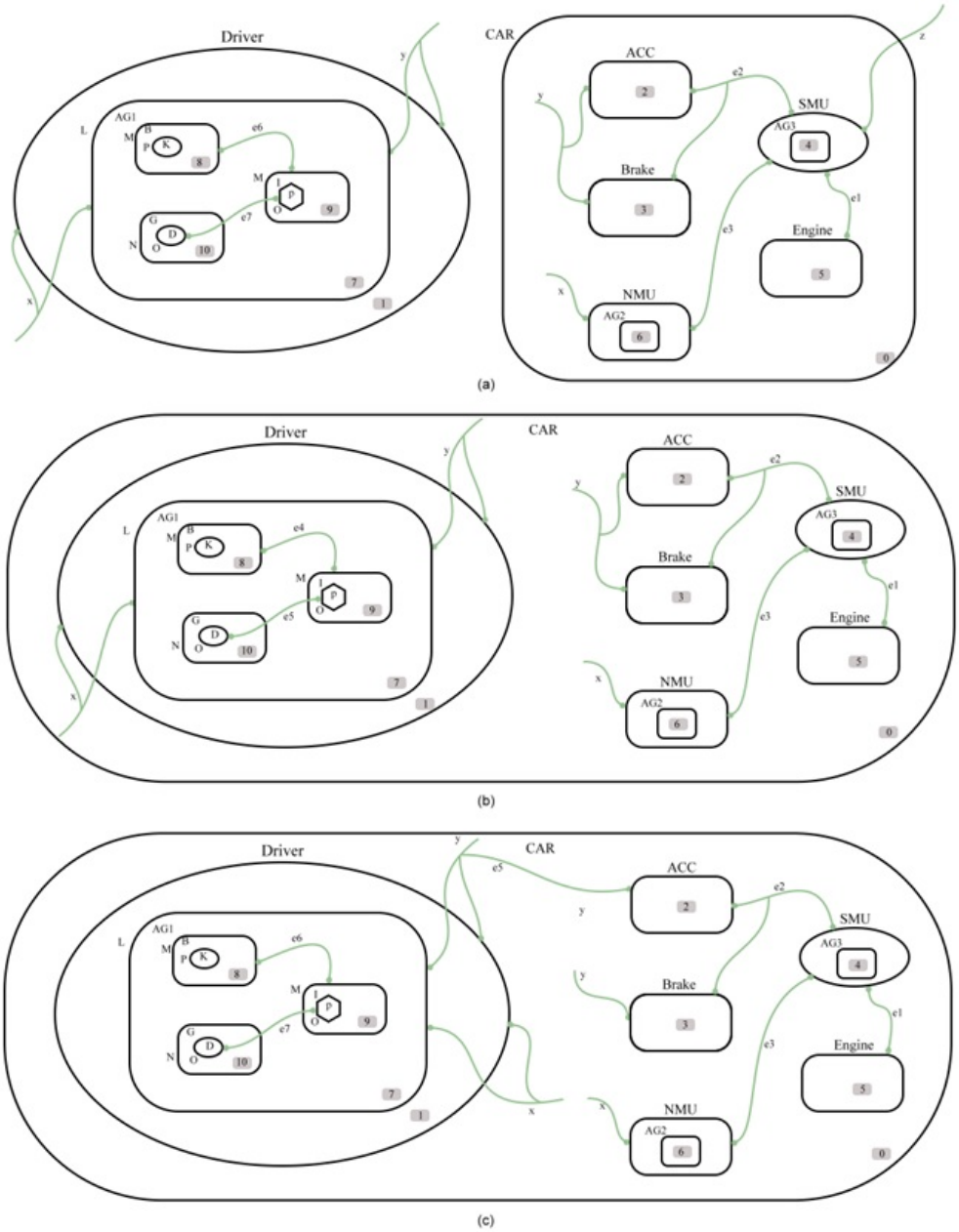


FIGURE 4.7 – Système de gestion de véhicule intelligent et d'aide à la conduite reconfiguration scénario 1

Les reconfigurations du système de gestion du véhicule intelligent et d'aide à la conduite décrites dans le scénario 2 sont données par le plan d'exécution

SC_2.

$$SC_2 = CAS_6 \overset{R5}{\rightarrow} CAS_7 \overset{R11}{\rightarrow} CAS_8 \overset{R10}{\rightarrow} CAS_9 \overset{R9}{\rightarrow} CAS_10$$

Nous constatons que l'état initial du scénario 2 est l'état final du scénario 1 représenté par le bigraphe CAS_6 (figure 9 (c)). En effet, vu que l'état actuel de la voiture est en mouvement (déplacement d'une parcelle à l'autre), la limitation de vitesse peut changer. Cela est régis par la présence d'un panneau de limitation de vitesse dans la parcelle de route accédée. Ainsi, la voiture par son interface z, détecte automatiquement le panneau routier et analyse ses informations. Ceci est réalisé, une fois l'exécution de la règle de réaction RL5 terminée. Dans le cas où la vitesse actuelle du véhicule est supérieure à la limitation indiquée par le panneau de signalisation, le nœud SMU à travers le lien e5 envoie une notification au pilote en passant par le nœud NMU. Simultanément, la vitesse de la voiture est adaptée automatiquement. Les règles de réactions qui permettent de modéliser cette reconfiguration sont les règles RL11, RL10, RL9. La figure 10 représente l'état final du scénario 2.

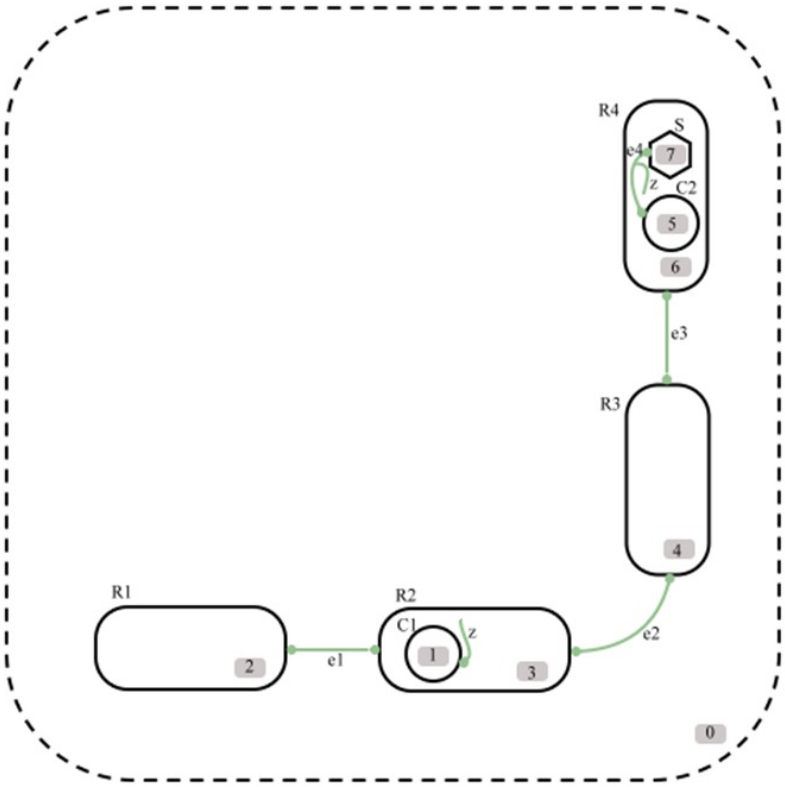


FIGURE 4.8 – Système de gestion de véhicule intelligent et d'aide à la conduite reconfiguration scénario 2

4.9 Conclusion

Dans ce chapitre, nous avons commencé en premier lieu, par identifier les concepts clés lies à la modélisation de système multi-agent de manière générale, ensuite nous nous sommes intéressés aux architectures logicielles et à leur possible apport dans la modélisation de SMA. L'apport principal de ce chapitre est la proposition d'une approche formelle qui aide à réduire la complexité de la modélisation et de la conception des architectures des systèmes multi-agents. Nous nous sommes basés sur les bigraphes comme cadre formel car nous avons jugé qu'à priori, les concepts des systèmes multi-agents et des bigraphes sont compatibles et par conséquent représentables en utilisant les bigraphes. Par la présentation du modèle BRS-MAS nous avons pu valider ces prioris. En effet, le modèle bigraphique présenté est assez générique et modulaire, il permet d'exprimer tous les éléments d'une architecture SMA. En conséquence, plusieurs types d'architectures de SMA peuvent être représentés (par exemple : les architectures modulaires, hiérarchiques, etc.). De plus, elle offre au concepteur la liberté de construire des SMA en suivant une approche top-down ou bottom-up.
Au niveau structurel, une architecture SMA est modélisée sur plusieurs niveaux d'abstraction. On distingue un niveau social qui représente la configuration de notre architecture SMA au niveau macro et le niveau agent (individuel) qui représente l'architecture interne de l'agent (le niveau micro). La reconfiguration de ce type d'architecture est, quant à elle, exprimée au moyen de règles de réactions. Dans le chapitre suivant, nous nous intéressons à une application de notre approche BRS-MAS sur un type bien déterminé d'architectures SMA à base d'agents BDI.

Chapitre 5

Une architecture basée BRS pour la modélisation de systèmes multi-agent BDI

Sommaire

5.1 Introduction

Au cours des dernières années, les systèmes logiciels ont tendance à être de plus en plus distribués, ouverts et concurrents. Cette évolution de l'informatique a changé la façon de penser, mais aussi celle de la conception de tels systèmes. Le paradigme multi-agents (SMA) est particulièrement adéquat pour développer ce type de systèmes.
En effet, un SMA peut être défini comme un ensemble d'entités faiblement couplées appelées agents, ces derniers communiquent entre eux de manière asynchrone afin d'atteindre un objectif local (interne à l'agent) ou global (au niveau SMA). Un agent représente l'entité de premier ordre dans un SMA. Il est considéré comme un système informatique autonome qui offre un haut niveau d'abstraction et des mécanismes qui permettent la prise en charge de concepts tels que le raisonnement, la représentation des connaissances, la communication, la perception, les engagements, les objectifs, croyances et intentions. De plus, de tels systèmes doivent souvent opérer dans des environnements dynamiques et faire face aux défis de la gestion constante de l'évolution

de ce dernier ; Donc ils doivent être flexibles, robustes et capables de s'adapter à leur environnement [Wooldridge 2009]. Par conséquent, le développement de systèmes multi-agents est une tâche d'ingénierie complexe. Pour relever ce défi, il est nécessaire d'élever le niveau d'abstraction du système pour mieux gérer la complexité et faciliter le raisonnement. L'architecture logiciel, vise à fournir des descriptions de haut niveau des systèmes, représentant non seulement leurs structures logiques, mais aussi beaucoup d'autres aspects fonctionnels et non fonctionnels (par exemple, le comportement, la sécurité, etc). Ainsi qu'un ensemble de propriétés et de contraintes que le système doit respecter. En effet, l'architecture logicielle offre une vision globale et un niveau élevé de la structure et de l'organisation d'un système. En outre, l'architecture logicielle joue un rôle clé en tant que pivot entre les exigences d'un système et sa mise en œuvre. À cet effet, les langages de descriptions d'architectures (ADL) ont été adoptés comme des outils formels pour décrire l'architecture de logiciel à haut niveau d'abstraction. Dans la littérature, plusieurs ADL tels que Darwin, Rapide, Dynamic-Wright [Allen 1998] et π-ADL [Oquendo 2004] ont été proposés pour la représentation et l'analyse d'architectures logicielles. Toutefois ces ADL ne sont pas adaptés à la représentation exhaustive des caractéristiques des architectures SMA, tels que le raisonnement, la communication(interaction), la perception et la reconfiguration dynamique.
Dans ce chapitre, les systèmes réactifs bigraphiques (BRS) [Milner 2009] sont adoptés comme cadre sémantique pour formaliser les architectures de SMA. Plus particulièrement, pour formaliser les architectures SMA basées sur des agents de type BDI (croyance-désire-intention). En plus, de leur rigueur et de leur aspect graphique, les BRS sont capables de représenter la localité et la connectivité, qui constituent les principaux concepts des architectures SMA. Un système réactif bigraphique se compose d'une catégorie de bigraphes et d'un ensemble de règles de réactions qui lui permettent de se reconfigurer. Par conséquent, les BRS sont très appropriés pour formaliser les aspects fondamentaux des architectures de SMA et leur reconfiguration. Dans ce qui suit, nous utilisons les BRS comme méthode formelle pour proposer un modèle nommé BDI-MAS. Ce modèle, est une application de notre approche générique BRS-MAS (présentée dans le chapitre 4) sur des architectures SMA qui contiennent des agents de type BDI. Le modèle proposé dans ce chapitre permet la spécification statique et dynamique, au niveau individuel (agent) et niveau social (SMA), ainsi que les relations et les contraintes des architectures BDI-MAS.

5.2 Principe de modélisation

Au niveau de la section 4.3 nous avons présenté les concepts nécessaires pour la modélisation des systèmes multi-agents de manière générale. En effet, nous nous sommes basés sur des définitions génériques et abstraites de ce qu'est un agent. De plus, nous ne nous sommes pas vraiment attardés sur la structure interne de ces agents vu qu'il existe une pléthore d'architectures différentes. Néanmoins ces dernières rentrent dans l'une des trois catégories d'agents (réactifs, cognitifs ou hybrides). Dans cette section nous allons exposer brièvement le concept d'agent BDI qui représente le bloc de base du modèle BDI-MAS.

5.2.1 Éléments architecturaux d'un agent BDI

Le modèle BDI (croyance, désir et intention) [Rao 1995] est l'approche la plus couramment utilisée pour la représentation de l'état interne des agents. Le modèle est basé sur une théorie du comportement humain qui est largement connue. Celle-ci a été développée par le philosophe Michael Bratman. Aucune implémentation spécifique n'est prescrite. Le modèle peut être décrit de différentes manières. En effet, un certain nombre d'implémentations différentes ont été développées. En outre, le modèle BDI a été utilisé pour construire un certain nombre d'applications du monde réel. Les composantes essentielles d'un agent BDI sont :

- **Les croyances :** des informations que l'agent possède sur l'environnement et sur d'autres agents qui existent dans le même environnement,
- **Les désirs :** des représentation les états de l'environnement, et parfois de lui-même, que l'agent aimerait voir réaliser.
- **Les intentions :** des désirs que l'agent a décidé d'accomplir ou les actions qu'il a décidé de faire pour accomplir ses désirs.

Un agent BDI met à jour ses croyances à partir des informations qu'il perçoit depuis son environnement, choisit entre les options qui lui sont offertes, détermine de nouvelles intentions et les réalise. Le concept d'intention représente la composante principale de cette approche car il permet de relier les objectifs, croyances et engagements de l'agent avec une théorie d'action. L'algorithme suivant décrit le processus de raisonnement d'agent BDI [Wooldridge 2000b].
Soient $B_0, D_0 et I_0$ les croyances, désirs et intentions initiales de l'agent.

Algorithme de contrôle d'agent BDI
1 B =B0

2 D =D0
3 I = I0
4 répéter
4.1 obtenir nouvelles perceptions p
4.2 B = revc(B, p)
4.3 I = options(D, I)
4.4 D = des(B, D ,I)
4.5 I = filtre(B, D, I)
4.6 PE = plan(B, I)
4.7 exécuter(PE)
jusqu'à ce que l'agent soit arrêté
fin

5.3 Sémantique basée BRS d'architectures de systèmes multi-agents BDI

En partant des caractéristiques du modèle BDI et celles des systèmes réactifs bigraphiques proposés dans la littérature, nous avons travaillé à trouver un modèle formel capable de spécifier les architectures de type BDI-MAS. Par conséquent, cette section propose une approche formelle basée sur les bigraphes dédiés à la conceptualisation d'architecture BDI-MAS.
Les bigraphes représentent un outil sophistiqué pour formaliser les éléments d'architecture BDI-MAS. Ces derniers offrent un support graphique et un langage de termes algébriques pour modéliser à la fois les aspects statiques et dynamiques des architectures BDI-MAS. Néanmoins, le modèle prend seulement en charge les éléments relatifs à la modélisation d'architecture BDI-MAS (structure interne des composants et relations entre composants) et ne considèrent pas la sémantique ou les heuristiques fonctionnelles pour lesquelles, par exemple, un plan est choisi parmi d'autres.
À haut niveau d'abstraction, un système multi-agent est considéré comme un ensemble d'entités informatiques (un ensemble d'agents) répartis sur plusieurs sites, souvent appelés nœuds. Le tableau 5.1 résume les éléments fondamentaux intervenant dans une architecture BDI-MAS.

Les éléments d'architecture d'un SMA BDI	Sémantique bigraphique
Agents, module de Croyances, module de Désirs, module d'Intention, plans.	Nœud
Location physique ou logique des agents	Racine

Les différents types de liens entre agents et modules	Lien /Hyper-lien
Éléments abstraits	Sites

TABLE 5.1 – Sémantique bigraphique des éléments d'un système multi-agent BDI

5.3.1 Description structurelle du modèle BDI-MAS

BDI-MAS, modélise l'architecture sur deux niveaux d'abstraction : le niveau de l'agent et le niveau social. Le premier décrit la structure interne et l'état de l'agent (c'est-à-dire les éléments de base de l'SMA), tandis que le second décrit l'assemblage et l'interaction entre les agents qui composent l'architecture SMA.
Niveau agent : La figure 5.1 décrit un agent BDI en présentant sa structure interne. Généralement, un agent est situé dans une racine représentant l'emplacement physique / logique de l'agent. Chaque agent (dénoté par AG) se compose de trois principaux nœuds, qui à leur tour contiennent d'autres nœuds qui les structurent. Dans ce qui suit, nous allons examiner de plus près les nœuds qui composent l'agent AG1 :

Les croyances (le nœud B) représentent la vision que l'agent a du monde. Elles correspondent aux informations que l'agent a sur son environnement et sur les autres agents qui forment l'architecture SMA. Ces croyances, peuvent être incorrectes, incomplètes ou incertaines. Les croyances peuvent changer (1) au fur et à mesure que l'agent, par sa capacité de perception sur l'environnement ou par l'interaction avec d'autres agents, recueille plus d'informations. Cela est réalisé à travers les interfaces entrantes et sortantes X et Y. (2) À la suite de l'exécution d'un plan, la mise à jour des croyances dans ce cas est réalisée à travers la liaison $e1$ qui relie les ports respectifs des nœuds B et I. Le nœud B est un nœud composite, il contient des nœuds atomiques qui représentent les connaissances qui constituent l'ensemble des croyances de l'agent AG (désigné par K).

Les désirs (nœud G) ou bien les objectifs de l'agent, représentent les états de l'environnement, et parfois de lui-même, que l'agent aimerait voir réaliser. Ces objectifs peuvent être internes ou externes (dans le cadre d'une collaboration entre agents, SMA) à l'agent. Un agent peut avoir des désirs. Dans ce cas, il doit en choisir un sous-ensemble qui est cohérent. Ce sous-ensemble cohérent de désirs est aligné avec les objectifs de l'agent. Ces désirs sont représentés comme des plans non encore instanciés pour permettre à

l'agent d'atteindre ses objectifs. Un désir est représenté par un nœud (désigné par D1, D2) qui contient un port par lequel il peut choisir le plan à instancier et donc à exécuter pour satisfaire le désir en question.

Les intentions (nœud I) d'un agent sont les désirs qu'il a décidé d'exécuter ou les actions qu'il a décidé d'accomplir ses désirs. Même si tous les désirs d'un agent sont cohérents, il peut ne pas être capable de tous les accomplir à la fois. Il existe des piles de plans instanciés où chaque plan (noté par P) est dédié à satisfaire un seul désir.

En outre, AG est doté de deux ports attachés à une interface entrante x et une interface sortante y. Ces interfaces permettent à un agent d'interagir avec les autres agents du SMA mais aussi avec son environnement (entités physiques du contexte). Elles sont utilisées pour envoyer et recevoir des messages entre agents. La présence du site 0, site 1, site 2, site3, site 4 signifie que le modèle prend en compte le déploiement dynamique de nouveaux agents, connaissances, plans et objectifs au sein de l'architecture.

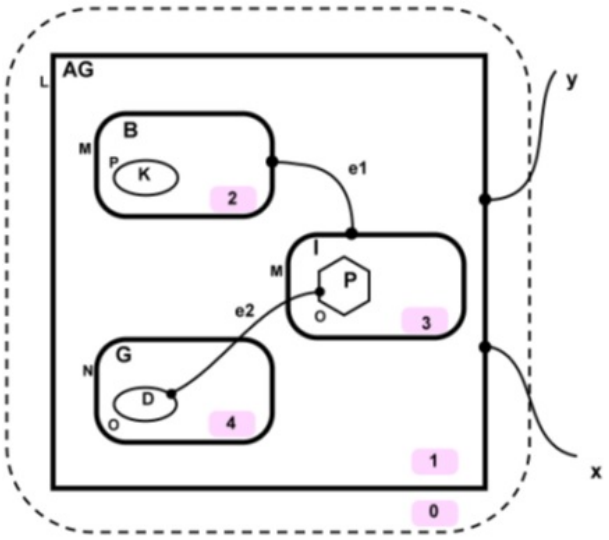


FIGURE 5.1 – Agent BDI

La spécification algébrique du bigraphe BDI-MAS présenté dans la figure 5.1 est la suivante :

$$AG_{xy}.(B_{e1}.(K|d_2)|G.(D1_{e2}|d_4)|I_{e1}.(P_{e2}|d_3)|d_0)$$

G_{BDI} est le bigraphe modélisant un agent BDI sur la signature K, il prend la forme :

$G_{BDI} = (\{AG, B, G, I, P, D, K\}, \{e1, e2\}, ctrl_{BDI}, G^P_{BDI}, G^L_{BDI}) : \langle 5, X\rangle \to \langle 1, Y\rangle$

La signature associée à un bigraphe BDI-MAS est la suivante : $K = \{L : (2, actif), M : (1, actif), N : (0, actif), O : (1, atomique), P : (0, atomique)\}$, L, M, N, O et P représente les contrôles associés aux différents nœuds. Les différents types de nœuds utilisés et leurs contrôles associés sont résumés dans le tableau 5.2.

Nœud	Contrôle	Attribut	Arité	Sens
AG	L	Actif	2	Agent
B	M	Actif	1	Module de Croyance
G	N	Actif	0	Module de but
I	M	Actif	1	Module d'intention
P	O	Atomique	1	Plan
D	O	Atomique	1	Désir
K	P	Atomique	0	Connaissance

TABLE 5.2 – Types de nœuds de l'architecture BDI-MAS

Niveau social : l'architecture d'un SMA est vue comme un ensemble interconnecté d'agents qui interagissent. Le modèle présenté offre les notations nécessaires pour décrire la structure d'un SMA en termes de configurations hiérarchiques de composants (agents). Le modèle fournit une base explicite et commune pour décrire les architectures de SMA (Voir figure 5.2). Chaque agent effectue une partie de la tâche globale et il interagit avec les autres agents du SMA pour combiner leurs comportements. ces interactions peuvent être assez complexes où chaque agent peut initier une communication, et répondre aux messages d'autres agents qui y participent pour atteindre les objectifs globaux du système.

La spécification algébrique du bigraphe BDI-MAS présenté dans la figure 5.2 est la suivante :

$$xy/AG_{xe3ye6}.(B_{e1}.(K|d2)|G.(D1_{e2}|d4)|I_{e1}.(P_{e2}|d3)|d1)|d0||$$
$$AG1_{xe3ye6}.(B1_{e4}.(K1|K2|d7)|G1.(D2_{e5}|d9)|I1_{e4}.(P1_{e5}|d8)|d6)|d5$$

L'interaction est le moyen de mise en relation dynamique d'agents dans le système. Nous distinguons plusieurs façons d'atteindre ce couplage :

- Par interaction indirecte non gérée portée par l'environnement ;

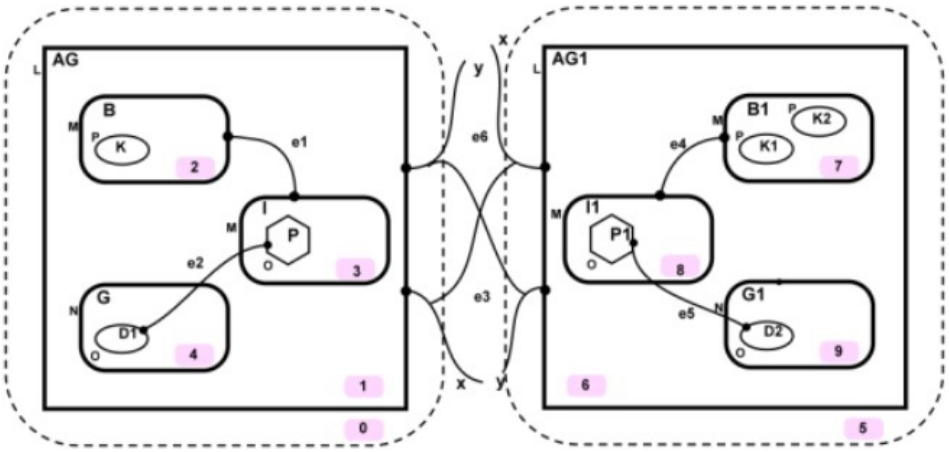


FIGURE 5.2 – Configuration bigraphique du Modèle BDI-MAS

- Par des interactions directes, ponctuelles et gérées ;

L'interaction directe est basée sur le passage de messages entre agents et l'échange d'informations via deux types de ports reliés à deux interfaces X, Y qui permettent aux agents d'échanger des informations sur leur environnement et l'avancement des tâches. De même, la négociation entre agents permet aux agents de décider ensemble d'une répartition des tâches (comme le réseau contractuel [Sandholm 1993] basé sur le processus de négociation). Les conséquences d'une interaction directe sont décidées par les lois comportementales des agents. Notre agent BDI-MAS a un haut niveau d'abstraction grâce à l'utilisation d'interfaces. Ainsi, il peut s'exprimer au moyen d'un langage élaboré et des protocoles de communication (tels que KQML [Finin 1994], FIPA ACL [Fipa 2002]).
Les interactions indirectes sont déterminées par les lois de l'environnement. Ainsi, un agent en effectuant une interaction indirecte ne sait pas avec quels autres agents il interagit, car il ne sait pas quel agent devra modifier son comportement en observant les changements dans l'environnement.
Le schéma général d'une interaction indirecte est le suivant :

- Un agent AG émet une action via l'interface de sortie Y qui est responsable de l'interaction avec l'environnement.
- Cette action modifie l'environnement,
- Ce changement dans l'environnement a pour conséquence, un changement dans les perceptions des agents voisins le cas pour l'agent AG1 à travers son interface d'entrée X. A son tour AG1 émet une action via son interface de sortie, à la suite de ses nouvelles perceptions.

- Il y a interaction (au sens de couplage entre agents) puisque le comportement de l'agent AG1 est basé sur l'action précédente effectuée par l'agent AG. Ce type d'interaction est à la base du mécanisme de stigmergie observé dans les systèmes naturels.

5.4 Spécification de la reconfiguration des architectures BRS-MAS

Bien que les bigraphes suffisent à spécifier formellement les éléments architecturaux de type BDI-MAS ainsi que leurs schémas d'interaction. La dynamique de l'architecture BDI-MAS est formalisée en utilisant des règles de réaction exprimant les changements de forme en termes de reconfigurations tout en préservant les contraintes architecturales. Dans cette sous-section, nous abordons la reconfiguration des architectures MAS au niveau social de manière générale en mettant l'accent sur les agents BDI au niveau de l'agent. Ensuite, nous donnerons des exemples de règles de réaction qui modélisent le comportement interne et externe ainsi que la reconfiguration de l'architecture BDI-MAS. Le tableau 5.3 illustre le modèle comportemental de l'architecture BDI-MAS au niveau social et agent.

Ainsi, une architecture SMA est représentée par un bigraphe, ses reconfigurations possibles au niveau social et agent par des règles de réactions élémentaires, ce qui forme un système réactif bigraphique. En outre, les méta-règles de réaction présentées dans le tableau 5.3 peuvent être utilisées pour exprimer différents scénarios d'évolution complexe par application successive de règles. Pour une meilleure compréhension, nous illustrerons dans le reste de cette section la représentation graphique de certaines règles de réactions citées ci-dessus.

Exemple RL1 : Création d'un nouveau groupe
En réponse à l'émergence de nouveaux objectifs au niveau social, la règle de réaction présentée dans la figure 5.3 illustre le processus de création d'un nouveau groupe. Sur la partie gauche de la règle (redex), la configuration initiale de l'architecture SMA est représentée par un groupe (racine 0), à droite de la règle (reactum) un nouveau groupe représenté par une racine 1 est créé. Cela peut s'expliquer par le déploiement d'un nombre important d'agents dans l'architecture SMA. Les sites 0 et 1 représentent des nœuds d'agents abstraits à ce niveau (structure interne du groupe).

Exemple RL3 : Déploiement d'un nouvel agent

SMA	BRS
Configuration de SMA	$G_{BDI} = (V_{BDI}, E_{BDI}, ctrl_{BDI}, G^P_{BDI}, G^L_{BDI}) : X \to Y$
Reconfiguration de SMA a SMA'	Meta regles de reaction : $RL = (MAS, MAS', m' \to m)$
Niveau Social	Création d'un nouveau groupe
	$R1 \stackrel{\text{def}}{=} d \to d \| d'$
	Suppression d'un groupe
	$R2 \stackrel{\text{def}}{=} d \| d' \to d$
	Déploiement d'un nouvel agent
	$R3 \stackrel{\text{def}}{=} d \to AG.(d')\|d$
	Migration d'un agent
	$R4 \stackrel{\text{def}}{=} AG.(d)\|d' \| d'' \to d'\|AG.(d)\|d''$
	Suppression d'agent
	$R5 \stackrel{\text{def}}{=} AG.(d)\|d' \to d'$
	Création de lien agent (local)
	$R6 \stackrel{\text{def}}{=} AG_{xy}.(d)\|AG'_{xy}.(d')\|d'' \to AG_{xye}.(d)\|AG'_{xye}.(d')\|d''$
	Création de lien agent (distant)
	$R7 \stackrel{\text{def}}{=} AG_{xy}.(d)\|d'' \| AG'_{xy}.(d')\|d''' \to AG_{xye}.(d)\|d'' \| AG'_{xye}.(d')\|d'''$
	Suppression de lien agent
	$R8 \stackrel{\text{def}}{=} AG_{xy}.(d) \| AG'_{xy}.(d') \to AG_{xye}.(d) \| AG'_{xye}.(d')$
Niveau Agent	Nouvelles connaissances d'agent
	$R9 \stackrel{\text{def}}{=} d \to B.(K\|d)\|d'' \to B.(K\|K'\|d)\|d''$
	Émergence d'un nouveau désir
	$R10 \stackrel{\text{def}}{=} G.(d)\|d' \to G.(D\|d)\|d'$
	Création d'un plan
	$R11 \stackrel{\text{def}}{=} I.(d)\|d' \to G.(P\|d)\|d'$
	Délégation d'un objectif (local)
	$R12 \stackrel{\text{def}}{=} G.(D\|d)\|G'.(d')\|d'' \to G.(D\|d)\|G'.(D\|d')\|d''$
	Délégation d'un objectif (distant)
	$R13 \stackrel{\text{def}}{=} G.(D\|d) \| G'.(d')\|d'' \to G.(D\|d) \| G'.(D\|d')\|d''$
	Choix d'un plan
	$R14 \stackrel{\text{def}}{=} G.(D\|d)\|I(P\|d')\|d'' \to G.(D_e\|d)\|I(P_e\|d')\|d''$

TABLE 5.3 – Règles de réaction modélisant les reconfigurations de l'architecture BDI-MAS

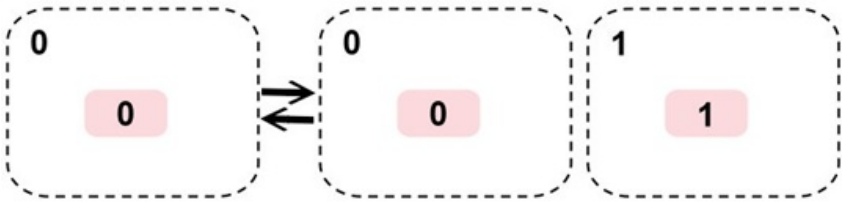


FIGURE 5.3 – Règle de réaction pour la création d'un nouveau groupe

La règle de réaction dans la figure 5.4 décrit la reconfiguration d'un SMA au niveau architectural. Cette reconfiguration est accomplie par le déploiement d'un nouvel agent dans la configuration actuelle (groupe) modélisée par la racine 0 contenant le site 0. A noter qu'au niveau du reactum de la règle (à droite), un nouvel agent représenté par le nœud AG est créé à l'intérieur de la racine 0. Le site 0 représente les nœuds d'agents absents abstraits, tandis que le site 1 exprime la structure interne de l'agent.

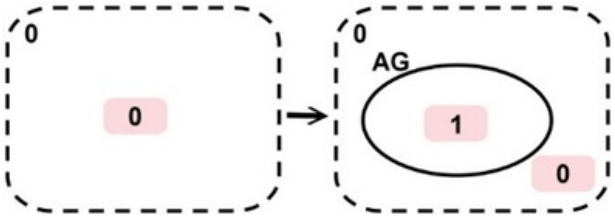


FIGURE 5.4 – Règle de réaction pour le déploiement d'un nouvel agent

Exemple RL 4 : Migration d'un agent
Un agent peut manifester le désir de migrer d'un groupe à un autre. Ce désir peut résulter du fait que les objectifs actuels de l'agent sont devenus contradictoires avec ceux des agents dans le même groupe. La règle de réaction pour la migration d'un agent représentée sur la figure 5.5 décrit la mobilité d'un agent d'un groupe à l'autre : l'agent dénoté AG situé dans le groupe 1 (racine 0) rejoint le groupe 2 (racine 1).

Exemple RL 7 : Création de lien agent (distant)
Cette règle de réaction représente la création d'un moyen de communication entre deux agents pour un but de coordination ou de coopération. Dans le redex de la règle de réaction (voir figure 5.6), on voit deux agents AG et AG' appartenant chacun à un groupe différent (dénoté par racine 0 et racine 1). Dans le réactum, la connexion est établie en créant un lien e1.

Exemple RL 9 : Nouvelles connaissances d'agent
La règle représentée sur la figure 5.7 représente l'ajout d'une nouvelle connais-

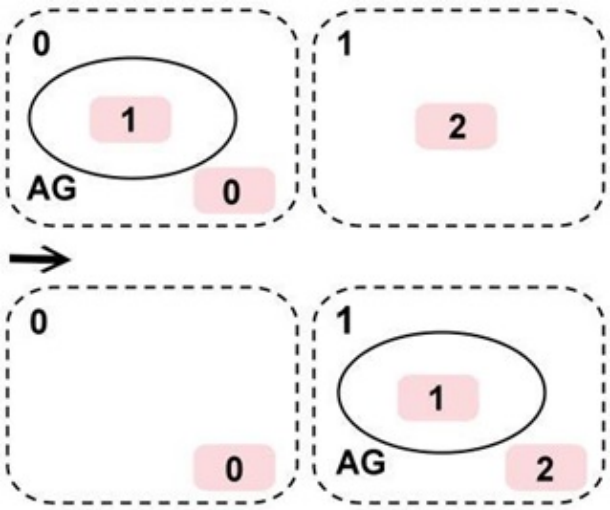


FIGURE 5.5 – Règle de réaction pour la migration d'un agent

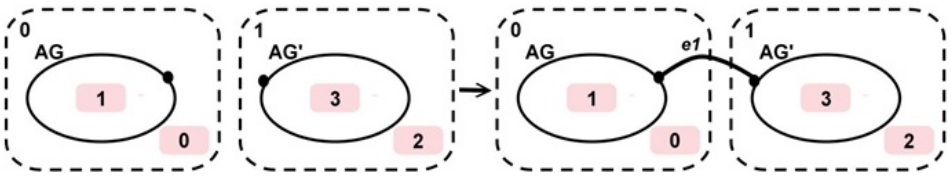


FIGURE 5.6 – Règle de réaction pour la création de lien agent (distant)

sance au nœud B, modélisant les croyances d'un agent BDI. Sur le côté droit (ou reactum de règle) une nouvelle connaissance notée par le nœud K' est ajoutée. Cela peut être dû à une nouvelle perception (par exemple changement dans l'environnement).

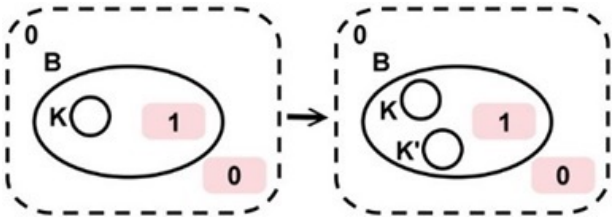


FIGURE 5.7 – Règle de réaction pour l'ajout de nouvelles connaissances (agent)

Exemple RL 12 : Délégation d'un objectif
La règle de réaction de la figure 5.8 modélise la délégation d'un objectif, entre deux agents. Dans le redex, les nœuds G et G 'représentent respectivement les ensembles de désirs de deux agents : le nœud D représente l'objectif à déléguer. La délégation consiste à faire passer le désir D de l'ensemble des désirs G d'un agent, à l'ensemble des désirs G' d'un autre agent.

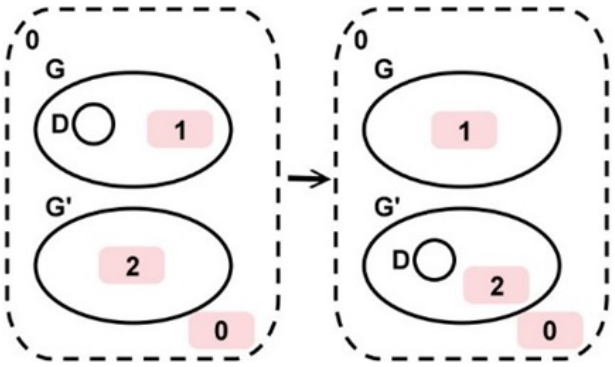


FIGURE 5.8 – Règle de réaction pour la délégation d'un objectif

Dans ce chapitre, nous avons présenté un ensemble de méta-règles de réactions définies pour modéliser la reconfiguration architecturale interne et externe de BDI-MAS. Ces méta- règles de réaction peuvent être instanciées et les règles de réactions résultantes seront utilisées pour modéliser diverses reconfigurations complexes.
Des règles de réactions supplémentaires peuvent être définies pour exprimer d'autres situations comportementales liées à l'architecture BDI-MAS.

Exemple 1 : Résolution d'un objectif interne (niveau agent) :
La figure de la règle de réaction 5.9 décrit comment notre agent BDI est capable de résoudre un objectif interne. Premièrement, on peut voir dans le redex, la présence dans le nœud G d'un désir D1 à satisfaire, on peut aussi noter la présence dans le nœud B d'une connaissance nœud dénotée par K. Ces nœuds sont nécessaires pour déclencher la règle de réaction. Dans le réactum, on peut observer l'apparition d'un nœud P, qui est le meilleur plan parmi ceux qui peuvent satisfaire le désir D1 (choisir le meilleur plan possible reste lié à des heuristiques qui ne peuvent pas apparaître au niveau architectural). Deuxièmement, la création d'un lien e2 entre le nœud D1 et P indique que le désir D1 peut être satisfait par l'exécution du plan P. Enfin, l'exécution de P induit deux cas possibles soit : (1) ajout / suppression de connaissances au niveau du nœud B (2) les croyances de l'agent ne changent pas.

$$AG_{xy}.(B_{e1}.(K|d2)|G.(D1|d4)|I_{e1}.(P|d3)|d1)|d0$$
$$\rightarrow$$
$$AG_{xy}.(B_{e1}.(K|K1|d2)|G.(D1_{e2}|d4)|I_{e1}.(P_{e2}|d3)|d1)|d0$$

Exemple 2 : Ajout d'un nouvel agent
La règle de réaction figure 5.10 décrit la reconfiguration d'un SMA au niveau architectural. C'est-à-dire l'ajout d'un nouvel agent dans une configuration donnée. Comme vous pouvez le voir, notre bigraphe contient une racine notée 0, par conséquent, un agent peut être ajouté dans la région 0, en raison de la présence du site 0 qui permet de dire que cette région peut évoluer par l'ajout d'agents. On peut même expliciter une contrainte sur le nombre d'ajouts possibles. Par exemple au-delà de deux agents dans la région 0, le site 0 disparaît. Cela signifie qu'il n'est plus possible d'ajouter d'autres agents dans cette région.

$$AG_{xy}.(B_{e1}.(K|K1|d2)|G.(D1_{e2}|d4)|I_{e1}).(P_{e2}|d3)|d1)|d0$$
$$\rightarrow$$
$$AG_{xy}.(B_{e1}.(K|K1|d2)|G.(D1_{e2}|d4)|I_{e1}.(P_{e2}|d3)|d1)|$$
$$AG1_{xy}.(B1_{e4}.(K1|K2|d7)|G1.(D2_{e5}|d9)|I1_{e4}.(P1_{e5}|d8)|d6)$$

Exemple 3 : Résolution d'un but externe (collaboration) :
La règle de réaction dans la figure 5.11, décrit comment un agent via ses interfaces (d'entrée et sortie) communique avec d'autres agents pour satisfaire ses buts. Au niveau du nœud G, on note un désir D1, sauf que l'agent actuel n'a pas assez de connaissances afin de résoudre ce désir. Donc, la

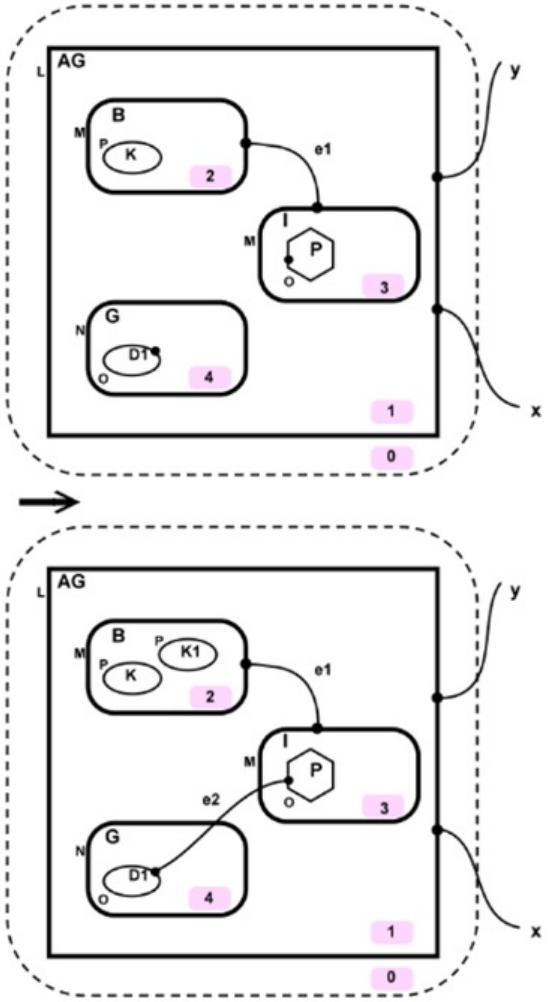


FIGURE 5.9 – Règle de réaction résolution de but interne

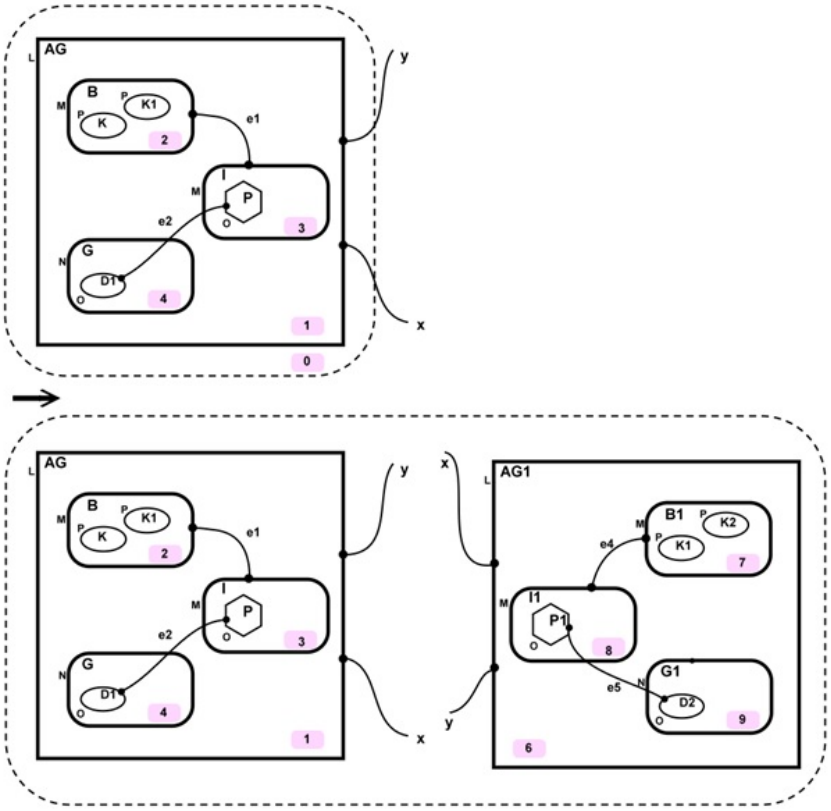


FIGURE 5.10 – Règle de réaction déploiement d'un nouvel agent

construction d'un plan n'est pas possible. Par conséquent, l'agent AG initie une connexion avec l'agent AG1 sous forme de demande (la correspondance entre les interfaces d'entrée et de sortie des deux agents entraîne la création du lien e4). L'agent AG1 reçoit cette demande et la traite dans le domaine de ses compétences. À ce stade, on a deux cas possibles soit : (1) l'agent refuse ou n'arrive pas à comprendre la demande, dans ce cas rien ne se passe au niveau interne de l'agent AG1 ; néanmoins, la demande l'agent AG, recevra un message d'information (de refus ou d'incompréhension de la demande qu'il a émise). (2) L'agent accepte et donc le désir D1 de l'agent AG est ajouté aux désirs de l'agent AG1. A partir du moment où la délégation de but est effectuée, il revient simplement à faire une résolution de but interne (Règle de réaction définie ci-dessus). Lorsque le désir D1 est satisfait par l'agent AG1, il met à jour ses croyances (nœud B). Une fois le processus de résolution interne de l'agent AG1 terminé, l'agent AG via son interface d'entrée (lien e5) reçoit la connaissance dont il a besoin et peut dès lors construire ses propres plans pour satisfaire le désir D1.

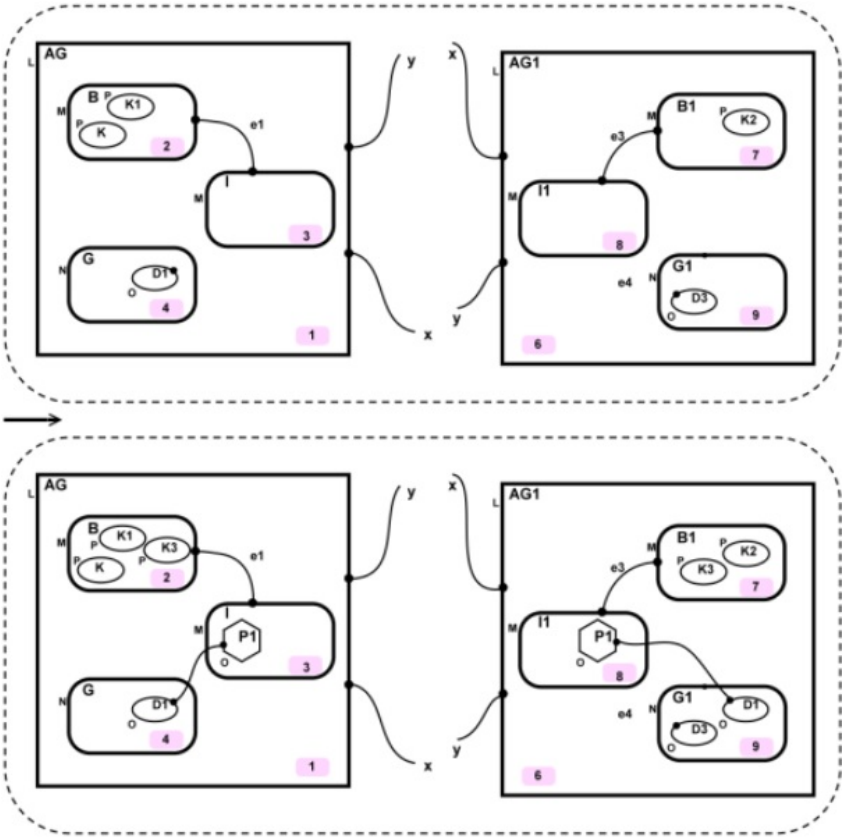


FIGURE 5.11 – Résolution d'un but externe (collaboration)

Listing 5.1 – Spécification algébrique de la règle de réaction figure 5.11.

```
rl0:
AGxy.(Be1.(K|K1|d2) |G.(D1|d4)|I.(d3)|d1)|
AG1xy.(B1e2.(K2|d7)|G1.(D3|d9)|I1e2.(d8) |d6)
->
y/AGxe3y.(Be1.(K |K1 |d2) |G.(D1|d4)|I.(d3)|d1)|
AG1xe3y.(B1e2.(K2|d7)|G1.(D3 |d9)|I1e2.(d8)|d6)

-----------------------------------------------------------
rl1:
y/AGxe3y .(Be1.(K |K1 |d2) |G.(D1|d4)|I.(d3)|d1)|
AG1xe3y.(B1e2.(K2|d7)|G1.(D3 |d9)|I1e2.(d8)|d6)
->
x/AGxe5y.(Be1.(K|K1|d2) |G.(D1|d4)|I.(d3)|d1)|
AG1xe5y.(B1e2.(K2|K3|d7)|G1.(D3|D1e4|d9)|I1e2.(P1e4 |d8)|d6)
-----------------------------------------------------------
rl2:
x/AGxe5y.(Be1.(K|K1|d2) |G.(D1|d4)|I.(d3)|d1)|
AG1xe5y.(B1e2.(K2|K3|d7)|G1.(D3|D1e4|d9)|I1e2.(P1e4 |d8)|d6)
->
AGxy.(Be1.(K|K1|d2) |G.(D1|d4)|I.(d3)|d1)|
AG1xy.(B1e2.(K2|d7)|G1.(D3|d9)|I1e2.(d8)|d6)
-----------------------------------------------------------
rl3:
x/AGxe5y.(Be1.(K|K1|d2) |G.(D1|d4)|I.(d3)|d1)|
AG1xe5y.(B1e2.(K2|K3|d7)|G1.(D3|D1e4|d9)|I1e2.(P1e4 |d8)|d6)
->
AGxy  .(Be1.(K |K1 |K3 |d2) |G.(D1e6 |d4) |I.(P2e6 |d3) |d1) |
AG1xy.(B1e2.(K2 |K3 |d7) |G1.(D3 |d9)|I1e2.(d8) |d6)
```

La résolution d'un but externe peut être exprimée dans une seule règle de réaction nommée $rl_{collaboration}$ qui n'est d'autre que l'exécution des règles $rl_0, rl_1 et rl_3$ dans le cas nominal. Dans le cas où l'agent AG refuse la connaissance (désignée par K3) fournie par l'agent AG1, on aura l'exécution suivante rl_0, rl_1 et rl_2.

5.5 Conclusion

Dans ce chapitre, nous avons présenté le modèle BDI-MAS qui est une instance de notre approche générique BRS-MAS appliquée sur des agents de type BDI. Nous avons d'abord adopté une architecture en deux couches pour décrire notre SMA. Cette vue offre un support méthodologique permettant de décomposer un système multi-agent en un ensemble d'éléments compréhensibles. Ensuite, nous avons proposé un modèle formel basé bigraphe pour la spécification structurelle de cette architecture. En outre, nous avons défini

l'architecture interne de notre agent BDI qui est représentée par une hiérarchie de nœuds interconnectés. Puis, nous avons montré comment ce type d'agent peut interagir avec son environnement mais aussi avec d'autres agents (Ceci dans le but de coopérer au sein d'un SMA), donc un système multi-agent est spécifié à la fois au niveau individuel (Agent) et social (SMA) qui simplifie considérablement la lisibilité et l'analyse des architectures SMA. Aussi, nous avons proposé un ensemble de méta-règles de réactions qui peuvent être instanciées pour modéliser les reconfigurations dynamiques sur les deux niveaux (agent et social). Pour le niveau social, nous avons spécifié la création / suppression de groupe, le déploiement, la mobilité d'agents ainsi que l'interaction entre ces agents qui est modélisée par la création et la suppression de liens. Au niveau agent, nous avons formalisé la délégation, la résolution interne de buts, l'ajout de connaissances, etc. Cet ensemble de règles de réactions permet au développeur d'analyser correctement les propriétés de l'architecture BDI-MAS ce qui inclut la modélisation du comportement des agents BDI et la description de la reconfiguration au niveau de l'architecture globale du SMA.

Chapitre 6

Vérification des architectures BRS-MAS

Sommaire

6.1 Introduction

L'émergence à grande échelle des réseaux informatiques a donné lieu à de nombreuses applications distribuées. Ces applications nécessitent une forte interaction entre les différentes entités distribuées sur le réseau. Ces entités peuvent partager les mêmes ressources et les mêmes objectifs. Plusieurs modèles de développement pour ces applications distribuées ont été proposés dans la littérature. Cependant, l'importance de la question, est d'être convaincu de la légitimité et de la confiance accordée aux applications informatiques. Ces préoccupations ont conduit à l'émergence des méthodes de développement et de vérification formelle.

La vérification du logiciel devient essentielle dans le développement des systèmes informatiques. En outre, l'utilisation des méthodes formelles contribue à l'intégration de la vérification dans le processus de conception. Par conséquent, elle facilite la détection précoce des défauts de conception. Ces méthodes, sont basées sur des théories et des notations mathématiques permettant à la fois de spécifier formellement le programme, de vérifier et de prouver que son implantation est conforme aux spécifications initiales. Les méthodes formelles sont reconnues par des standards, de sorte que le septième et dernier niveau de confiance des Critères Communs [Ernst 2009] est accordé aux applications créées en utilisant les méthodes formelles. La prise de conscience de l'importance de vérifier plus attentivement les programmes et la maturité des outils dédiés à cette tâche a généré une croissance considérable des programmes de

vérification formelle au cours de la dernière décennie. En outre, il existe un certain nombre de techniques de vérifications formelles et la vérification de modèles est l'une parmi tant d'autres [Zhang 2010] et BigMc (vérificateur de modèle Bigraphical) [Perrone 2012] est un des outils de vérification spécifié en tant que système réactif bigraphique. BigMc s'assure que le comportement du système répond aux propriétés attendues. Cette vérification est entièrement automatisée et consiste à explorer tous les états possibles de système. Le résultat de cette analyse est de valider que chaque propriété est satisfaite ou non. Dans ce dernier cas, cet outil renvoie un contre-exemple. Dans notre cas, nous utilisons BigMC pour vérifier les propriétés de sûreté et de vivacité telles que l'absence de blocage et de violation que le modèle ne devrait pas permettre lors de son exécution. Tout d'abord, nous spécifions l'aspect structurel (c'est-à-dire les nœuds, leur signature et les interfaces entrantes et sortantes ...) puis l'aspect dynamique (c'est-à-dire les règles de réaction, exemple : résolution interne du but ...) en utilisant la syntaxe de BigMC. Ensuite, nous formulons les propriétés que nous souhaiterions vérifier sur chaque exemple. Enfin, nous analyserons et validerons le résultat donné par l'outil BigMC.

6.2 Exemple illustratif

Nous présentons, dans cette section, une étude de cas d'un système de commerce électronique. Cette étude de cas représente un système multi-agent BDI (Croyance, Désir et Intention) qui modélise un système de commerce électronique. Le système comprend trois agents : un client (acheteur) et deux vendeurs. Le client négocie avec les deux vendeurs pour décider le quel choisir sur la base du prix.

Nous modélisons l'état initial du système multi-agent par un bigraphe G_{BDI} présenté dans la Figure 6.1, nous avons trois agents répartis physiquement sur deux racines ; La racine 0 contient le client (acheteur), modélisé par le noeud CL alors que la racine 1 contient les deux vendeurs indiqués respectivement par les nœuds BY1 et BY2. Ces agents ont la même architecture standard définie dans notre modèle d'agent BDI-MAS où chaque agent est doté de moyens abstraits de communication. Dans cet exemple, l'interaction est basée sur le passage des messages entre le client (nœud CL) et les deux vendeurs (nœuds BY1 et BY2) à travers les interfaces entrantes et sortantes, X et Y, ce qui permet aux agents d'échanger des informations sur l'avancement de leurs tâches, ainsi que sur le processus de négociation. Ce dernier, permet aux agents de décider ensemble de la distribution des tâches (comme le réseau contractuel [Sandholm 1993] basé sur le processus de négociation). Les conséquences d'une interaction directe sont décidées par les lois comportementales des agents dans notre exemple, l'émergence d'un

désir modélisé par le nœud D1 dans l'ensemble des désirs G du client (agent désigné par CL). Notre agent BDI-MAS montre un haut niveau d'abstraction en utilisant des interfaces abstraites (noms internes et externes). Ainsi, il peut s'exprimer au moyen de langages élaborés et de protocoles de communication (tels que KQML [Finin 1994], FIPA ACL [Fipa 2002]).

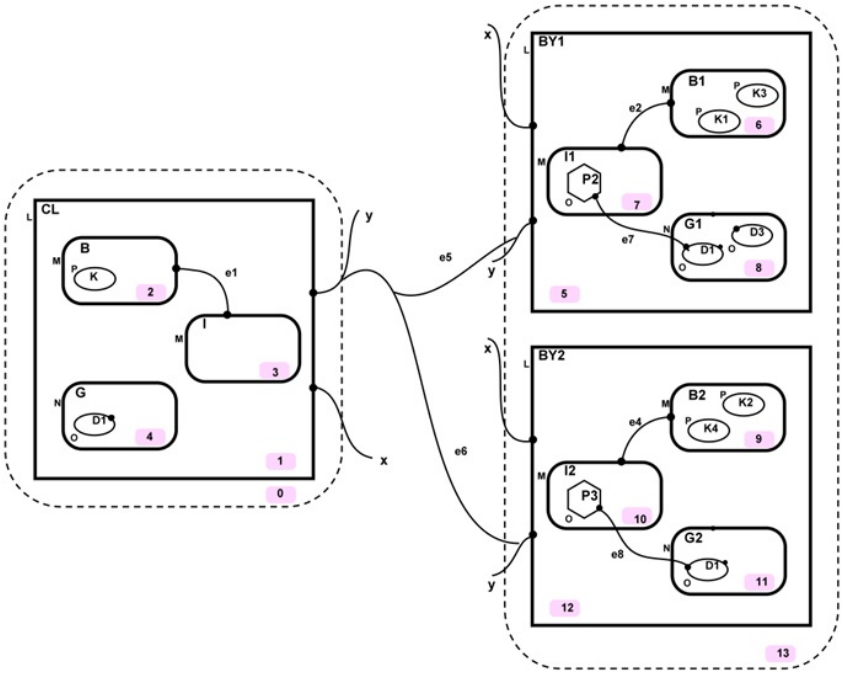


FIGURE 6.1 – Configuration initiale du système de E-commerce

$$CL_{xy}.(B_{e1}.(K|d_2))|G.(D1|d_4)|I_{e1}.(d_3)|d_1)|d_0||$$
$$BY1_{xy}.(B1_{e2}.(K1|d_6))|G1.(D2_{e3}|d_8)|I1_{e2}.(P1_{e3}|d_7)|d_5)|$$
$$BY2_{xy}.(B2_{e4}.(K2|d_9))|G2.(d_1 1)|I2_{e4}.(d_1 0)|d_1 2)|d_1 3$$

Dans ce qui suit, nous montrons la reconfiguration de l'architecture SMA actuelle par l'application successive des règles de réaction définies dans le chapitre précédent (règles de réaction de connexion / déconnexion d'agent, délégation, exécution d'un plan et l'ajout de nouvelles connaissance, etc.). Le client interagit avec les deux fournisseurs afin de réaliser sa transaction. Le bigraphe représentant l'état final obtenu en appliquant ces règles de réaction, est donné par la figure 6.2.

Scénario :

- Tout d'abord, l'acheteur doit manifester le désir d'acheter un produit (le désir de connaître le prix d'un produit donné) le désir est noté par D1.
- Ensuite, un protocole de négociation (le réseau contractuel, FIPA) est adopté.
- L'agent CL envoie un appel de propositions (CFP) à tous les vendeurs via son interface de sortie y.
- Une connexion est établie entre l'interface de sortie de l'agent CL et les interfaces d'entrée des agents BY1, BY2.
- Les agents BY1 et BY2 reçoivent le CFP sous la forme d'un désir (désigné par D1).
- Les agents BY1 et BY2 formulent leurs propositions en effectuant une résolution de but interne du désir D1. Il en résulte l'émergence de la connaissance K3, K4 dans les modules de croyances des agents B1 et B2 des vendeurs.
- L'agent CL, à travers son interface d'entrée, crée les liens e5 et e6 pour recevoir les propositions des deux vendeurs.
- Enfin, l'agent CL choisit la meilleure proposition en se basant sur des critères en accord avec la politique de l'agent (par exemple, le budget du client). La connaissance K4 est ajoutée au module de croyances B de l'agent CL.
- Ainsi, le désir D1 (pour connaître le prix d'un produit) est résolu. Par conséquent, une mise à jour des croyances (module B) est effectuée. De ce fait, une nouvelle connaissance K5 apparaît. Les règles de réactions qui décrivent le scénario ci-dessus sont présentées par le tableau 6.1 Tableau 6.1 La spécification algébrique de règles de réactions de l'exemple.

Listing 6.1 – La spécification algébrique de règles de réactions de l'exemple.

```
Call for proposal rule :
CLxy.(Be1.(K|d2) |G.(D1|d4)|Ie1.(d3)|d1)|d0||
BY1xy.(B1e2.(K1|d6) |G1.(D2e3|d8)|I1e2.(P1e3|d7)|d5)|
BY2xy.(B2e4.(K2|d9)|G2.(d11)|I2e4.(d10)|d12)|d13
->
CLxye5e6 .(Be1.(K|d2)|G.(D1|d4)|Ie1.(d3)|d1)|d0||
BY1xye5.(B1e2.(K1|K3|d6)|G1.(D1e7|D3|d8)|I1e2.(P2e7|d7) |d5)|
BY2xye6.(B2e4.(K2|K4|d9)|G2.(D1e8|d11)|I2e4.(P3e8|d10)|d12)|
   d13
-----------------------------------------------------------------

Make and send proposal rule:
```

```
CLxye5e6.(Be1.(K|d2)|G.(D1|d4)|Ie1.(d3)|d1)|d0||
BY1xye5.(B1e2.(K1|K3|d6)|G1.(D1e7|D3|d8)|I1e2.(P2e7|d7)|d5)|
BY2xye6.(B2e4.(K2|K4|d9)|G2.(D1e8 |d11)|I2e4.(P3e8|d10)|d12)|
    d13
->
CLxye9e10.(Be1.(K|d2)|G.(D1|d4)|Ie1.(d3)|d1)|d0||
BY1xye9.(B1e2.(K1|K3|d6)|G1.(D3|d8)|I1e2.(d7) |d5)|
BY2xye10.(B2e4.(K2|K4|d9)|G2.(d11)|I2e4.(d10)|d12)|d13
-------------------------------------------------------------

Chose a proposal rule:
CLxye9e10.(Be1.(K|d2)|G.(D1|d4)|Ie1.(d3)|d1)|d0 ||
BY1xye9.(B1e2.(K1|K3|d6)|G1.(D3|d8)|I1e2.(d7)|d5)|
BY2xye10.(B2e4.(K2|K4|d9)|G2.(d11)|I2e4.(d10)|d12)|d13
->
CLxye10e11.(Be1.(K|K4|d2)|G.(D1e12|d4)|Ie1.(P4e12|d3)|d1)|d0
    ||
BY1xy.(B1e2.(K1|K3|d6)|G1.(D3|d8)|I1e2.(d7)|d5)|
BY2xye10e11.(B2e4.(K2|K4|d9)|G2.(d11)| I2e4.(d10)|d12)|d13
-------------------------------------------------------------

Validate the proposal rule:
CLxye10e11.(Be1.(K|K4|d2)|G.(D1e12|d4)|Ie1.(P4e12|d3)|d1)|d0
    ||
BY1xy.(B1e2.(K1|K3|d6)|G1.(D3|d8)|I1e2.(d7)|d5)|
BY2xye10e11.(B2e4.(K2|K4|d9)|G2.(d11)|I2e4.(d10)|d12)|d13
->
CLxy.(Be1.(K|K4|K5|d2)|G.(D4|d4)|Ie1.(d3)|d1)|d0||
BY1xy.(B1e2.(K1|K3|d6)|G1.(D3|d8)|I1e2.(d7) |d5)|
BY2xy.(B2e4.(K2|K4|d9)|G2.(d11)|I2e4.(d10)|d12)|d13
```

$$CL_{xy}.(B_{e1}.(K|d_2))|G.(D1|d_4)|I_{e1}.(d_3)|d_1)|d_0||$$
$$BY1_{xy}.(B1_{e2}.(K1|d_6))|G1.(D2_{e3}|d_8)|I1_{e2}.(P1_{e3}|d_7)|d_5)|$$
$$BY2_{xy}.(B2_{e4}.(K2|d_9))|G2.(d_{11})|I2_{e4}.(d_{10})|d_{12})|d_{13}$$

6.3 Vérification de l'accessibilité

Notre objectif dans cette section est de démontrer la faisabilité et de vérifier l'exactitude de notre modèle d'architecture proposé, BRS-MAS. Pour ce faire, nous utilisons le système de commerce électronique présenté dans la section 2 de ce chapitre. Par conséquent, dans ce qui suit, nous présentons la spécification bigraphique complète de l'exemple illustratif en utilisant la syntaxe de BigMC.

Pour une meilleure clarté, cette implémentation est divisée en trois parties représentées par les tableaux 6.3, 6.4 et 6.5. Le tableau 6.3 représente la partie structurelle du système de commerce électronique dans la syntaxe de BigMc.

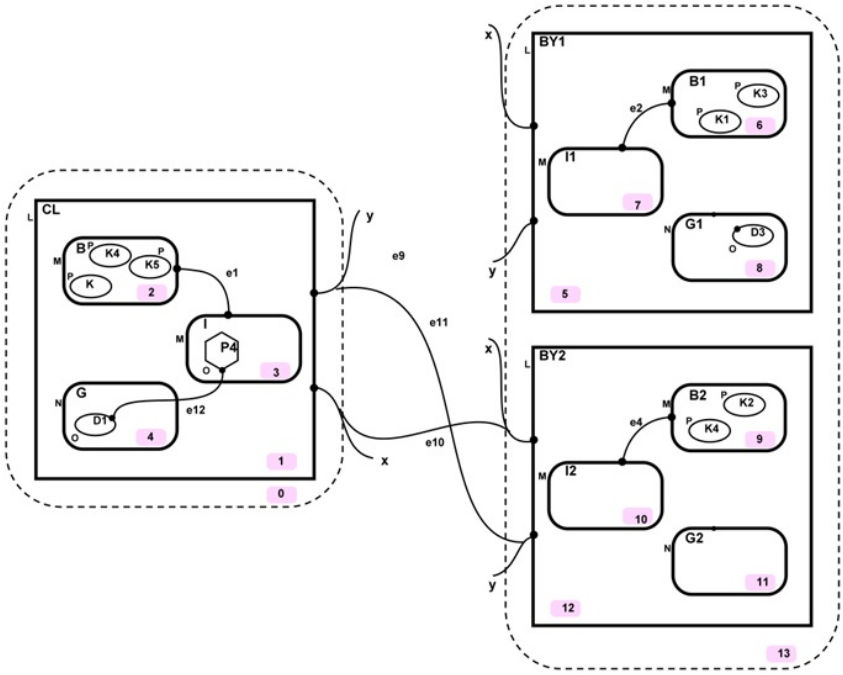


FIGURE 6.2 – Reconfiguration du système de commerce électronique

$$
\begin{aligned}
M &::= E; M \mid E \\
E &::= \%passive\ k : arity \\
E &::= \%active\ k : arity \\
E &::= \%rule\ n\ T \to T \\
E &::= \%property\ n\ P \\
E &::= T \to T \mid T \\
T &::= K.T \mid T \mid T \mid T \parallel T \mid \$n \mid K \mid nil \\
K &::= k[names] \mid k \\
names &::= n, names \mid n \\
n &::= [a-zA-Z][a-zA-Z0-9]* \mid - \\
P &::= matches(T) \mid terminal() \mid !P
\end{aligned}
$$

FIGURE 6.3 – La grammaire complète de BigMC [Perrone 2012]

Listing 6.2 – La spécification algébrique de règles de réactions de l'exemple

```
# Consumer nodes
%active Consumer: 2;
%active Intensions: 1;
%active Beliefs: 1;
%active Goals: 0;
%passive K: 0;
%passive K5: 0;
%passive Desire1: 1;

# Seller 1 nodes
%active Seller1: 2;
%active Intensions1: 1;
%active Beliefs1: 1;
%active Goals1: 0;
%passive K1: 0;
%passive K3: 0;
%passive Desire3: 1;
%passive Plan2: 1;

# Seller 2 nodes
%active Seller2: 2;
%active Intensions2: 1;
%active Beliefs2: 1;
%active Goals2: 0;
%passive K2: 0;
%passive K4: 0;
%passive Plan3: 1;

# System edges
%name in;
%name out;
%name e1;
%name e2;
%name e3;
%name e4;
%name e5;
%name e6;

\# E-commerce System model
Consumer[-,-].(Beliefs[e1].(K)|Goals.(Desire1[-]) |Intensions[
    e1]) |
Seller1[-,-].(Beliefs1[e2].(K1)|Goals1 |Intensions1[e2]) |
Seller2[-,-].(Beliefs2[e3].(K2)|Goals2 |Intensions2[e3]) ;
```

Le tableau 6.4 montre la partie dynamique de notre système de commerce électronique, à travers une séquence de règles de réaction modélisant le comportement des agents et leurs interactions, comme décrit dans l'exemple illus-

tratif. Nous pouvons également distinguer deux types de règles de réaction : de lien et de place. Le premier est chargé de créer ou de supprimer des liens entre les nœuds tandis que le second type est responsable de la création, du déplacement ou de la suppression de nœuds dans notre architecture BDI-MAS.

Listing 6.3 – Spécification des règles de réaction du système de commerce électronique

```
# Call for proposal reaction rules
Consumer[-,-].(Beliefs[e1].(K)|Goals.(Desire1[-])|Intensions[
    e1]) |
Seller1[-,-].(Beliefs1[e2].(K1)|Goals1 |Intensions1[e2]) |
Seller2[-,-].(Beliefs2[e3].(K2)|Goals2 |Intensions2[e3])
->
Consumer[in,-].(Beliefs[e1].(K|$0)|Goals.(Desire1[-]) |
    Intensions[e1]) |
Seller1[in,-].(Beliefs1[e2].(K1)|Goals1.(Desire1[-] |$1) |
    Intensions1[e2].($3)) |
Seller2[in,-].(Beliefs2[e3].(K2)|Goals2.(Desire1[-]|$2) |
    Intensions2[e3].($4));

# Make proposal reaction rules (seller1)
Seller1[in,-].(Beliefs1[e2].(K1)|Goals1.(Desire1[-]|$1)|
    Intensions1[e2].($3))
 ->
Seller1[in,-].(Beliefs1[e2].(K1|$5)|Goals1.(Desire1[e4]| $1) |
Intensions1[e2].(Plan1[e4] | $3));

Seller1[in,-].(Beliefs1[e2].(K1|$5)|Goals1.($1)|Intensions1[e2
    ].($3))
->
Seller1[in,-].(Beliefs1[e2].(K1|K3|$7)|Goals1.($1) |
    Intensions1[e2].($3));

# Make proposal reaction rules (seller2)
Seller2[in,-].(Beliefs2[e3].(K2)|Goals2.(Desire1[-]|$2)|
    Intensions2[e3].($4))
->
Seller2[in,-].(Beliefs2[e3].(K2|$6)|Goals2.(Desire1[e5] |$2) |
Intensions2[e3].(Plan2[e5] | $4));

Seller2[in,-].(Beliefs2[e3].(K2|$6)|Goals2.($2) |Intensions2[
    e3].($4))
->
Seller2[in,-].(Beliefs2[e3].(K2|K4| $8)|Goals2.($2) |
    Intensions2[e3].($4));

# Send proposals to consumer reaction rule
```

```
Consumer[in,-].(Beliefs[e1].(K|$0)|Goals.(Desire1[-]) |
   Intensions[e1]) |
Seller1[in,-].(Beliefs1[e2].(K1|K3|$7)|Goals1 |Intensions1[e2
   ]) |
Seller2[in,-].(Beliefs2[e3].(K2|K4|$8)|Goals2 |Intensions2[e3
   ])
 ->
Consumer[-,out].(Beliefs[e1].(K|K4|$0)|Goals.(Desire1[-]) |
   Intensions[e1]) |
Seller1[-,out].(Beliefs1[e2].(K1|K3|$7)|Goals1 |Intensions1[e2
   ]) |
Seller2[-,out].(Beliefs2[e3].(K2|K4|$8)|Goals2 |Intensions2[e3
   ]);

# Chose a proposal rule reaction rule (Consumer)
Consumer[-,out].(Beliefs[e1].(K|K4|$0)|Goals.(Desire1[-]) |
   Intensions[e1]) |
Seller1[-,out].(Beliefs1[e2].(K1|K3|$7)|Goals1 |Intensions1[e2
   ]) |
Seller2[-,out].(Beliefs2[e3].(K2|K4|$8)|Goals2 |Intensions2[e3
   ])
->
Consumer[-,-].(Beliefs[e1].(K|K4|$0)|Goals.(Desire1[-]) |
   Intensions[e1].($9)) |
Seller1[-,-].(Beliefs1[e2].(K1|K3|$7)|Goals1 |Intensions1[e2])
    |
Seller2[-,-].(Beliefs2[e3].(K2|K4|$8)|Goals2 |Intensions2[e3])
   ;

# Internal goal resolution reaction rule (Consumer)
Consumer[-,-].(Beliefs[e1].(K|K4|$0)|Goals.(Desire1[-]) |
   Intensions[e1].($9))
->
Consumer[-,-].(Beliefs[e1].(K|K4|$0)|Goals.(Desire1[e6]) |
   Intensions[e1].(Plan3[e6]));

Consumer[-,-].(Beliefs[e1].(K|K4|$0)|Goals.($11) |Intensions[
   e1].($10))
->
Consumer[-,-].(Beliefs[e1].(K|K4|K5|$12)|Goals.($11) |
   Intensions[e1]);
```

Maintenant, que nous avons spécifié nos aspects structurels et dynamiques en utilisant la syntaxe de BigMc, l'avant-dernière étape avant la vérification est de spécifier ou de formuler les propriétés à vérifier sur notre modèle. La première propriété que nous aimerions vérifier est appelée $violation\_free$. Nous utilisons pour cela le prédicat $!match(T)$, ce qui signifie que nous ne devons pas trouver une correspondance avec l'expression entre parenthèses lors

de l'exécution de notre système. La deuxième propriété est $deadlock\_free$ qui utilise le prédicat $!Terminal()$. Ce prédicat tel que transcrit ici indique qu'il y aura un état futur accessible à partir de la règle de réaction actuelle. Pour des propriétés plus élaborées, les opérateurs booléen commun tels que AND, OR et NOT sont utilisés, voir le tableau 6.3.

Listing 6.4 – Spécification de la propriété $deadlock\_free$ en utilisant la syntaxe de BigMC

```
%property deadlock_free !terminal() &&
%property violation_free !match ((Consumer[-,- ].(Beliefs[e1
   ].(K|K4|K5|$0)|
Goals.(Desire1[-]) |Intensions[e1].(Plan3[-])));
```

Les tableaux 6.3, 6.4 et 6.5 combinés représentent la spécification bigraphique complète du système de commerce électronique et les propriétés à vérifier, écrites dans la syntaxe de BigMC.
Le résultat de la vérification est représenté dans la figure 6.3. Il montre qu'après 27 étapes, modèle checker, BigMc a exécuté toutes les règles de réaction et a atteint l'état souhaité avec succès. (On peut voir que le consommateur a acquis une nouvelle connaissance notée par K4. Ainsi, il est maintenant en mesure d'atteindre son objectif en créant le lien e6 entre le plan3 et le désir1) sans signaler de blocage ou de violation.

Listing 6.5 – Résultats de la vérification du modèle

```
> C:\Progra~1\BigMC/bin/bigmc -m 1000 -r 50 -p C:\DOCUME~1\
   dhte\LOCALS~1\Temp\bigmc_model5537536387463849034.bgm
1: (Consumer[-,-].(Intensions[e1].nil | Beliefs[e1].K.nil |
   Goals.Desire1[-].nil) | Seller1[-,-].(Beliefs1[e2].K1.nil |
    Goals1.nil | Intensions1[e2].nil) | Seller2[-,-].(Beliefs2
   [e3].K2.nil | Goals2.nil | Intensions2[e3].nil))
2: (Consumer[in,-].(Beliefs[e1].K.nil | Goals.Desire1[-].nil |
    Intensions[e1].nil) | Seller1[in,-].(Beliefs1[e2].K1.nil |
    Goals1.nil | Intensions1[e2].nil) | Seller2[in,-].(
   Beliefs2[e3].K2.nil | Goals2.nil | Intensions2[e3].nil))
3: (Consumer[in,-].(Beliefs[e1].(K.nil) | Goals.Desire1[-].nil
    | Intensions[e1].nil) | Seller1[in,-].(Beliefs1[e2].K1.nil
    | Goals1.(Desire1[-].nil) | Intensions1[e2].nil) | Seller2
   [in,-].(Beliefs2[e3].K2.nil | Goals2.(Desire1[-].nil) |
   Intensions2[e3].nil))
4: (Consumer[in,-].(Beliefs[e1].K.nil | Goals.Desire1[-].nil |
    Intensions[e1].nil) | Seller1[in,-].(Beliefs1[e2].(K1.nil
   | K3.nil) | Goals1.nil | Intensions1[e2].nil) | Seller2[in
   ,-].(Beliefs2[e3].K2.nil | Goals2.nil | Intensions2[e3].nil
   ))
5: (Consumer[in,-].(Beliefs[e1].K.nil | Goals.Desire1[-].nil |
```

```
    Intensions[e1].nil) | Seller1[in,-].(Beliefs1[e2].K1.nil |
    Goals1.nil | Intensions1[e2].nil) | Seller2[in,-].(
   Beliefs2[e3].(K2.nil | K4.nil) | Goals2.nil | Intensions2[
   e3].nil))
6: (Consumer[in,-].(Beliefs[e1].(K.nil) | Goals.Desire1[-].nil
    | Intensions[e1].nil) | Seller1[in,-].(Goals1.(Desire1[e4
   ].nil) | Beliefs1[e2].(K1.nil) | Intensions1[e2].(Plan1[e4
   ].nil)) | Seller2[in,-].(Beliefs2[e3].K2.nil | Goals2.(
   Desire1[-].nil) | Intensions2[e3].nil))
7: (Consumer[in,-].(Beliefs[e1].(K.nil) | Goals.Desire1[-].nil
    | Intensions[e1].nil) | Seller1[in,-].(Beliefs1[e2].(K1.
   nil | K3.nil) | Goals1.(Desire1[-].nil) | Intensions1[e2].
   nil) | Seller2[in,-].(Beliefs2[e3].K2.nil | Goals2.(Desire1
   [-].nil) | Intensions2[e3].nil))
8: (Consumer[in,-].(Beliefs[e1].(K.nil) | Goals.Desire1[-].nil
    | Intensions[e1].nil) | Seller1[in,-].(Beliefs1[e2].K1.nil
    | Goals1.(Desire1[-].nil) | Intensions1[e2].nil) | Seller2
   [in,-].(Beliefs2[e3].(K2.nil) | Goals2.(Desire1[e5].nil) |
   Intensions2[e3].(Plan2[e5].nil)))
9: (Consumer[in,-].(Beliefs[e1].(K.nil) | Goals.Desire1[-].nil
    | Intensions[e1].nil) | Seller1[in,-].(Beliefs1[e2].K1.nil
    | Goals1.(Desire1[-].nil) | Intensions1[e2].nil) | Seller2
   [in,-].(Beliefs2[e3].(K2.nil | K4.nil) | Goals2.(Desire1
   [-].nil) | Intensions2[e3].nil))
10: (Seller1[in,-].(Intensions1[e2].nil | Goals1.nil |
   Beliefs1[e2].(K3.nil | K1.nil)) | Consumer[in,-].(Goals.
   Desire1[-].nil | Beliefs[e1].K.nil | Intensions[e1].nil) |
   Seller2[in,-].(Intensions2[e3].nil | Goals2.nil | Beliefs2[
   e3].(K4.nil | K2.nil)))
11: (Consumer[in,-].(Goals.Desire1[-].nil | Intensions[e1].nil
    | Beliefs[e1].(K.nil)) | Seller1[in,-].(Beliefs1[e2].(K1.
   nil | K3.nil) | Goals1.(Desire1[e4].nil) | Intensions1[e2
   ].(Plan1[e4].nil)) | Seller2[in,-].(Beliefs2[e3].K2.nil |
   Goals2.(Desire1[-].nil) | Intensions2[e3].nil))
12: (Consumer[in,-].(Beliefs[e1].(K.nil) | Goals.Desire1[-].
   nil | Intensions[e1].nil) | Seller1[in,-].(Goals1.(Desire1[
   e4].nil) | Beliefs1[e2].(K1.nil) | Intensions1[e2].(Plan1[
   e4].nil)) | Seller2[in,-].(Beliefs2[e3].(K2.nil | K4.nil) |
    Goals2.(Desire1[-].nil) | Intensions2[e3].nil))
13: (Consumer[in,-].(Beliefs[e1].(K.nil) | Goals.Desire1[-].
   nil | Intensions[e1].nil) | Seller1[in,-].(Goals1.(Desire1[
   e4].nil) | Beliefs1[e2].(K1.nil) | Intensions1[e2].(Plan1[
   e4].nil)) | Seller2[in,-].(Beliefs2[e3].(K2.nil) | Goals2.(
   Desire1[e5].nil) | Intensions2[e3].(Plan2[e5].nil)))
14: (Consumer[in,-].(Beliefs[e1].(K.nil) | Goals.Desire1[-].
   nil | Intensions[e1].nil) | Seller1[in,-].(Beliefs1[e2].(K1
   .nil | K3.nil) | Goals1.(Desire1[-].nil) | Intensions1[e2].
   nil) | Seller2[in,-].(Beliefs2[e3].(K2.nil | K4.nil) |
   Goals2.(Desire1[-].nil) | Intensions2[e3].nil))
15: (Consumer[in,-].(Beliefs[e1].(K.nil) | Goals.Desire1[-].
```

```
   nil | Intensions[e1].nil) | Seller1[in,-].(Beliefs1[e2].(K1
   .nil | K3.nil) | Goals1.(Desire1[-].nil) | Intensions1[e2].
   nil) | Seller2[in,-].(Beliefs2[e3].(K2.nil) | Goals2.(
   Desire1[e5].nil) | Intensions2[e3].(Plan2[e5].nil)))
16: (Seller2[in,-].(Intensions2[e3].(Plan2[e5].nil) | Goals2.(
   Desire1[e5].nil) | Beliefs2[e3].(K4.nil | K2.nil)) |
   Seller1[in,-].(Goals1.(Desire1[-].nil) | Intensions1[e2].
   nil | Beliefs1[e2].K1.nil) | Consumer[in,-].(Intensions[e1
   ].nil | Goals.Desire1[-].nil | Beliefs[e1].(K.nil)))
17: (Consumer[-,out].(Beliefs[e1].(K.nil | K4.nil) | Goals.
   Desire1[-].nil | Intensions[e1].nil) | Seller2[-,out].(
   Intensions2[e3].nil | Goals2.nil | Beliefs2[e3].(K2.nil |
   K4.nil)) | Seller1[-,out].(Intensions1[e2].nil | Beliefs1[
   e2].(K3.nil | K1.nil) | Goals1.nil))
18: (Seller2[in,-].(Intensions2[e3].(Plan2[e5].nil) | Goals2.(
   Desire1[e5].nil) | Beliefs2[e3].(K2.nil)) | Seller1[in,-].(
   Intensions1[e2].(Plan1[e4].nil) | Goals1.(Desire1[e4].nil)
   | Beliefs1[e2].(K1.nil | K3.nil)) | Consumer[in,-].(Beliefs
   [e1].(K.nil) | Intensions[e1].nil | Goals.Desire1[-].nil))
19: (Consumer[in,-].(Goals.Desire1[-].nil | Intensions[e1].nil
    | Beliefs[e1].(K.nil)) | Seller1[in,-].(Beliefs1[e2].(K1.
   nil | K3.nil) | Goals1.(Desire1[e4].nil) | Intensions1[e2
   ].(Plan1[e4].nil)) | Seller2[in,-].(Beliefs2[e3].(K2.nil |
   K4.nil) | Goals2.(Desire1[-].nil) | Intensions2[e3].nil))
20: (Seller2[in,-].(Intensions2[e3].(Plan2[e5].nil) | Goals2.(
   Desire1[e5].nil) | Beliefs2[e3].(K2.nil | K4.nil)) |
   Seller1[in,-].(Intensions1[e2].(Plan1[e4].nil) | Beliefs1[
   e2].(K1.nil) | Goals1.(Desire1[e4].nil)) | Consumer[in,-].(
   Intensions[e1].nil | Goals.Desire1[-].nil | Beliefs[e1].(K.
   nil)))
21: (Seller2[in,-].(Goals2.(Desire1[e5].nil) | Beliefs2[e3].(
   K2.nil | K4.nil) | Intensions2[e3].(Plan2[e5].nil)) |
   Seller1[in,-].(Intensions1[e2].nil | Goals1.(Desire1[-].nil
   ) | Beliefs1[e2].(K1.nil | K3.nil)) | Consumer[in,-].(
   Intensions[e1].nil | Goals.Desire1[-].nil | Beliefs[e1].(K.
   nil)))
22: (Seller2[-,-].(Intensions2[e3].nil | Goals2.nil | Beliefs2
   [e3].(K4.nil | K2.nil)) | Seller1[-,-].(Intensions1[e2].nil
    | Goals1.nil | Beliefs1[e2].(K3.nil | K1.nil)) | Consumer
   [-,-].(Intensions[e1].nil | Goals.Desire1[-].nil | Beliefs[
   e1].(K.nil | K4.nil)))
23: (Seller2[in,-].(Beliefs2[e3].(K2.nil | K4.nil) | Goals2.(
   Desire1[e5].nil) | Intensions2[e3].(Plan2[e5].nil)) |
   Seller1[in,-].(Intensions1[e2].(Plan1[e4].nil) | Goals1.(
   Desire1[e4].nil) | Beliefs1[e2].(K1.nil | K3.nil)) |
   Consumer[in,-].(Beliefs[e1].(K.nil) | Intensions[e1].nil |
   Goals.Desire1[-].nil))
24: (Seller1[-,-].(Beliefs1[e2].(K3.nil | K1.nil) | Goals1.nil
    | Intensions1[e2].nil) | Seller2[-,-].(Beliefs2[e3].(K4.
   nil | K2.nil) | Intensions2[e3].nil | Goals2.nil) |
```

```
   Consumer[-,-].(Intensions[e1].Plan3[e6].nil | Goals.Desire1
   [e6].nil | Beliefs[e1].(K4.nil | K.nil)))
25: (Seller1[-,-].(Intensions1[e2].nil | Beliefs1[e2].(K1.nil
   | K3.nil) | Goals1.nil) | Seller2[-,-].(Beliefs2[e3].(K4.
   nil | K2.nil) | Goals2.nil | Intensions2[e3].nil) |
   Consumer[-,-].(Goals.Desire1[-].nil | Beliefs[e1].(K.nil |
   K5.nil | K4.nil) | Intensions[e1].nil))
26: (Consumer[-,-].(Goals.Desire1[e6].nil | Intensions[e1].nil
    | Beliefs[e1].(K4.nil | K5.nil | K.nil)) | Seller2[-,-].(
   Beliefs2[e3].(K4.nil | K2.nil) | Intensions2[e3].nil |
   Goals2.nil) | Seller1[-,-].(Intensions1[e2].nil | Goals1.
   nil | Beliefs1[e2].(K3.nil | K1.nil)))
27: (Seller1[-,-].(Beliefs1[e2].(K1.nil | K3.nil) | Goals1.nil
    | Intensions1[e2].nil) | Seller2[-,-].(Beliefs2[e3].(K4.
   nil | K2.nil) | Goals2.nil | Intensions2[e3].nil) |
   Consumer[-,-].(Goals.Desire1[e6].nil | Beliefs[e1].(K4.nil
   | K5.nil | K.nil) | Intensions[e1].Plan3[e6].nil))
[mc::step] Complete!
[mc::report] [q: 0 / g: 27] @ 28
```

6.4 Conclusion

Dans ce chapitre, nous avons essayé à la fois de montrer la faisabilité et l'apport de notre approche de modélisation architectural BRS-MAS, dans la modélisation d'architecture SMA à travers la modélisation du système de e-commerce présenté dans ce chapitre. Dans cet exemple illustratif, la modélisation est effectuée en deux parties complémentaires. En premier lieu, nous avons modélisé la structure du système de e-commerce par un bigraphe qui représente les agents intervenant dans l'architecture SMA du système ainsi que les différents liens entre ces agents et leurs répartitions. La deuxième étape consiste, en la modélisation de la partie dynamique ou comportementale du système. A cet effet, un ensemble de règles de réactions a été proposé pour modéliser la reconfiguration des états internes des agents mais également la collaboration entre agents (en modélisant le protocole de négociation FIPA, « contract net protocol ») au niveau SMA. Les règles de réactions définies dans ce chapitre ont été instanciées à partir des méta-règles de réaction définies lors de la présentation de l'approche au niveau des chapitre 4 et 5. Une fois la modélisation terminée, nous sommes passés à la vérification de la spécification architecturale résultante. Ceci a été effectué en utilisant un outil dédié à la vérification de système réactif bigraphique, BigMC. D'abord, un passage de la spécification bigraphique de la partie structurelle et dynamique du système a été effectué vers l'algèbre de terme spécifique au modèle checker BigMC. Ensuite, nous avons défini un ensemble de propriétés (de sûreté et de vivacité)

à vérifier. Le résultat de la vérification sur les propriétés à montré l'absence d'interblocage ainsi que l'absence de violation dans le modèle proposé.

Chapitre 7

Conclusion générale

Sommaire

7.1 Conclusion

Cette thèse s'est concentrée sur la conception des modèles pour les architectures de SMA qui prennent en charge l'interaction dynamique et autonome, et qui permettent de refléter l'évolution des besoins et des objectifs dans un environnement ouvert. La motivation de ces modèles résulte de la nécessité de concevoir des solutions fiables, efficaces qui intègrent la prise en charge des notions de localité et du changement dynamique du SMA et de son environnement, tout en facilitant l'interaction entre les agents dans un environnement ouvert. Le modèle BRS-MAS présenté dans le chapitre 4, utilise le paradigme de l'agent comme outil de conception. Au niveau abstrait, la notion d'agent peut se référer à toute entité autonome participant à une interaction (y compris les personnes). Cependant, tout au long de la thèse, nous avons utilisé des modèles théoriques d'agents et d'interaction pour construire des modèles pour les architectures de SMA. Le principal objectif de notre modèle BRS-MAS n'est pas la conception des entités individuelles qui forment l'architecture mais la conception de l'environnement où ces entités interagissent de manière à trouver un équilibre entre l'autonomie des agents, leurs besoins de coordination et les attentes environnementales au niveau SMA. Ce dernier chapitre traite des résultats de ce travail de thèse, dans la section 7.1, et donne un aperçu sur les travaux futurs, dans la section 7.2.

7.2 Discussion des résultats

Les agents offrent un moyen pour modéliser des systèmes complexes qui comportent des composants multiples et distincts, et sont souvent utilisés comme métaphore pour des entités autonomes et intelligentes [Luck 2005]. Une approche évidente et pragmatique pour démontrer l'applicabilité d'une approche ou d'un modèle serait simplement d'aller de l'avant et de le faire. Cependant, même si les agents sont adéquats pour représenter des entités rationnelles, en raison de leurs autonomie, flexibilité et comportements (réactifs et proactifs), considérer leurs assemblages au niveau SMA est plus que nécessaire. Dans un SMA, le comportement global du système et les aspects collectifs - tels que la stabilité, la prévisibilité et l'engagement envers les objectifs et besoins globaux - doivent être pris en considération. C'est-à-dire que le concept d'interaction et de comportement social des agents revêt une importance capitale lorsque les systèmes multi-agents sont considérés d'un point de vue architectural. Cela conduit à une prise de conscience croissante où les systèmes multi-agents peuvent être mieux compris et développés s'ils sont inspirés par des méthodologies du génie logiciel [Artikis 2001, Castelfranchi 2000, Zambonelli 2001]. Il s'agit, à bien des égards, d'un concept important au sein du domaine de l'ingénierie logicielle orientée agent.

Ce travail de recherche, a préconisé l'utilisation d'architectures de système multi-agent pour concevoir les systèmes d'information d'aujourd'hui, qui doivent maintenant être basés sur des architectures ouvertes, pour accueillir de nouveaux composants et répondre à de nouvelles exigences. Dans le modèle BRS-MAS, les systèmes multi-agents sont munis de moyens qui permettent réellement d'avoir une architecture ouverte et évolutive, pour exploiter les services de nouveaux agents ou remplacer ceux existants.

Au début de cette thèse, nous avons essayé d'identifier les approches et modèles de SMA qui pourraient répondre à nos objectifs sans pour autant découvrir qu'il n'existait aucun modèle prêt à être directement utilisé, ou qui pourrait être considéré comme un point de départ dans notre recherche. Au chapitre 2, nous avons présenté des méthodes de développement de SMA. Ensuite, nous avons discuté des développements actuels de la recherche, soulignant leurs caractéristiques et leurs lacunes en tant que cadres de modélisation de SMA. Comme il n'y avait aucune approche adéquate disponible comme outil de modélisation pour les architectures de SMA et leurs reconfigurations, notre solution était de développer un modèle qui satisfait ces exigences. Le cadre de modélisation BRS-MAS présenté, et largement discuté au chapitre 3, est la réponse du modèle BRS-MAS aux exigences d'autonomie, d'interaction, et d'évolution présentées au chapitre 1. Par conséquent, ce travail vise à abor-

der les problèmes actuels liés au manque de bonnes techniques et notation pour la conception SMA :

- Quelle est l'approche de modélisation la plus adéquate pour gérer la complexité des SMA et de leurs propriétés ?
- Quels sont les prérequis architecturaux et les concepts clés utilisés lors de la modélisation de SMA et leur reconfiguration ?
- Comment avoir une spécification qui soit facilement analysable et vérifiable ?

L'approche adoptée pour répondre à ces préoccupations de conception, a été de développer un SMA de manière systématique en analysant le système en fonction de ses objectifs, et de le concevoir en adoptant le niveau d'abstraction requis pour faciliter son implémentation. Chaque système a son propre cadre dont le bon établissement peut conduire au bon système, et même à une analyse et à une conception appropriée pour l'étendre. En effet, une architecture bien pensée pour un système, fournit l'image complète de ce dernier et offre les lignes directrices pour son développement. De plus, même si notre objectif principal était le développement de solutions pratiques, il est important de développer une théorie formelle pour le cadre BRS-MAS. Le rôle des méthodes formelles est de fournir une description claire et précise de ce qu'un système est censé faire, plutôt que de la formulation de son fonctionnement. Le fait que BRS-MAS ait une sémantique formelle précise, permet l'utilisation de techniques de conception structurée et d'analyse formelle, facilitant le développement, la composition et la réutilisation [Esteva 2001] et pouvant donc servir à guider et à soutenir le concepteur tout en construisant et en affinant un modèle conceptuel. En conséquent, nous avons proposé une approche générique, centrée-architecture basée sur les systèmes réactifs bigraphiques comme fondement formel, pour la spécification et la vérification des architectures des systèmes multi-agents et leur reconfiguration. La modélisation a abordé les dimensions structurelles et dynamiques des architectures des systèmes multi-agents au niveau de l'agent et du SMA.

Au niveau structurel, le modèle permet la conception de la configuration hiérarchique d'agents interactifs (nœuds). En outre, la topologie organisationnelle et l'interconnexion entre les agents sont modélisées séparément, en utilisant deux structures respectivement : le graphe de places et de liens. D'autre part, la dynamique de cette approche est spécifiée par un ensemble de méta-règles de réactions qui régissent l'évolution architecturale et sa reconfiguration. En outre, une vérification formelle de la spécification résultante est effectuée pour vérifier les propriétés bigraphiques comme l'interblocage. Par conséquent, l'approche proposée facilite la détection précoce des défauts de conception au stade de la conception en effectuant une vérification architecturale afin de

faciliter le passage à la mise en œuvre de notre SMA.

7.3 Perspectives

Les recherches futures concernent les questions de mise en œuvre, théoriques et pratiques. Elle sont détaillées dans le reste de cette section.

7.3.1 L'implémentation

La direction principale pour les travaux futurs est évidemment le développement d'outils pratiques pour construire des modèles BRS-MAS. Ces outils dédiés pour la création des architectures, faciliteront la conception, la mise en œuvre et la vérification des systèmes multi-agents. De tels outils devraient être capables de guider le processus d'ingénierie, en suivant la méthodologie de développement, et permettent la spécification et la reconfiguration automatique des architectures de SMA selon le modèle BRS-MAS. De tels outils devraient également soutenir la vérification des modèles BRS-MAS. Dans le suivi de cette thèse, nous prévoyons de développer un environnement pour spécifier un modèle BRS-MAS et générer automatiquement un système multi-agent qui implémente ce modèle. Le système multi-agent qui en résulte devrait respecter la représentation en couche du modèle BRS-MAS (niveau social et agent). L'outil devrait en outre fournir des mécanismes pour l'expression de propriétés telles que la sécurité et la robustesse, pour permettre la construction d'applications du monde réel.

7.3.2 La théorie

À notre avis, le développement des systèmes multi-agents ouverts tend de plus en plus à être inspiré par les méthodologies orienté-organisations. C'est-à-dire l'intégration d'autres concepts lors de la conception des systèmes multi-agents tels que l'organisation, les groupes, les rôles et les normes sociales, etc, tout en séparant la spécification des problèmes sociaux de la conception des agents individuels, en respectant l'exigence d'autonomie de l'agent. Par conséquent, un nouveau domaine de recherche est la description d'un modèle social formel, avec des architectures formelles d'agents. Une telle approche permettra de déterminer les relations sémantiques exactes ainsi que les propriétés du système, et par conséquent des sociétés d'agents ouvertes. Enfin, l'objectif final est de développer une méthodologie claire et exhaustive autour de ces abstractions organisationnelles, pour la conception des architectures de systèmes multi-agents fiables.

Bibliographie

[Aboud 2012] Nour Aboud. *Service-Oriented Integration of Component and Organizational MultiAgent Models*. PhD thesis, Pau, 2012. (Cité en pages et 20.)

[Allen 1998] Robert Allen, Rémi Douence et David Garlan. Specifying and analyzing dynamic software architectures, pages 21–37. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. (Cité en pages 3, 53 et 73.)

[Artikis 2001] A Artikis, L Kamara et J Pitt. *Towards an open agent society model and animation*. In Proceedings of the Agent-Based Simulation II workshop, Passau, pages 48–55, 2001. (Cité en page 106.)

[Bellifemine 2005] Fabio Bellifemine, Federico Bergenti, Giovanni Caire et Agostino Poggi. *JADE-a java agent development framework*. In Multi-Agent Programming, pages 125–147. Springer, 2005. (Cité en page 61.)

[Benzadri 2016] Zakaria Benzadri. *Spécification et Vérification Formelle des Systémes Cloud*. PhD thesis, Université Constantine 2-Abdelhamid Mehri, 2016. (Cité en page 37.)

[Brazier 1997] Frances M. T. Brazier, Barbara M. Dunin-Keplicz, Nick R. Jennings et Jan Treur. *Desire : Modelling Multi-Agent Systems in a Compositional Formal Framework*. International Journal of Cooperative Information Systems, vol. 06, no. 01, pages 67–94, 1997. (Cité en page 3.)

[Bresciani 2004] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia et John Mylopoulos. *Tropos : An agent-oriented software development methodology*. Autonomous Agents and Multi-Agent Systems, vol. 8, no. 3, pages 203–236, 2004. (Cité en pages 30 et 34.)

[Brooks 1991] Rodney A Brooks. *Intelligence without representation*. Artificial intelligence, vol. 47, no. 1-3, pages 139–159, 1991. (Cité en page 16.)

[Bundgaard 2008] Mikkel Bundgaard, Arne John Glenstrup, Thomas Hildebrandt, Espen Højsgaard et Henning Niss. *Formalizing WS-BPEL and higher order mobile embedded business processes in the bigraphical programming languages (BPL) tool*. Rapport technique, Technical Report TR-2008-103, IT University of Copenhagen, 2008. (Cité en page 49.)

[Caire 2001] Giovanni Caire, Wim Coulier, Francisco Garijo, Jorge Gomez, Juan Pavon, Francisco Leal, Paulo Chainho, Paul Kearney, Jamie Stark, Richard Evans*et al.* *Agent oriented analysis using MESSAGE/UML*. In International Workshop on Agent-Oriented Software Engineering, pages 119–135. Springer, 2001. (Cité en page 28.)

[Castelfranchi 2000] Cristiano Castelfranchi. *Engineering social order*. In International Workshop on Engineering Societies in the Agents World, pages 1–18. Springer, 2000. (Cité en page 106.)

[Cherfia 2016] Taha Abdelmoutaleb Cherfia. *Un Framework Basé Bigraphes pour la Conception et l'Analyse des Systèmes Sensibles au Contexte*. PhD thesis, Université Constantine 2-Abdelhamid Mehri, 2016. (Cité en pages , 37, 38 et 55.)

[Conforti 2006] Giovanni Conforti, Damiano Macedonio et Vladimiro Sassone. *BiLog : Spatial logics for bigraphs*. (Under revision), 2006. (Cité en page 37.)

[Cossentino 2005] Massimo Cossentino. *From requirements to code with the PASSI methodology*. Agent-oriented methodologies, vol. 3690, pages 79–106, 2005. (Cité en pages 18, 28 et 34.)

[da Silva 2008] Viviane Torres da Silva. *From the specification to the implementation of norms : an automatic approach to generate rules from norms to govern the behavior of agents*. Autonomous Agents and Multi-Agent Systems, vol. 17, no. 1, pages 113–155, 2008. (Cité en page 23.)

[Damgaard 2006] Troels Christoffer Damgaard et Lars Birkedal. *Axiomatizing binding bigraphs*. Nordic Journal of Computing, vol. 13, no. 1/2, page 58, 2006. (Cité en page 48.)

[Dastani 2014] Mehdi Dastani, Christiaan Floor et John-Jules Ch Meyer. *Programming Agents with Emotions*. In Emotion Modeling, pages 57–75. Springer, 2014. (Cité en page 15.)

[Demazeau 1990] Yves Demazeau et J-P Müller. Decentralized ai, volume 2. Elsevier, 1990. (Cité en pages 2 et 53.)

[Demazeau 1995] Yves Demazeau. *From interactions to collective behaviour in agent-based systems*. In In : Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo. Citeseer, 1995. (Cité en page 19.)

[Dib 2014] Ahmed T. Dib et Zaïdi Sahnoun. *Formal Specification of Multi-Agent System Architecture*. In Proceedings of the 1st International Conference on Advanced Aspects of Software Engineering, ICAASE 2014, Constantine, Algeria, November 2-4, 2014., pages 65–72, 2014. (Cité en page 6.)

[Dib 2015] Ahmed Taki Eddine Dib et Zaïdi Sahnoun. *Model checking of Multi Agent System architectures using BigMC*. In 2015 Federated Conference on Computer Science and Information Systems, FedCSIS 2015, Lódz, Poland, September 13-16, 2015, pages 1717–1722, 2015. (Cité en page 6.)

[Dib 2016] Ahmed Taki Eddine Dib, Kamel Barkaoui et Zaïdi Sahnoun. *Specification and verification of reconfigurable multi-agent system architectures.* Multiagent and Grid Systems, vol. 12, no. 2, pages 105–124, 2016. (Cité en page 6.)

[Dignum 2004a] MV Dignum. A model for organizational interaction : based on agents, founded in logic. SIKS, 2004. (Cité en pages 21 et 34.)

[Dignum 2004b] Virginia Dignum, Javier Vázquez-Salceda et Frank Dignum. *Omni : Introducing social structure, norms and ontologies into agent organizations.* In International Workshop on Programming Multi-Agent Systems, pages 181–198. Springer, 2004. (Cité en pages , 23 et 34.)

[Elsborg 2009] Ebbe Elsborg. *Bigraphs : Modelling, Simulation, and Type Systems.* PhD thesis, PhD thesis, IT University of Copenhagen, 2009. (Cité en page 49.)

[Ernst 2009] Dieter Ernst et Sheri Martin. *The Common Criteria for Information Technology Security Evaluation-Implications for China's Policy on Information Security Standards.* 2009. (Cité en page 91.)

[Esteva 2001] Marc Esteva, Juan-Antonio Rodriguez-Aguilar, Carles Sierra, Pere Garcia et Josep L Arcos. *On the formal specification of electronic institutions.* In Agent mediated electronic commerce, pages 126–147. Springer, 2001. (Cité en page 107.)

[Esteva 2002] Marc Esteva, David De La Cruz et Carles Sierra. *ISLANDER : an electronic institutions editor.* In Proceedings of the first international joint conference on Autonomous agents and multiagent systems : part 3, pages 1045–1052. ACM, 2002. (Cité en pages 22 et 34.)

[Esteva 2004] Marc Esteva, Bruno Rosell, Juan A Rodriguez-Aguilar et Josep Ll Arcos. *Ameli : An agent-based middleware for electronic institutions.* In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1, pages 236–243. IEEE Computer Society, 2004. (Cité en pages 22 et 23.)

[Faulkner 2003] Stéphane Faulkner et Manuel Kolp. *Towards an Agent Architectural Description Language for Information Systems.* In ICEIS (3), pages 59–66. Citeseer, 2003. (Cité en pages 30, 32 et 34.)

[Ferber 1995] Jacques Ferber. Les systèmes multi-agents : vers une intelligence collective. InterEditions, 1995. (Cité en pages 8, 18 et 19.)

[Ferber 1998] Jacques Ferber et Olivier Gutknecht. *A meta-model for the analysis and design of organizations in multi-agent systems.* In Multi Agent Systems, 1998. Proceedings. International Conference on, pages 128–135. IEEE, 1998. (Cité en pages , 21 et 57.)

[Ferber 2003] Jacques Ferber, Olivier Gutknecht et Fabien Michel. *From agents to organizations : an organizational view of multi-agent systems.* In International Workshop on Agent-Oriented Software Engineering, pages 214–230. Springer, 2003. (Cité en pages 21 et 34.)

[Finin 1994] Tim Finin, Richard Fritzson, Don McKay et Robin McEntire. *KQML as an agent communication language.* In Proceedings of the third international conference on Information and knowledge management, pages 456–463. ACM, 1994. (Cité en pages 79 et 93.)

[Fipa 2002] ACL Fipa. *Fipa acl message structure specification.* Foundation for Intelligent Physical Agents, http ://www. fipa. org/specs/-fipa00061/SC00061G. html (30.6. 2004), 2002. (Cité en pages 79 et 93.)

[Franklin 1997] Stan Franklin et Art Graesser. *Is it an Agent, or just a Program ? : A Taxonomy for Autonomous Agents.* Intelligent agents III agent theories, architectures, and languages, pages 21–35, 1997. (Cité en page 8.)

[García-Camino 2005] Andrés García-Camino, Pablo Noriega et Juan A Rodríguez-Aguilar. *Implementing norms in electronic institutions.* In Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pages 667–673. ACM, 2005. (Cité en page 23.)

[Genesereth 1994] Michael R Genesereth et Steven P Ketchpel. *Software agents.* Commun. ACM, vol. 37, no. 7, pages 48–53, 1994. (Cité en page 61.)

[Georgeff 1998] Michael Georgeff, Barney Pell, Martha Pollack, Milind Tambe et Michael Wooldridge. *The belief-desire-intention model of agency.* In International Workshop on Agent Theories, Architectures, and Languages, pages 1–10. Springer, 1998. (Cité en page 16.)

[Gomez-Sanz 2002] Jorge Gomez-Sanz et Juan Pavon. *Meta-modelling in agent oriented software engineering.* Advances in Artificial Intelligence-IBERAMIA 2002, pages 606–615, 2002. (Cité en pages 28 et 34.)

[Grohmann 2007] Davide Grohmann et Marino Miculan. *Directed bigraphs.* Electronic Notes in Theoretical Computer Science, vol. 173, pages 121–137, 2007. (Cité en page 48.)

[Grosz 1996] Barbara J Grosz et Sarit Kraus. *Collaborative plans for complex group action.* Artificial Intelligence, vol. 86, no. 2, pages 269–357, 1996. (Cité en page 14.)

[Halpern 1997] Joseph Y Halpern. *A theory of knowledge and ignorance for many agents.* Journal of Logic and Computation, vol. 7, no. 1, pages 79–108, 1997. (Cité en pages 2 et 53.)

[Hannoun 2000] Mahdi Hannoun, Olivier Boissier, Jaime S Sichman et Claudette Sayettat. *MOISE : An organizational model for multi-agent systems*. In Advances in Artificial Intelligence, pages 156–165. Springer, 2000. (Cité en pages , 24 et 25.)

[Harrington 2002] David Harrington, Bert Wijnen et Randy Presuhn. *An architecture for describing simple network management protocol (SNMP) management frameworks*. 2002. (Cité en pages 9 et 61.)

[Haynes 1998] THOMAS Haynes et SANDIP SEN. *Learning cases to resolve conflicts and improve group behavior*. International Journal of Human-Computer Studies, vol. 48, no. 1, pages 31–49, 1998. (Cité en pages 2 et 53.)

[Højsgaard 2011] Espen Højsgaard et Arne J Glenstrup. *The bpl tool : A tool for experimenting with bigraphical reactive systems*. Bigraphical Languages and their Simulation, page 85, 2011. (Cité en page 49.)

[Hübner 2002a] Jomi Fred Hübner, Jaime Simao Sichman et Olivier Boissier. *A model for the structural, functional, and deontic specification of organizations in multiagent systems*. In Brazilian Symposium on Artificial Intelligence, pages 118–128. Springer, 2002. (Cité en page 19.)

[Hübner 2002b] Jomi Fred Hübner, Jaime Simao Sichman et Olivier Boissier. *Moise+ : towards a structural, functional, and deontic model for mas organization*. In Proceedings of the first international joint conference on Autonomous agents and multiagent systems : part 1, pages 501–502. ACM, 2002. (Cité en pages 24 et 34.)

[Jennings 2000] Nicholas R Jennings. *On agent-based software engineering*. Artificial intelligence, vol. 117, no. 2, pages 277–296, 2000. (Cité en page 16.)

[Jennings 2001] Nicholas R Jennings et Michael Wooldridge. *Agent-oriented software engineering*. Handbook of agent technology, vol. 18, 2001. (Cité en pages , 27, 29, 31 et 56.)

[Jiao 2003] Wenpin Jiao, Minghui Zhou et Qianxiang Wang. *Formal framework for adaptive multi-agent systems*. In IEEE/WIC International Conference on Intelligent Agent Technology, 2003. IAT 2003., pages 442–445, Oct 2003. (Cité en page 2.)

[Kefalas 2005] Petros Kefalas, Ioanna Stamatopoulou et Marian Gheorghe. *A formal modelling framework for developing multi-agent systems with dynamic structure and behaviour*. In International Central and Eastern European Conference on Multi-Agent Systems, pages 122–131. Springer, 2005. (Cité en pages 32 et 34.)

[Kraus 1995] Sarit Kraus, Jonathan Wilkenfeld et Gilad Zlotkin. *Multiagent negotiation under time constraints.* Artificial Intelligence, vol. 75, no. 2, pages 297–345, 1995. (Cité en pages 2 et 53.)

[Krivine 2008] Jean Krivine, Robin Milner et Angelo Troina. *Stochastic bigraphs.* Electronic Notes in Theoretical Computer Science, vol. 218, pages 73–96, 2008. (Cité en page 48.)

[Lacomme 2011] Laurent Lacomme. *Un modele générique pour les organisations dynamiques en univers multi-agent.* PhD thesis, Grenoble, 2011. (Cité en page 20.)

[Luck 2005] Michael Luck, Peter McBurney, Onn Shehory et Steve Willmott. Agent technology : computing as interaction (a roadmap for agent based computing). University of Southampton, 2005. (Cité en pages 8 et 106.)

[Mansutti 2014] Alessio Mansutti, Marino Miculan et Marco Peressotti. *Multi-agent systems design and prototyping with bigraphical reactive systems.* In IFIP International Conference on Distributed Applications and Interoperable Systems, pages 201–208. Springer, 2014. (Cité en page 32.)

[Martin 1999] David L Martin, Adam J Cheyer et Douglas B Moran. *The open agent architecture : A framework for building distributed software systems.* Applied Artificial Intelligence, vol. 13, no. 1-2, pages 91–128, 1999. (Cité en page 61.)

[Milner 2006] Robin Milner. *Pure bigraphs : Structure and dynamics.* Information and computation, vol. 204, no. 1, pages 60–122, 2006. (Cité en pages 37 et 48.)

[Milner 2008] Robin Milner. *Bigraphs and Their Algebra.* Electron. Notes Theor. Comput. Sci., vol. 209, pages 5–19, Avril 2008. (Cité en pages 37 et 41.)

[Milner 2009] Robin Milner. The space and motion of communicating agents. Cambridge University Press, New York, NY, USA, 1st édition, 2009. (Cité en pages 37, 39, 45 et 73.)

[Müller 2011] Jörg P Müller et Markus Pischel. *The agent architecture inteRRaP : Concept and application.* 2011. (Cité en page 16.)

[Müller 2014] Jörg P. Müller et Klaus Fischer. Application impact of multi-agent systems and technologies : A survey, pages 27–53. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. (Cité en pages 2, 17 et 53.)

[Navarro 2013] Laurent Navarro, Vincent Corruble, Fabien Flacher et Jean-Daniel Zucker. *A flexible approach to multi-level agent-based simulation*

with the mesoscopic representation. In Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, pages 159–166. International Foundation for Autonomous Agents and Multiagent Systems, 2013. (Cité en pages et 58.)

[Norman 1997] Timothy Norman, Nick Jennings, Peyman Faratin et E Mamdani. *Designing and implementing a multi-agent architecture for business process management.* Intelligent Agents III Agent Theories, Architectures, and Languages, pages 261–275, 1997. (Cité en page 61.)

[Nwana 1996] Hyacinth S Nwana. *Software agents : An overview.* The knowledge engineering review, vol. 11, no. 03, pages 205–244, 1996. (Cité en page 8.)

[Oquendo 2004] Flavio Oquendo. *πADLL : An Architecture Description Language Based on the Higher-order Typed π-calculus for Specifying Dynamic and Mobile Software Architectures.* SIGSOFT Softw. Eng. Notes, vol. 29, no. 3, pages 1–14, Mai 2004. (Cité en pages 3 et 73.)

[Panait 2005] Liviu Panait et Sean Luke. *Cooperative multi-agent learning : The state of the art.* Autonomous agents and multi-agent systems, vol. 11, no. 3, pages 387–434, 2005. (Cité en page 14.)

[Park 2005] Sooyong Park et Vijayan Sugumaran. *Designing multi-agent systems : a framework and application.* Expert Systems with Applications, vol. 28, no. 2, pages 259–271, 2005. (Cité en pages 30 et 34.)

[Parnas 1972] David Lorge Parnas. *On the criteria to be used in decomposing systems into modules.* Communications of the ACM, vol. 15, no. 12, pages 1053–1058, 1972. (Cité en page 58.)

[Perrone 2012] Gian Perrone, Søren Debois et Thomas T Hildebrandt. *A model checker for bigraphs.* In Proceedings of the 27th Annual ACM Symposium on Applied Computing, pages 1320–1325. ACM, 2012. (Cité en pages , 49, 92 et 96.)

[Poslad 2000] Stefan Poslad, Phil Buckle et Rob Hadingham. *The FIPA-OS agent platform : Open source for open standards.* In proceedings of the 5th international conference and exhibition on the practical application of intelligent agents and multi-agents, volume 355, page 368, 2000. (Cité en page 8.)

[Rao 1995] Anand S Rao, Michael P Georgeff*et al. BDI agents : From theory to practice.* In ICMAS, volume 95, pages 312–319, 1995. (Cité en pages 2, 9, 53, 61 et 74.)

[Ricci 2011] Alessandro Ricci et Andrea Santi. *Agent-oriented computing : Agents as a paradigm for computer programming and software development.* In Proc. of the 3rd Int Conf. on Future Computational Tech-

nologies and Applications. Wilmington : Xpert Publishing Services, pages 42–51. Citeseer, 2011. (Cité en page 17.)

[Rihawi 2014] Omar Rihawi. *Modelling and simulation of distributed large scale situated multi-agent systems*. PhD thesis, Lille 1, 2014. (Cité en page 57.)

[Rosenschein 1994] Jeffrey S Rosenschein et Gilad Zlotkin. Rules of encounter : designing conventions for automated negotiation among computers. MIT press, 1994. (Cité en page 14.)

[Rosenschein 1998] Jeffrey S Rosenschein et Gilad Zlotkin. Rules of encounter. MIT Press, 1 édition, 1998. (Cité en pages 2 et 53.)

[Russell 1995] Stuart Russell, Peter Norvig et Artificial Intelligence. *A modern approach*. Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs, vol. 25, page 27, 1995. (Cité en page 8.)

[Sahli 2017] Hamza Sahli. *Modélisation des Systèmes Élastiques Cloud : vers la Vérification Formelle de leur Comportement*. PhD thesis, Université Constantine 2-Abdelhamid Mehri, 2017. (Cité en pages , 37, 42, 43, 44, 46 et 47.)

[Sandhlom 1997] Tuomas W. Sandhlom et Victor R.T Lesser. *Coalitions among computationally bounded agents*. Artificial Intelligence, vol. 94, no. 1-2, pages 99–137, 1997. (Cité en pages 2 et 53.)

[Sandholm 1993] Tuomas Sandholm. *An implementation of the contract net protocol based on marginal cost calculations*. In AAAI, volume 93, pages 256–262, 1993. (Cité en pages 79 et 92.)

[Scalabrin 1996] E Scalabrin, L Vandenberghe, Hilton de Azevedo et J Barthès. *A generic model of cognitive agent to develop open systems*. Advances in Artificial Intelligence, pages 61–70, 1996. (Cité en page 61.)

[Seghrouchni 2004] Amal El Fallah Seghrouchni et Alexandru Suna. *Himalaya framework : Hierarchical intelligent mobile agents for building large-scale and adaptive systems based on ambients*. In International Workshop on Massively Multiagent Systems, pages 202–216. Springer, 2004. (Cité en pages 32 et 34.)

[Sevegnani 2015] Michele Sevegnani et Muffy Calder. *Bigraphs with sharing*. Theoretical Computer Science, vol. 577, pages 43–73, 2015. (Cité en page 49.)

[Sevegnani 2016] Michele Sevegnani et Muffy Calder. *BigraphER : rewriting and analysis engine for bigraphs*. In International Conference on Computer Aided Verification, pages 494–501. Springer, 2016. (Cité en page 49.)

[Shaw 1996] Mary Shaw et David Garlan. Software architecture : Perspectives on an emerging discipline. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. (Cité en page 3.)

[Shehory 1998] Onn Shehory et Sarit Kraus. *Methods for task allocation via agent coalition formation.* Artificial intelligence, vol. 101, no. 1-2, pages 165–200, 1998. (Cité en page 14.)

[Shehory 2014a] Onn Shehory et Arnon Sturm. *A brief introduction to agents.* In Agent-Oriented Software Engineering, pages 3–11. Springer, 2014. (Cité en pages , 10 et 12.)

[Shehory 2014b] Onn Shehory et Arnon Sturm. Multi-agent systems : A software architecture viewpoint, pages 57–78. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. (Cité en pages 2 et 53.)

[Shoham 1992] Yoav Shoham et Moshe Tennenholtz. *On the synthesis of useful social laws for artificial agent societies (preliminary report).* In AAAI, pages 276–281, 1992. (Cité en pages 2 et 53.)

[Shoham 1993] Yoav Shoham. *Agent-oriented programming.* Artificial intelligence, vol. 60, no. 1, pages 51–92, 1993. (Cité en page 8.)

[Specification 2017] FIPA Inform Communicative Act Specification. *Foundation for Intelligent Physical Agents, 2000*, 2017. (Cité en page 13.)

[Sycara 1998] Katia P. Sycara. *The many faces of agents.* AI magazine, vol. 19, no. 2, pages 11–12, 1998. (Cité en page 56.)

[Tambe 1997] Milind Tambe. *Agent architectures for flexible.* In Proc. of the 14th National Conf. on AI, USA : AAAI press, pages 22–28, 1997. (Cité en page 14.)

[Vázquez-Salceda 2003] Javier Vázquez-Salceda et Frank Dignum. *Modelling electronic organizations.* In International Central and Eastern European Conference on Multi-Agent Systems, pages 584–593. Springer, 2003. (Cité en page 23.)

[Wittig 1994] T Wittig, Nick R Jennings et EH Mamdani. *ARCHON : framework for intelligent co-operation.* Intelligent Systems Engineering, vol. 3, no. 3, pages 168–179, 1994. (Cité en page 61.)

[Wooldridge 1995] Michael Wooldridge et Nicholas R Jennings. *Intelligent agents : Theory and practice.* The knowledge engineering review, vol. 10, no. 02, pages 115–152, 1995. (Cité en page 16.)

[Wooldridge 1998] Michael J Wooldridge. Agent technology : foundations, applications, and markets. Springer Science & Business Media, 1998. (Cité en page 13.)

[Wooldridge 2000a] Michael Wooldridge, Nicholas R Jennings et David Kinny. *The Gaia methodology for agent-oriented analysis and design.* Autonomous Agents and multi-agent systems, vol. 3, no. 3, pages 285–312, 2000. (Cité en pages 16, 26 et 34.)

[Wooldridge 2000b] Michael J Wooldridge. Reasoning about rational agents. MIT press, 2000. (Cité en page 74.)

[Wooldridge 2009] Michael Wooldridge. An introduction to multiagent systems. John Wiley & Sons, 2009. (Cité en pages , 8, 19, 58 et 73.)

[Wooldridge 2013] M Wooldridge. *Intelligent Agents, Multiagent Systems, Ed. by G. Weiss*, 2013. (Cité en page 13.)

[Xu 2003] Haiping Xu et Sol M. Shatz. *A framework for model-based design of agent-oriented software.* IEEE Transactions on software engineering, vol. 29, no. 1, pages 15–30, 2003. (Cité en page 3.)

[Yu 2006] Zhenhua Yu et Yuanli Cai. *Object-oriented Petri nets based architecture description language for multi-agent systems.* IJCSNS, vol. 6, no. 1, pages 123–131, 2006. (Cité en pages 30, 32 et 34.)

[Zambonelli 2001] Franco Zambonelli, Nicholas R Jennings et Michael Wooldridge. *Organisational abstractions for the analysis and design of multi-agent systems.* In Agent-oriented software engineering, pages 235–251. Springer, 2001. (Cité en page 106.)

[Zambonelli 2003] Franco Zambonelli, Nicholas R Jennings et Michael Wooldridge. *Developing multiagent systems : The Gaia methodology.* ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 12, no. 3, pages 317–370, 2003. (Cité en page 26.)

[Zhang 2010] Pengcheng Zhang, Henry Muccini et Bixin Li. *A classification and comparison of model checking software architecture techniques.* Journal of Systems and Software, vol. 83, no. 5, pages 723–744, 2010. (Cité en page 92.)