

وزارة التعليم العالي و البحث العلمي

FACULTE DES SCIENCES DE L'INGENIEUR

BADJI MOKHTAR-ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR-ANNABA



جامعة باجي مختار - عنابة

DEPARTEMENT D'INFORMATIQUE

ANNEE 2018-2019

THESE

Présentée en vue de l'obtention du diplôme de **Doctorat en Sciences**

Un Modèle Organisationnel pour La Conception des Systèmes Multi-Agents Temps Réel

Option

Intelligence Artificielle

Par

LAOUADI Mohamed Amin

Directeur de Thèse

Mr. MOKHATI Farid

Professeur Université d'Oum El Bouaghi

Co-directrice de Thèse

M^{me}. SERIDI-BOUCHELAGHEM Hassina Professeur Université Badji Mokhtar- Annaba

Soutenue publiquement le 20/01/2019

DEVANT LE JURY

Président : ATIL Fadila

Professeur

Université Badji Mokhtar -Annaba

Examineurs: GUERRAM Tahar Maitre de Conférences A Université d'Oum El Bouaghi

HAMRI Salah

Maitre de Conférences A Université d'Oum El Bouaghi

Je dédie cette Thèse

À

Mes Très chers Parents

Ma bien aimée épouse & Mon petit Prince *Raid*

Mes très chers Frères & Sœur

Ma belle Soeur

Ma belle Famille

Et à Tous mes proches & Amis

Laouadi Mohamed Amin

Remerciements

Il est toujours délicat de remercier l'ensemble des personnes qui ont contribué à l'aboutissement de ce travail de recherche. Que ceux qui ne sont pas mentionnés ne m'en tiennent pas rigueur.

Je tiens à exprimer mes profonds remerciements à Mr. MOKHATI Farid, Professeur à l'Université d'Oum-El Bouaghi, pour son encadrement, sa disponibilité, sa confiance et pour le soutien qu'il a su m'accorder. Son expérience et ses compétences scientifiques ont toujours été sources d'enrichissement me permettant de mener à bien ce travail de recherche.

Je désire sincèrement remercier Mme. SERIDI- BOUCHELAGHEM Hassina, Professeur à l'Université Badji Mokhtar-Annaba, pour avoir eu le privilège d'être son étudiant. Ses qualités scientifiques, ses conseils et son dynamisme ont contribué à l'aboutissement de ce travail de recherche. Je la remercie pour la grande confiance qu'elle m'a accordée.

Je tiens à remercier chaleureusement Mr. GUERRAM Tahar Maître de Conférences 'A' à l'Université d'Oum El Bouaghi et Mr. HAMRI Salah Maître de Conférences 'A' à l'Université d'Oum El Bouaghi, pour avoir acceptés de rapporter la thèse et de m'avoir apportés leur regard sur notre travail.

Je remercie Mme. ATIL Fadila Professeur à l'Université Badji Mokhtar-Annaba de m'avoir honoré en acceptant de présider mon jury.

Enfin, de manière plus personnelle, je souhaite remercier ma famille pour leur soutien inconditionnel. Merci pour m'avoir encouragé, supporté et pour avoir accepté tant de sacrifices durant ces années.

الملخص

حالياً، تعتبر هندسة البرمجيات الموجهة للأعوان، مجال بحثي نشط للغاية. لسنوات عديدة، لدى مصممي الأنظمة متعددة الأعوان منهجيات تطوير ولغة نمذجة دون أن تعكس القيود الزمنية المختلفة التي قد يكون عليها الأنظمة متعددة الأعوان ذات الوقت الحقيقي. في الواقع، ليس من السهل تصوّر مفاهيم الأنظمة متعددة الأعوان ذات الوقت الحقيقي باستخدام أساليب هندسة البرمجيات التقليدية. وبناء على ذلك، فإن قضية بحثية حرجية لمجتمع عامل الوقت الحقيقي تمثلت في تعريف النموذج التنظيمي المناسب لتحليل وتصميم خصائصها الأساسية. لقد كان التقدم في هذا المجال بطيئاً ولكنه ثابت بشكل عام. تعتبر معظم المقترحات المنظمة ككيان لها غرض محدد أو هدف عام دون الأخذ بعين الاعتبار التقييد الزمني لعامل الوقت الحقيقي. ثم يستخدمون مجموعة من المفاهيم التنظيمية، على سبيل المثال، الأدوار، الأعراف، المجموعات، التفاعلات، وما إلى ذلك؛ للسيطرة على سلوكيات الوكلاء وتعديلها فيما يتعلق بالهدف العام للمنظمة.

في رأينا، هذه الرؤية للتنظيم غير كاملة. نقترح تمديدها بقيد زمني من خلال توسيع مخططات عون لغة النموذج الموحدة [Lao13]. وعلى وجه الخصوص، نعتقد أن مفهوم التقييد الزمني الذي يجب على وكيل الوقت الحقيقي احترامه أثناء دورة حياته أمر مهم للغاية ليلم إدراجه في المنظمة.

نهجنا يأخذ تصوير الشخصيات الاجتماعية جنباً إلى جنب مع وكيل القيود الزمنية باعتبارها جوهر نمذجة الأنظمة متعددة الأعوان ذات الوقت الحقيقي ويقدم أول فكرة عن النموذج التنظيمي لتحديد الأنظمة متعددة الأعوان ذات الوقت الحقيقي. إنه يسمح بتمثيل الجانب الوظيفي والجوانب الثابتة والجوانب الديناميكية لـ الأنظمة متعددة الأعوان ذات الوقت الحقيقي بناءً على المنظور التنظيمي. ومع ذلك، فإن إضفاء الطابع الرسمي على الجوانب التنظيمية لـ الأنظمة متعددة الأعوان ذات الوقت الحقيقي هو في رأينا طريقة مهمة لكل من أنشطة التحليل والتصميم. علاوة على ذلك، يتطلب تصميم الأنظمة متعددة الأعوان ذات الوقت الحقيقي المشاركة باللغات الرسمية. من بين هذه اللغات: ربما تكون آر تي مود [Olv07] أفضل اللغات المعروفة والأكثر استخداماً لكل من البرمجة الوظيفية. علاوة على ذلك، آر تي مود قوية للغاية من حيث المواصفات والتحقق من الأنظمة المترامنة، مما يجعلها مرشحاً جيداً للمواصفات والتحقق من الأنظمة متعددة الأعوان ذات الوقت الحقيقي. مع العلم أنه لا يوجد حالياً أي عمل يطبق آر تي مود على المواصفات التنظيمية الرسمية للأنظمة متعددة الأعوان ذات الوقت الحقيقي وعلى كل من تطبيقات الوقت الحقيقي.

في الواقع، يمثل هذا الإطار التنظيمي الرسمي نهجاً هاماً لنمذجة الأنظمة متعددة الأعوان ذات الوقت الحقيقي، وهو امتداد حديث يجمع بين مخططات عون لغة النموذج الموحدة الموسعة [Lao13]، مع مواصفات آر تي مود [Lao14] لدعم المواصفات الرسمية لـ الأنظمة متعددة الأعوان ذات الوقت الحقيقي التنظيمية. في هذا الإطار، يتم توحيد نقاط القوة في كلا النهجين، ولكنها لا تزال تقتصر إلى المزيد من التفاصيل التنظيمية التي يجب أخذها بعين الاعتبار. لذلك، تم وصف منهج قائم على إعادة كتابة المنطق لنماذج النموذج الوكيل [Fer03a] و نموذج مويس+ [Hub02a] لمؤسسة أنظمة متعددة العوامل لتكون قادرة على التفكير، من ناحية، حول النموذج التنظيمي، ومن ناحية أخرى، حول دلالاتها العامة. علاوة على ذلك، تم بحث إمكانية تضمين البعد الزمني للتعامل مع العوامل المعيارية للقيود الزمنية في نموذج مويس+ لتحديد نموذج ملموس لنظام الأنظمة متعددة الأعوان ذات الوقت الحقيقي من خلال لغة مود في الوقت الفعلي. سمح لنا هذا بأن يكون لدينا في جانب واحد نموذجاً تنظيمياً يسهل تصميم الأنظمة متعددة الأعوان ذات الوقت الحقيقي وعلى الجانب الآخر إطاراً للمواصفات رسمياً يوفر دعماً أفضل لعمليات التحقق الخاصة بهم. من أجل التحقق من صحة هذا العمل، قمنا بتطبيق عملية الترجمة لمحاكاة المواصفات الرسمية مع سيناريوهات التفاعل المحددة مسبقاً، وفي حالتنا اخترنا مثالين، أحدهما يتعامل مع إدارة السلاسل اللوجستية والآخر مع إدارة المؤتمرات.

الكلمات الرئيسية : النموذج التنظيمي؛ الأنظمة متعددة الأعوان ذات الوقت الحقيقي؛ مواصفات رسمية؛ عون لغة النموذج الموحدة؛ آر تي مود؛ منطق إعادة الكتابة؛ إدارة السلاسل اللوجستية؛ نظام إدارة المؤتمرات.

Abstract

Currently, Agent Oriented Software Engineering (AOSE) is a very active research domain. For many years, MAS designers have development methodologies and modeling language without reflects the different temporal restrictions that RT-MAS may have. Indeed, it is not easy to conceptualize RT-MAS concepts using conventional agent software engineering approaches. Consequently, a critical research issue for the real time agent community has been the definition of suitable organizational model for analyzing and designing their main properties. The progress in this area has been slow but generally steady. Most of the proposals consider an organization as an entity that has some concrete purpose or global objective without taking in to account the temporal restriction of real time agent. Then they use a set of organizational concepts, e.g. roles, norms, groups, interactions, etc; to control and modify the dynamics or agents behaviors with regard to the global objective of the organization.

In our opinion, this vision of organization is incomplete. We propose to extend them with temporal constraint by extending Agent UML (AUML) diagrams [Lao13]. In particular, we believe that the concept of temporal restriction that real time agent must respect during their life cycle is very important to be included in organization.

Our approach takes the depiction of social characters combined with agent temporal restriction as the core of RTMAS modeling and presents the first idea of an organizational model to specify RTMAS. It allows representing functional aspect, static aspects, and dynamic aspects of RTMAS based on organizational perspective. However formalizing the organizational aspects of RT-MAS is in our opinion, an importance way for both analysis and design activities. Furthermore, the RT-MAS design requires the involvement with formal languages. Among these languages: Real Time Maude (RT-Maude) [Olv07] is probably the best known and most widely used languages for both functional and object oriented programming. Furthermore, RT-Maude is very powerful in terms of specification, validation and verification of concurrent systems, making it a good candidate for specification and validation of RT-MAS. Knowing that, there is currently no work applying RT-Maude to RT-MAS formal organizational specification and both to real time applications.

In fact, this formal organizational framework represents an important approach for modeling RT-MAS, it is a recent extension which combines the AUML extended diagrams [Lao13], with RT-Maude specification [Lao14] to support the formal specification of RT-MAS organizational requirements. In this framework, the strengths of both approaches are unified, but it still lacks more organizational details to be considered. So, a rewriting logic based approach is described for prototyping the Agent-Group-Role (AGR) [Fer03a] and Model of Organization for multi-agent SystEms (Moise+) [Hub02a] models for multi-agent systems organization to be able to reason, on the one hand, about the organizational model, and on the other hand, about its general semantics. Furthermore, the possibility of including a temporal dimension to deal with agents temporal constraints issues was investigated in Moise+ model to specify the concrete model of Real-Time Multi-Agent System through Real-Time Maude language. This allowed us to have on one side an organizational model facilitating the design of the RTMAS and on the other side a formal specification framework providing providing better support for their verification processes. For the validation of this work we applied the translation process to simulate the formal specifications with predefined interaction scenarios, in our case we chose two examples, one dealing with supply chain management and the other with conference management.

Keywords: Organizational model; Real-Time Multi-Agent System ; Formal Specification; Agent UML; Real-Time Maude; Rewriting Logic; Supply Chain Management; Conference Management system.

Résumé

Actuellement, l'ingénierie orientée agent est un domaine de recherche très actif. Depuis de nombreuses années, les concepteurs de Système Multi-Agents (SMA) ont des méthodologies de développement et des langages de modélisation qui ne reflètent pas les différentes contraintes temporelles qu'un Système Multi-Agents Temps Réel (SMATR) peut avoir. En effet, il n'est pas facile de conceptualiser les concepts SMATR en utilisant des approches d'ingénierie logicielle conventionnelles. Par conséquent, un problème de recherche critique pour la communauté des agents temps réel est celui de la définition d'un modèle organisationnel approprié pour l'analyse et la conception de leurs propriétés principales. Les progrès dans ce domaine ont été lents mais généralement stables. La plupart des propositions considèrent une organisation comme une entité qui a un but concret ou un objectif global sans prendre en compte les restrictions temporelles des agents temps réel du système. Elles utilisent un ensemble de concepts organisationnels, comme rôles, normes, groupes, interactions, etc. pour contrôler et modifier les comportements des agents par rapport à l'objectif global de l'organisation.

À notre avis, cette vision de l'organisation est incomplète. Nous proposons de les étendre pour tenir compte des contraintes temporelles en étendant les diagrammes Agent UML (AUML) [Lao13]. En particulier, nous croyons que le concept de restriction temporelle qu'un agent temps réel doit respecter au cours de son cycle de vie est très important pour être inclus dans l'organisation.

Notre approche illustre bien la représentation des concepts organisationnels conjointement avec les restrictions temporelles des agents comme la base de la modélisation du SMATR, proposant ainsi un nouveau modèle organisationnel pour spécifier les SMATR. Elle permet de représenter les aspects fonctionnels, statiques et dynamiques du système suivant la perspective organisationnelle. Cependant, formaliser les aspects organisationnels de SMATR est, à notre avis, un moyen important pour les activités d'analyse et de conception. De plus, la conception de SMATR nécessite l'implication de langages formels. Parmi ces langages: Real Time Maude (RT-Maude) [Olv07] est un bon candidat pour la spécification de SMATR car il est très puissant en termes de spécification, de validation et de vérification formel des systèmes concurrents orientés objet. Sachant qu'il n'y a pas de travail qui applique RT-Maude aux spécifications organisationnelles formelles de SMATR.

En effet, le framework organisationnel formel obtenu représente une extension récente qui combine les diagrammes AUML étendus [Lao13], avec la spécification RT-Maude [Lao14] pour supporter la spécification formelle de SMATR organisationnel. Dans ce framework, les points forts des deux approches sont unifiés mais il manque encore plus de détails organisationnels à prendre en considération. Pour palier à cela nous avons appliqué notre approche basée sur la logique de réécriture pour prototyper les modèles Agent-Groupe-Rôle (AGR) [Fer03a] et Model of Organization for multi-agent SystEms (Moise+) [Hub02a] de SMA centré organisation afin de pouvoir raisonner, d'une part, sur le modèle organisationnel et d'autre part sur sa sémantique générale. Toutefois, nous avons étudié la possibilité d'inclure une dimension temporelle pour traiter les problèmes de contraintes temporelles dans le modèle Moise+ afin de spécifier le modèle concret du SMATR en utilisant le langage RT-Maude ce qui nous a permis d'avoir d'un côté un modèle organisationnel facilitant la conception des SMATR et de l'autre côté un framework de spécification formelle offrant un meilleur support à leurs processus de vérification. Pour la validation de ce travail nous avons appliqué le processus de translation pour simuler les spécifications formelles avec des scénarios d'interaction prédéfinis, dans notre cas on a choisi deux exemples l'un portant sur la gestion de chaîne logistique et l'autre concernant la gestion de conférence.

Mots clés : Modèle organisationnel ; Système Multi-Agents temps réel; Spécification formelle; Agent UML; Real Time Maude ; Logique de réécriture ; Gestion de chaîne logistique; Système de Gestion de conférence.

Table des matières

Introduction Générale

1. CONTEXTE DE LA RECHERCHE ET ELEMENTS DE PROBLEMATIQUE	1
2. MOTIVATIONS & OBJECTIFS SCIENTIFIQUES	3
3. DEMARCHE SUIVIE & CONTRIBUTIONS	5
4. ORGANISATION DU MANUSCRIT	6

Chapitre 1 : Les Systèmes Multi-Agents Temps Réel 09

Introduction	10
1. Les Systèmes Multi-Agents	11
1.1. Le Concept d'Agents	11
1.2. Agent versus objet	12
1.3. Caractéristiques des agents	13
1.4. Classification des agents	13
1.5. Approche Voyelles	14
1.5.1. Agent	15
1.5.2. Environnement	15
1.5.3. Interaction	15
1.5.4. Organisation	17
1.6. Points Forts & Faibles du Paradigme Orienté Agent	18
1.6.1. Avantages des systèmes multi-agents	19
1.6.2. Difficultés de l'approche orientée-agent	20
1.7. Domaines d'application des systèmes multi-agents	21
2. Les Systèmes Multi-Agent Temps Réel	22
2.1. Définition	23
2.2. Les Systèmes Temps Réel	23
2.2.1. Définition	23
2.2.2. Les Contraintes Temps Réel	24
2.2.2.1. Contraintes temps réel associées aux actions	24
2.2.2.2. Contraintes temps réel associées aux événements	24
2.2.2.3. Contraintes temps réel associées aux données	25
2.2.3. Les Caractéristiques d'un Système Temps Réel	25
2.3. Mise en Œuvre du Temps dans les SMA	26
3. Conclusion	28

Chapitre 2 : La Vue Organisationnelle dans Les Systèmes Multi-Agents	29
Introduction	30
1. Concepts de Base	32
2. Les Méthodologies de développement des SMA	33
2.1. Les Méthodologies Orientées Agent	33
2.2. Les Méthodologies Orientées Organisation	34
3. Les Caractéristiques Organisationnelles	34
3.1. Les Topologies de l'Organisation	35
3.1.1. Hiérarchies	35
3.1.2. Holarchies	35
3.1.3. Coalitions	35
3.1.4. Equipes (Teams)	36
3.1.5. Congrégations	36
3.1.6. Sociétés	36
3.1.7. Fédérations	37
3.1.8. Les Structures de Type Marché	37
3.2. L'Emergence	37
3.3. Réorganisation et Auto-Organisation	38
4. Une étude Comparative des Modèles Organisationnels SMA	39
4.1. Modèles Organisationnels	39
4.1.1. Modèles d'analyse au sein de GAIA	39
4.1.2. Modèle Agent-Groupe-Rôle	41
4.1.3. Moise+	43
4.1.4. INGENIAS	44
4.1.5. RT-Message	46
4.2. Synthèse	47
4.3. Etude Comparative	49
5. Conclusion	52
Chapitre 3 : Langages de Spécification et Conception des Systèmes Multi-Agents Temps Réel	54
Introduction	55
1. Langages de Modélisation orientée agent	56
1.1. Unified Modeling Language (UML)	57
1.2. Agent UML	58
1.2.1. Rappel	58

1.2.2. Les Principales Représentations d'Agent UML	59
1.2.2.1. Les Diagrammes de Cas D'utilisation AUML	60
1.2.2.2. Les Diagrammes de Classe AUML Etendus	60
1.2.2.3. Les Diagrammes de Protocole AUML Etendus	62
1.2.2.4. Les Diagrammes d'Etat Transition AUML	62
2. Langages de Spécification Formelle pour SMATR	63
2.1. Approches Formelles	64
2.2. Maude	66
2.2.1. Fondement de Maude : Logique de Réécriture	66
2.2.2. Structure de Maude	68
2.2.2.1. Caractéristiques de Maude	69
2.2.2.2. Concepts de base	69
2.2.2.3. Les Modules Fonctionnels	71
2.2.2.4. Les Modules Systèmes	72
2.2.2.5. Les Modules Orientés-Objet	73
2.2.3. Extension Maude : Real Time-Maude	75
2.2.3.1. Les Modules Temporisés (Timed modules)	76
2.2.3.2. Les Modules Temporisés Orientés-Objet (Timed Object-Oriented Modules)..	79
2.2.4. Vérification de Modèles	80
3. Synthèse	82
4. Conclusion	85
Chapitre 4 : Un Framework Formel pour la Spécification des SMA centré Organisation	86
Introduction	87
1. Formalisation du Modèle Agent-Group-Rôle	89
1.1. Aperçu du Framework Formel	89
1.2. Architecture du Framework Formel	91
2. Etude de Cas: Gestion de chaîne logistique	95
2.1. Description	95
2.2. Modèle organisationnel d'agent pour la gestion de chaîne logistique	97
2.3. Modélisation AUML de la gestion de chaîne logistique	98
2.3.1. Diagramme de Classe conceptuel de la gestion de chaîne logistique	98
2.3.2. Diagramme de Cas d'utilisation de la gestion de chaîne logistique	99
2.3.3. Diagramme de protocole de la gestion de chaîne logistique	99
2.4. Application du Processus de Translation	100
2.5. Validation de la description générée	102
3. Discussion	104
4. Conclusion	105

Chapitre 5 : Vers un Modèle Organisationnel pour la Spécification des SMATR	107
Introduction	108
1. Modèle Organisationnel pour les SMATR	109
1.1. Méta-Modèle Organisationnel	110
1.2. Définition des rôles d'organisation d'agent avec les diagrammes de cas d'utilisation temporelles AUML centré organisation	111
1.3. Modélisation statique d'organisation d'agents avec les diagrammes de classe temporel AUML centré organisation	112
1.4. Modélisation des comportements dynamique d'organisation d'agent	113
2. Spécification formelle du modèle organisationnel de SMATR	113
2.1. Processus de Translation	114
3. Discussion	117
4. Conclusions	118
Chapitre 6 : Formalisation du Modèle Organisationnel RT-Moise+	119
Introduction	120
1. Méta-modèle Organisationnel RT-Moise+	121
2. Spécification Formelle de RT-Moise+	123
2.1. Architecture du Framework proposé	123
3. Cas d'Etude: Système de Gestion de Conférence (SGC)	129
3.1. Description	129
3.2. Décomposition selon RT-Moise+ du Système de Gestion de Conférence	130
3.2.1. Spécification Structurelle (SS)	130
3.2.2. Spécification Fonctionnelle(SF)	131
3.3. Modélisation AUML du Système de Gestion de Conférence	133
3.3.1. Diagramme de classe du Système de Gestion de Conférence	133
3.3.2. Diagramme de cas d'utilisation du Système de Gestion de Conférence	134
3.3.3. Diagramme de Communication du Système de Gestion de Conférence	135
3.3.4. Diagrammes d'état transition du Système de Gestion de Conférence	135
3.3.5. Diagramme de Protocole du Système de Gestion de Conférence	137
3.3.6. Diagrammes de temps du Système de Gestion de Conférence	138
3.4. Application du Processus de Translation	140
3.5. Validation de la Description Générée	143
3.6. Vérification de la Description Générée	145
3.6.1. Propriétés du comportement individuel	145
3.6.2. Vérification des propriétés	147
4. Conclusion	148

Conclusion & Perspectives	150
1. Bilan	151
2. Perspectives	154
Bibliographie	156
Liste des Productions Scientifiques	176

Liste des Figures

Figure 1.1. Interaction entre un agent et son environnement [Tra07]	11
Figure 1.2. L'approche Voyelles [Tra07]	14
Figure 1.3. Caractérisation d'un SMA [Zam00].....	18
Figure 2.1. Topologie à structure de hiérarchie.....	35
Figure 2.2. Topologie à structure de Holarchie.....	35
Figure 2.3. Topologie à structure de coalition.....	36
Figure 2.4. Topologie à structure d'équipe.....	36
Figure 2.5. Topologie à structure de congrégation.....	36
Figure 2.6. Topologie à structure de société.....	36
Figure 2.7. Topologie à structure de fédération.....	37
Figure 2.8. Topologie à structure de marché.....	37
Figure 2.9. Méta-modèle de Gaia étendu [Aza07].....	41
Figure 2.10. Topologie organisationnelle supportée dans Gaia étendu	41
Figure 2.11. Modèle agent [Gat07a].....	42
Figure 2.12. Méta-modèle d'Agent Groupe Rôle [Fer03a].....	42
Figure 2.13. Topologie organisationnelle supportée dans AGR.....	43
Figure 2.14. Méta-modèle de Moise ⁺ [Dav11].....	44
Figure 2.15. Topologie organisationnelle supportée dans Moise ⁺	44
Figure 2.16. Un métamodèle Ingenias résumé [Aza07].....	46
Figure 2.17. Topologie organisationnelle supportée dans INGENIAS	46
Figure 3.1. Diagramme de cas d'utilisation d'un agent client.....	60
Figure 3.2. Les branchements définis dans AUML	62
Figure 3.3. Exemple d'un événement déclenchant une transition [Gag07].....	63
Figure 3.4. Visualisation des règles d'inférence d'une théorie de réécriture [Gag07].....	68
Figure 3.5. Le module système BANK-ACCOUNT.	74
Figure 3.6. Le module orienté-objet BANK-ACCOUNT.....	75
Figure 3.7. L'exécution du module BANK-ACCOUNT.....	75
Figure 3.8. Le module temporisé DISCRETE-CLOCK-1.....	77
Figure 3.9. Le module temporisé DISCRETE-BROKEN-CLOCK-1.....	78
Figure 3.10. Le module temporisé DISCRETE-CLOCK-24-2.....	78
Figure 3.11. Le module temporisé DISCRETE-CLOCK-24.....	78
Figure 3.12. Le module temporisé DENSE-CLOCK-1.....	78

Figure 3.13. Le module temporisé orienté-objet RTT-1.....	79
Figure 3.14. Le module temporisé orienté-objet DISCRETE-RTT-1.....	80
Figure 3.15. Aperçu du processus de vérification de modèles [Gag07].	81
Figure 4.1. Les Modules du Framework Formel.....	91
Figure 4.2. La gestion de chaîne logistique [Hug02a].....	96
Figure 4.3. Structure du système multi-agents de la gestion de chaîne logistique.....	96
Figure 4.4. Décomposition agents selon AGR de la gestion de chaîne logistique.....	98
Figure 4.5. Diagramme de Classe conceptuel AUML de la gestion de chaîne logistique.....	98
Figure 4.6. Diagrammes de cas d'utilisation de la gestion de chaîne logistique.....	99
Figure 4.7. Diagramme de protocole de la gestion de chaîne logistique pour le cas d'utilisation Place-Command [Hug02a].....	100
Figure 4.8. La Configuration initiale de l'exemple de gestion de chaîne logistique.....	103
Figure 4.9. Résultat d'exécution de la Configuration initiale de l'exemple de gestion de chaîne logistique.....	104
Figure 5.1. Le méta-modèle organisationnel.....	111
Figure 5.2. Cas d'utilisation organisationnel stéréotypés.....	112
Figure 5.3. notation d'organisation d'Agent.....	112
Figure 5.4. Diagramme de classe temporel AUML centré organisation.....	113
Figure 5.5. Aperçu du processus de translation.....	115
Figure 5.6. Modules Maude généré.....	115
Figure 5.7. module orienté objet EXTERNAL-AGENT-O-CLASS.....	115
Figure 5.8. module orienté objet AGENT-O-CLASS.....	116
Figure 5.9. module orienté objet temporisé REAL-TIME-AGENT-O-CLASS.....	116
Figure 5.10 module orienté objet temporisé O-USE-CASEi.....	116
Figure 5.11. module orienté objet temporisé RTMAS-ORGANIZATIONAL-REQUIREMENTS... ..	116
Figure 5.12. Tick Rule.....	117
Figure 6.1. Méta-Modèle RT-Moise+.....	121
Figure 6.2. Ontologie des concepts organisationnels de RT-Moise+.....	122
Figure 6.3. Architecture du Framework proposé.....	123
Figure 6.4. Le module fonctionnel ROLE.....	123
Figure 6.5. Le module orienté objet GROUP.....	124
Figure 6.6. Le module orienté objet Action.....	124
Figure 6.7. Le module orienté objet GOAL.....	124
Figure 6.8. Le module fonctionnel AGENT-STATE-CHART.....	125

Figure 6.9. Le module fonctionnel TEMPORAT-CONSTRAINTS.....	125
Figure 6.10. Le module orienté objet AGENT-MISSIONS.....	125
Figure 6.11. Le module orienté objet MESSAGE.....	126
Figure 6.12. Le module orienté objet MAILBOX.....	126
Figure 6.13. Le module fonctionnel ACQUAINTANCE-LIST.....	127
Figure 6.14. Le module orienté objet MESSAGE-OPERATIONS.....	127
Figure 6.15. Le module orienté objet AGENT-CLASS.....	127
Figure 6.16. Le module orienté Objet REAL-TIME-AGENT.....	128
Figure 6.17. Le module orienté Objet AGENTS.....	128
Figure 6.18. Le module orienté Objet REAL-TIME-AGENTS.....	128
Figure 6.19. Le module fonctionnel AGENT-INTRA-BEHAVIOR.....	128
Figure 6.20. Le module fonctionnel AGENT-INTER-BEHAVIOR.....	129
Figure 6.21. Le module temporisé orienté objet SYSTEM_USE_CASES.....	129
Figure 6.22. Le module temporisé orienté objet RTMOISE-BASED-MAS.....	129
Figure 6.23. Structure du SGC selon RT-Moise+.....	131
Figure 6.24. Exemple de Schéma Social du SGC selon RT-Moise+.....	131
Figure 6.25. Diagramme de Classe du SGC.....	134
Figure 6.26. Diagramme de Cas D'utilisation du SGC	134
Figure 6.27. Diagramme de Communication du SGC	135
Figure 6.28. Diagramme d'état transition de l'agent PCChair.....	135
Figure 6.29. Diagramme d'état transition de l'agent Data-Base.....	136
Figure 6.30. Diagramme d'état transition de l'agent PCMember.....	136
Figure 6.31. Diagramme d'état transition de l'agent Author.....	136
Figure 6.32. Diagramme d'état transition de l'agent Reviewer.....	137
Figure 6.33. Diagramme d'état transition de l'agent Printer.....	137
Figure 6.34. Diagramme de protocole du SGC.....	138
Figure 6.35. Diagramme de temps de l'agent PCChair.....	139
Figure 6.36. Diagramme de temps de l'agent DataBase.....	139
Figure 6.37. Diagramme de temps de l'agent PCMember.....	139
Figure 6.38. Diagramme de temps de l'agent Author.....	139
Figure 6.39. Diagramme de temps de l'agent Reviewer.....	140
Figure 6.40. Diagramme de temps de l'agent Printer.....	140
Figure 6.41. Le module fonctionnel AUTHOR-INTRA-BEHAVIOR.....	141
Figure 6.42. Le module temporisé orienté objet SYSTEM-USE-CASE.....	143
Figure 6.43. Configuration Initiale du Scenario 1.....	143

Figure 6.44. Exécution de la configuration Initiale du Scenario 1.....	144
Figure 6.45. Configuration Initiale du Scenario 2.....	144
Figure 6.46. Exécution de la configuration Initiale du Scenario 2.....	144
Figure 6.47. Une partie du module RTMOISE-BASED-MAS-PREDICATES.....	146
Figure 6.48. Une partie du module RTMOISE-BASED-MAS-CHECK.....	147
Figure 6.49. Quelques appels à la fonction modelCheck.....	147
Figure 6.50. Les résultats des appels à la fonction modelCheck.....	148

Liste des Tableaux

Tableau 2.1. Comparaison des caractéristiques organisationnelles de différentes approches	51
Tableau 2.2. Approches d'identification des modèles d'agent temps réel	52
Tableau 3.1. Notations et opérateurs utilisés dans les modules RT-Maude	84

Introduction Générale

Dans cette introduction nous commençons par décrire le contexte général et les éléments de problématique de notre travail de recherche pour passer aux motivations et objectifs scientifiques qui les sous-tendent. Ensuite, nous présentons la démarche suivie avec l'ensemble de nos contributions. Et nous terminons par une description de l'organisation du reste de cette thèse.

1. Contexte de recherche et éléments de Problématique

La technologie agent est un paradigme prometteur pour l'ingénierie des systèmes complexes et distribués. Dans le but d'assister le développement des Systèmes Multi-Agents (SMA), les méthodologies orientées agents ont été créées ces dernières années afin de supporter la modélisation d'un grand nombre d'applications complexes. Cependant, les agents sont vus comme des entités autonomes qui se comportent selon des objectifs. Ils sont aussi des membres d'une société et doivent échanger des informations avec d'autres agents tout en gardant avec eux des relations à un niveau organisationnel. "Les nouvelles méthodologies orientées agents (qu'on notera par la suite 'MOA') doivent être capable de "capturer et représenter les structures organisationnelles, définir les interactions et les modèles (patterns) de collaboration entre les agents, leurs rôles interne et les dépendances entre les groupes d'agents".

Ce manque d'intérêt global pour la question de l'organisation de sociétés d'agents vient sans doute d'un courant dominant qui définit les SMA comme de simples agrégats d'agents qui interagissent, tout en se comportant d'une manière individualiste, presque égoïste. D'après [Gut99], le concepteur d'un SMA ne s'occupe que des actions individuelles des agents, et suppose que les structures sociales se formeront à partir des interactions. De plus, il est communément admis que le comportement d'un agent est une conséquence directe de son état mental : croyances, désirs, intentions, buts, engagements, etc. Même les travaux sur les agents réactifs [Fer96] conservent souvent cette prééminence de l'agent en considérant que les structures vont émerger des interactions entre individus, même si ces individus ne sont pas supposés avoir d'états mentaux au sens strict.

Cette multitude d'interactions, si elle permet la résolution de problèmes complexes, amène également des situations conflictuelles (conflit d'actions, accès à une ressource, etc) et des situations où des actions redondantes sont entreprises par les agents. Il devient alors nécessaire de coordonner les interactions existant entre les agents, afin de structurer leurs activités [Foi98]. Ce problème est abordé au travers de la notion d'*organisation* qui est un autre aspect fondamental du paradigme multi-agents. Cette notion se préoccupe de définir, de gérer et de modifier les relations entre les individus. Elle correspond donc à un concept liant la description d'une société d'agents et celle des activités de contrôle et de coordination. Cette description se fait en termes de définition, de rôles, d'assignation de tâches, de comportements, de relations d'autorité, etc [Gass92]. Elle permet de contrôler l'ensemble des agents pour qu'ils résolvent ensemble un problème, ceci en restreignant les comportements individuels pour limiter les conflits potentiels et pour gérer le problème de cohérence globale dans le système. [Foi98]

Mais l'inconvénient de cette optique centrée sur l'agent [Gut99] est qu'il est souvent difficile de construire des systèmes complexes de grande taille, pour les raisons suivantes :

- Hétérogénéité des langages. La multiplicité des langages et protocoles d'interaction rend difficile la fédération d'agents non basés sur le même formalisme. L'importance de ce problème a d'ailleurs déclenché divers efforts de normalisation : KQML, FIPA, ...

- Hétérogénéité des modèles d'agents. De très nombreuses architectures et modèles de comportement d'agent ont été proposés. Les travaux portent généralement sur un SMA conçu avec un seul modèle, objet de l'étude. La réunion, au sein d'un même système, d'agents conçus selon des architectures diverses est le plus souvent délicat.

Argente, Julian et Botti [Arg06] ont établi que dans l'analyse et la conception de système multi-agent, les méthodologies de développement ont évolué d'une vue initiale du système qui est centrée sur les aspects individuels des agents à une vue actuelle du système en tant qu'organisation dans laquelle les agents forment des groupes et des hiérarchies et suivent des règles de comportement spécifiques. Par conséquent, ils distinguent deux approches méthodologiques différentes: les méthodologies centrées sur l'agent et celles centrées sur l'organisation. Les méthodologies centrées sur l'agent ne sont pas assez matures; ils réduisent le potentiel de déploiement d'une communauté d'agents à grande échelle développée par des personnes différentes [Obo11]. Nous nous intéressons à cette limitation en utilisant l'abstraction organisationnelle pour modéliser les SMA en général et les systèmes multi-agents temps réel (SMATR) en particulier.

Dans cette thèse, nous nous intéressons aux méthodologies centrées sur l'organisation car il existe déjà de nombreuses revues de celles centrées sur l'agent telles que [Woo01]. De plus, nous analysons ces méthodologies pour savoir si elles prennent en compte la dimension temporelle dans leurs processus de développement.

Nous partageons la même vue des auteurs de [Gut99], qui soutiennent que la prise en compte de concepts organisationnels en tant que concepts primitifs est un point clé de la construction de systèmes complexes, et une solution possible au problème de l'hétérogénéité. En adaptant [Gas92], les mêmes auteurs de [Gut99] définissent l'organisation comme un framework d'activités et d'interaction, mis en place par la définition des groupes, rôles et de leurs relations.

En évitant de centrer leurs propos sur l'agent, les auteurs de [Gut99] considèrent l'organisation comme des relations structurelles dans un ensemble d'agents. Elle peut être décrite uniquement par sa structure, c'est à dire par la manière dont sont définis et liés les groupes et rôles, sans se concerner du comportement effectif des agents.

De plus, en s'appuyant aussi sur [Aza07], cette organisation permet de structurer les différents composants du système ainsi que leur mode de fonctionnement global. L'organisation s'intéresse aussi aux dimensions sociales des agents [Hab68]. Elle peut être considérée comme une structure décrivant les interactions et autres relations qui existent entre les membres de la dite organisation [Fox81] (dans le but d'accomplir un objectif commun). L'organisation peut donc apparaître comme une structure de coordination et de communication [Mal87].

D'après [CSB05], il existe deux principales visions du concept organisation. Dans la première vision, l'organisation est une entité à part entière qui est composée d'un groupe d'agents. La structure organisationnelle ainsi que le comportement global sont pré-établis. Plusieurs contraintes peuvent être exprimées au niveau de la topologie, des protocoles d'interaction, des normes utilisées ainsi que le déroulement des activités. Dans la deuxième vision, l'organisation émerge du comportement global des différents composants du système. Les différents éléments faisant partie de l'organisation sont préalablement connus et leur comportement individuel est spécifié. Cependant, le comportement global est inconnu. Ces cas sont surtout rencontrés dans les applications d'ingénierie sociale qui consistent à étudier la dynamique des structures sociales. L'objectif de ces applications est de découvrir le modèle organisationnel généré suite aux activités et décisions des différents acteurs sociaux ainsi que leur interaction, Comme c'est le cas des sociétés de

fournis [Rou98]. Dans cette vision, il est intéressant de spécifier de manière explicite au niveau de l'organisation, les règles de gestion de conflits ou d'interférences générés suite aux activités de chacun des agents [Aza07].

Avec l'évolution du paradigme agent et la complexité des systèmes distribués, la vision organisationnelle des SMA est largement acceptée. Dans cette perspective, les agents seront représentés à travers une société, dans la quelle chacun joue un rôle particulier et interagit avec d'autres agents. Ceci revient à définir des structures d'interaction sociale comme les groupes, équipes (teams) ou coalitions qui modélisent le comportement et les interactions observés dans le monde réel. Plusieurs de ces aspects ne sont pas considérés dans les méthodologies classiques (orientées agent basées objet). En conséquence, durant ces dernières années plusieurs efforts ont été faits pour introduire les aspects organisationnels au sein des MOA existantes ou pour créer de nouvelles méthodologies qui prennent en compte la dimension sociale des SMA. Malheureusement, à ce jour il n'y a pas de méthodologie qui soit largement adoptée et considérée comme un standard.

Dans ce contexte, si la recherche dans ce domaine est en pleine évolution depuis plus d'une décennie, la plupart des travaux de recherche publiés dans la littérature ne sont pas encore tout à fait exploités dans la pratique, en particulier dans l'industrie. La plupart des applications industrielles complexes sont de nature temps réel où le temps est un facteur primordial. Une tendance récente dans le développement des systèmes temps réel est l'utilisation des systèmes multi-agents temps réel (SMATR). À l'opposé des SMA classiques, les SMATR sont bien appropriés pour tenir compte des caractéristiques intrinsèques des systèmes distribués temps réel.

Cependant, alors que plusieurs modèles organisationnels ont été proposés dans la littérature pour assurer l'aspect social dans les applications basées système multi-agent. Ces modèles ont considérablement enrichi le domaine par différents paradigmes pour améliorer le processus de développement de SMA, mais de l'autre côté, ils ont totalement omis les contraintes temps réel, qui sont un aspect important pour la construction des SMATR.

Dans l'objectif de faciliter le développement des SMATR, quelques méthodologies ont été proposées dans la littérature. Ces méthodologies, bien qu'elles soient à l'origine de l'apparition de travaux intéressants dans ce domaine, n'ont pas encore atteint un niveau de maturité élevé. En effet, l'organisation en tant que dimension très importante dans les SMA est peu abordée dans ces méthodologies. La pénétration des SMATR dans l'industrie nécessite des modèles organisationnels permettant de spécifier la structure et les fonctionnalités de la société d'agents tout en respectant les contraintes de temps relatives à l'application à développer.

2. Motivations & Objectifs scientifiques

L'adoption d'une approche centrée sur l'organisation pour développer un système multi-agent (SMA) repose sur la définition d'un ensemble de contraintes qu'un groupe d'agents respectent pour atteindre facilement leurs objectifs sociaux [Lem98]. Cet ensemble de contraintes est généralement décrit comme des modèles organisationnels. En ayant un modèle d'organisation sous-jacent, le système multi-agent peut assurer un certain niveau de rendement et d'efficacité [Gas01], puisque le modèle établit des moyens pour coordonner et contrôler mieux le comportement des agents [Mur14a].

Une tendance récente dans le développement de systèmes distribués temps réel est l'utilisation de SMATR. Contrairement aux SMA conventionnels, les SMATR reflètent les caractéristiques intrinsèques des systèmes temps réel, plus précisément les contraintes de temps. Pendant de

nombreuses années, les concepteurs de SMA ont des méthodologies de développement et des langages de modélisation qui ne reflètent pas les différentes contraintes temporelles que ces systèmes peuvent avoir. En effet, il n'est pas facile de conceptualiser les concepts de SMATR en utilisant des approches classiques d'ingénierie logicielle d'agent. Par conséquent, un problème de recherche critique pour la communauté des agents temps réel a été la définition d'un modèle organisationnel approprié pour l'analyse et la conception de leurs principales propriétés.

L'organisation en tant que dimension très importante dans les SMA est peu abordée dans les méthodologies de développement des SMATR qui n'ont pas de Modèle Organisationnel qui tient en compte la formalisation des besoins organisationnels du futur système. L'idée est que les méthodologies SMA orientées Organisation actuelle nécessitent de nouvelles techniques et des propriétés spécifiques pour qu'on puisse les utiliser au développement des SMATR. Notre objectif de ce travail est l'analyse des caractéristiques et besoins essentiels, que les méthodologies SMA doivent prendre en considération afin de mieux répondre aux exigences des systèmes temps réel. Particulièrement, on s'intéresse à la perspective organisationnelle pour améliorer et enrichir la conception des SMATR. Nous voulons nous situer à un niveau d'abstraction supérieur à celui exprimant les systèmes et leurs entités, en se positionnant au niveau des structures sociales. Donc, notre intérêt principal est le développement d'un modèle organisationnel pour les systèmes multi-agents temps réel. Ce modèle permettra de réduire la complexité du problème à traiter en explicitant la répartition des tâches entre les agents et en fixant les relations qui les relient.

De plus, même les méthodologies proposées pour le développement de SMATR comme «RT-Message» [Jul04], «BDI-ASDP étendu pour le temps réel» [Mel05] et «la Méthode de développement de Lichen Zhang» [Zha06], sont inappropriées. Elles ont certainement apportées des réponses importantes dans le processus de développement de SMATR. Cependant, l'aspect méthodologique n'est pas encore maîtrisé. En effet, aucune de ces méthodologies ne dispose d'un modèle organisationnel prenant en compte la formalisation des exigences organisationnelles du futur système.

La modélisation et la formalisation des aspects organisationnels de SMATR sont, à notre avis, un moyen important pour les activités d'analyse et de conception. En outre, la conception de SMA nécessite l'implication de techniques de génie logiciel.

Notre travail couvre essentiellement le problème de la conception de SMATR, en l'associant avec un modèle organisationnel ayant des restrictions temps réel dans un environnement distribué. Notre effort est concentré sur l'aspect fonctionnel des agents au sein de l'organisation sans entrer dans les détails des architectures d'agents et des modèles de comportements. Cela n'implique pas que de telles structures doivent être statiques, ni que les groupes d'agent doivent être complètement indépendants comme l'indique les auteurs de [Gut99]. Au contraire, la puissance des structures d'organisations vient de la capacité des agents à se joindre à de tels groupes et à y acquérir de nouvelles aptitudes qu'ils n'auraient pas obtenues sinon et qu'ils utiliseront dans d'autres groupes.

Le développement de système avec des fonctionnalités venant des deux domaines de recherche (SMA & paradigme organisationnel) produit une augmentation de la complexité et la nécessité d'adapter les techniques actuelles. Pour résoudre ce problème, nous proposons d'étendre les diagrammes Temporisés Agent UML (AUML) [Lao13] avec des concepts organisationnels. Donc l'idée principale est de développer une méthode qui soit basée sur un modèle organisationnel intégrant les contraintes temporelles combiné à des diagrammes AUML étendue.

Il n'y a actuellement aucun travail qui applique les représentations AUMML pour des spécifications organisationnelles de SMATR dédiées aux applications temps réel.

Dans le volet formel, avoir une sémantique formelle fournit des spécifications précises du comportement attendu du système multi-agent basé sur un modèle organisationnel, permettant ainsi aux utilisateurs de les comprendre et aussi aux modélisateurs de raisonner à leur sujet et de prouver leur exactitude. Un effort a été fait pour fournir un framework de spécification formel appliqué au système multi-agents centré sur l'organisation en utilisant un langage de spécification basé sur la logique de réécriture appelé Maude pour traduire les modèles organisationnels, ce qui permet d'aborder ces problèmes. Ce langage formel possède une extension pour la description des systèmes temps réel appelé Real Time Maude (RT-Maude) et qui sera largement utilisée dans notre travail. Une telle formalisation offre des avantages supplémentaires, tels que permettre la simulation des spécifications ou donner accès aux outils Maude de raisonner dessous.

Maude prend en charge la définition de syntaxe par l'utilisateur et, par conséquent, il offre des prototypes d'analyseurs gratuitement. Un autre avantage de l'utilisation de Maude est qu'il fournit un support d'outils pour des possibilités intéressantes telles que la simulation de modèle, l'analyse d'accessibilité et la vérification de modèle [Cla07b]. De plus, du fait que les spécifications de la logique de réécriture sont exécutables, le framework formel proposé peut être utilisé comme un prototype, ce qui permet de le simuler afin de pouvoir fournir un premier retour sur l'adéquation de la solution. S'ajoute à cela que Maude [Tro11] offre aussi différentes possibilités pour réaliser la simulation, y compris l'exécution pas-à-pas, plusieurs stratégies d'exécution, etc.

3. Démarche suivie & Contributions

Au début on a effectué une étude bibliographique profonde sur les systèmes multi-agents temps réel, tout en mettant l'accent sur les mécanismes essentiels de l'organisation au sein des SMA et SMATR d'une part et, sur la dualité entre la structure organisationnelle et le processus d'organisation pour définir une organisation d'autre part.

Ensuite on a établi une étude comparative entre les différents modèles organisationnels existants dans les SMA et SMATR, en définissant un ensemble de critères et de paramètres sous différentes dimensions d'analyse. On a essayé aussi d'explorer les différentes façons permettant d'exprimer les contraintes qu'on veut imposer aux agents avec une organisation. Cette étude comparative a mis l'accent, sur les composants structurels, les méta-modèles, et les structures organisationnelles supportées par chacune des méthodologies organisationnelles analysées.

Dans cette analyse comparative on revient également sur la préoccupation principale relative à notre sujet, à savoir, la proposition d'un modèle organisationnel qui prend en charge les différentes dimensions d'analyse vues (fonctionnelle, structurelle et dynamique) et principalement la dimension temporelle du SMATR, vu que la plupart des méthodologies étudiées l'ont négligée. Ce modèle permettra de réduire la complexité du problème à traiter en explicitant la répartition des tâches entre les agents ainsi que leurs contraintes temporelles et en fixant les relations qui les relient d'un point de vue organisationnel. En conclusion, une telle étude adresse quelques pistes intéressantes à court et à moyen terme pour définir de nouvelle technique et caractéristiques facilitant la conception des SMATR.

Après on a proposé un méta-modèle organisationnel (éventuellement par affinement et/ou extension de modèles existants) pour l'analyse et la conception des SMATR à travers la description de composant (organisations, rôles, agents); la définition de fonctionnalité (but, tâches); et les

interactions, étendant ainsi les autres méta-modèles organisationnels-agents afin de prendre en considération les concepts temporels liés à l'organisation. Tout en mettant l'accent, sur les extensions apportées aux différents diagrammes du formalisme de modélisation Agent UML (diagrammes de cas d'utilisation ; de class ;...) afin de combiner les avantages que peut offrir l'utilisation de ce formalisme pour la représentation des concepts organisationnels identifiés lors de l'étude comparative effectuée précédemment dès la phase d'analyse et la spécification des contraintes temporelles liées aux SMATR. *Ce travail a fait l'objet d'une communication internationale [Lao13] publié dans les proceedings de la conférence "Science and Information (SAI) 2013" à London UK, et indexée sous IEEE Xplore.*

Et pour faire face au manque de sémantique formelle des diagrammes AUML on a proposé la structure de base d'un Framework de spécification formelle permettant de traduire les différentes dimensions d'analyse vues (fonctionnelle, structurelle et dynamique) et principalement la dimension temporelle du SMATR avec le langage Real Time Maude pour fournir des descriptions précises facilitant leurs processus de vérification et de validation sur des études de cas concrètes. *Ce travail a fait l'objet d'un Book Chapter [Lao14] publié dans Springer Book Series-Studies in Computational Intelligence (Volume 542, 2014, pp 171-180) Sous springer Verlag.*

Par la suite les résultats des travaux [Lao13, Lao14] ont été exploités avec des extensions faites d'une part pour formaliser le modèle organisationnel Agent-Groupe-Rôle (AGR). Cette approche a été illustrée en utilisant l'étude de cas Gestion de Chaîne Logistique (en anglais Supply Chain-Management). Et d'autre part étendre le modèle Model of Organization for multi-agent SystEms (Moise+) pour qu'il intègre la dimension temporelle. Ce qui nous a amené à mettre à jour le Framework de spécification formelle des SMATR proposé dans [Lao14] pour tenir compte des extensions proposées dans le méta-modèle [Lao13] concernant les spécificités organisationnelles des SMATR. Pour ensuite expérimenter les résultats de l'application du nouveau Framework proposé sur l'étude de cas Système de Gestion de Conférence (SGC en anglais Conference Management System) à travers le processus de traduction et on termine par la vérification formelle de quelques propriétés organisationnelles en utilisant la technique du modèle-checking du langage Real-Time Maude. *Cette dernière contribution a fait l'objet d'un article [Lao17] publié dans une revue internationale de renommée appelée "Multiagent and Grid Systems" (MGS), vol. 13, no. 4, pp. 395-419, 2017, sous le publisher: IOS PRESS.*

4. Organisation du manuscrit

Après ce bref exposé de nos propositions, cette section introduit le plan de la thèse. Ce dernier est organisé en deux parties dont la première comprend trois chapitres consacrés au contexte dans lequel s'inscrivent nos travaux. Le chapitre 1 propose une revue de la littérature sur les systèmes multi-agents temps réel (SMATR). Le chapitre 2 quant à lui décrit la vue organisationnelle dans les SMA tout en mettant l'accent sur leurs méthodologies de développement centrées organisation à travers une étude comparative faite sur quelques modèles organisationnels. Alors que le chapitre 3 présente les langages de spécification et de conception utilisés dans les SMA de façon générale et le formalisme de représentation graphique Agent UML [Hah08] utilisé pour la modélisation, ainsi que le langage de spécification formelle Real Time Maude (RT-Maude) sur lequel se base la partie formelle de notre approche en particulier, et cela afin de mieux cerner les spécificités de notre approche.

La deuxième partie, composée des trois chapitres descriptifs des apports des travaux et de leur validation est consacrée aux aspects théoriques et réalisations pratiques de notre modèle. Le chapitre 4 décrit d'une part le processus de formalisation qui sera appliqué au modèle organisationnel Agent-Groupe-Rôle (AGR) en se basant sur la logique de réécriture présentée dans le chapitre précédent et d'autre part validé le framework obtenu à travers l'exemple Gestion de chaîne logistique. Tandis que, le chapitre 5 présente les principaux concepts organisationnels et temporels de notre méta-modèle ainsi que les extensions réalisées dans les diagrammes AUML, avant de décrire la structure de base du framework de spécification formelle proposé. Le chapitre 6 quant à lui décrit l'extension temps réel ajoutée au modèle organisationnel Moise+ qui sera noté "RT-Moise+" suivie de sa formalisation qui est basée sur l'approche présentée dans le chapitre précédent. Dans ce même chapitre 6, nous y abordons le problème de la conformité du modèle d'exécution par rapport au modèle de contraintes et nous montrons comment il est possible de vérifier les contraintes temps réel en utilisant le modèle checker offert par le langage RT-Maude à partir du framework du modèle proposé RT-Moise+. Nous terminons cette deuxième partie par un bilan sur les travaux que nous avons effectués et ceux qui, inévitablement, restent à faire et qui vont être des perspectives.

Première partie

Dans cette première partie, nous nous intéressons à une problématique particulière qui est la prise en compte des aspects temporels dans les systèmes multi-agents tout en mettant l'accent sur leurs aspects organisationnels et les méthodes de spécification formelle.

Le chapitre 1 est consacré aux systèmes multi-agents temps réel. Nous y définissons les notions temporelles évoquées ou utilisées dans le cadre de cette thèse, ainsi que les notions d'agents, de systèmes multi-agents, et nous présentons leurs caractéristiques afin de montrer l'importance des interactions et surtout de l'organisation dans ces systèmes.

Le chapitre 2 quant à lui décrit les principes et concepts de base permettant de mieux comprendre la vue d'organisation au sein des systèmes multi-agents. Il présente aussi quelques aspects des modèles organisationnels utilisés dans les SMA afin de montrer que l'évolution des méthodes de spécification des SMA, notamment celles basées sur des approches organisationnelles où les concepts sociaux (rôles, groupes, interactions, ...) sont pris en compte, offrent de nouvelles possibilités. Suivie d'une comparaison entre ces modèles choisies dans le but d'établir notre méta-modèle organisationnel de SMATR pour faciliter la prise en compte du temps dans les différentes dimensions d'un SMA.

Le chapitre 3 présente les fondements sur lesquels s'appuient les autres chapitres. Nous décrivons pour commencer, les langages graphique de modélisation et de conception utilisés dans les SMA, en particulier le langage UML et son extension AUML permettant d'expliciter des contraintes liées au temps facilitant la communication entre les agents. Ensuite, on va parler des langages formels et leurs fondements logiques on prenant le cas du langage de spécification formelle Maude et son extension pour le temps réel Real Time Maude.

Deuxième partie

Nous commençons cette partie par la formalisation du modèle AGR en appliquant les concepts décrits précédemment et on valide le résultat obtenu à travers l'exemple de gestion de chaîne logistique (en anglais Supply Chain Management) (**chapitre 4**). Après cela, nous présentons le méta-modèle proposé et les extensions apportées aux diagrammes AUML, suivie de leur processus

de translation en spécification formelle RT-Maude (**Chapitre 5**). Par la suite, nous décrivons les aspects temporels ajoutés dans le modèle organisationnel Moise+ qu'on notera RT-Moise+ et on validera dans un premier temps le framework formel obtenu par l'instanciation du SMATR développé selon RT-Moise+ dans l'exemple de Système de Gestion de Conférence (SGC en anglais Conference Management System) illustrant ainsi un fonctionnement temporel centré organisation. Pour montrer dans un deuxième temps comment il est possible de vérifier les contraintes temps réel par une analyse de type modèle checking en utilisant le modèle checker offert par RT-Maude sur le framework formel de RT-Moise+ (**chapitre 6**). A la fin de cette partie on va établir un bilan sur les travaux que nous avons effectués et ceux qui, inévitablement, restent à faire pour enrichir ce travail et qu'on considère comme des perspectives.

Le chapitre 4 présente une de nos contribution qui est une formalisation basé sur la logique de réécriture pour le modèle organisationnel Agent-Groupe-Rôle (AGR). Nous décrivons dans un premier temps le modèle AGR en utilisant les diagrammes étendu AUML puis nous détaillons sa spécification formelle à l'aide du langage Maude qui sera validé à travers l'étude de cas de Gestion de chaîne logistique (Supply Chain Management).

Le chapitre 5 constitue une autre contribution. Il propose la conception du méta-modèle organisationnel (éventuellement par affinement et/ou extension de modèles existants) adapté aux SMATR, ensuite il présente de manière approfondie les extensions faites aux diagrammes AUML pour terminer par la structure de base du framework de spécification formelle à utiliser.

Le Chapitre 6 dans lequel nous proposons, la prise en compte de la dimension temporelle au sein du modèle organisationnel Moise+ qu'on notera RT-Moise+ n'est qu'une deuxième application de notre approche pour la prise en compte des différents aspects temporels et concepts liés aux SMATR pour aboutir à la formalisation du modèle RT-Moise+ en utilisant le langage RT-Maude. La conception de cette contribution s'appuie sur les résultats du chapitre précédent. Et pour valider ce travail on va faire l'instanciation du SMATR basé sur RT-Moise+ dans l'exemple de système de gestion de conférence illustrant ainsi un fonctionnement temporel centré organisation. Donc, en plus d'apporter une seconde validation de l'efficacité de l'approche, nous cherchons à illustrer son aspect de flexibilité, c'est-à-dire, la facilité d'adaptation sur différents modèles organisationnels. Dans ce chapitre nous y abordons aussi le problème de la conformité du modèle d'exécution par rapport au modèle de contraintes et nous montrons comment il est possible de vérifier les contraintes temps réel par une analyse de type modèle checking à partir du framework formel du modèle RT-Moise+ proposé en utilisant le modèle checker offert par RT-Maude.

A la fin de cette partie un bilan des travaux de recherche est donné, suivie des perspectives pouvant être développées par la suite, ouvrant ainsi de nouvelles voies et propositions dans le but d'améliorer ce travail.

Chapitre 1

Les Systèmes Multi-Agents Temps Réel

Introduction	10
1. Les Systèmes Multi-Agents	11
1.1. Le Concept d'Agents	11
1.2. Agent versus objet	12
1.3. Caractéristiques des agents	13
1.4. Classification des agents	13
1.5. Approche Voyelles	14
1.6. Points Forts & Faibles du Paradigme Orienté Agent	18
1.7. Domaines d'application des systèmes multi-agents	21
2. Les Systèmes Multi-Agent Temps Réel	22
2.1. Définition	23
2.2. Les Systèmes Temps Réel	23
2.3. Mise en Œuvre du Temps dans les SMA	26
3. Conclusion	28

Introduction

La technologie agent a suscité beaucoup d'attention ces dernières années, en conséquence, l'industrie commence à employer cette technologie. Mais, au début très peu de travaux se sont intéressés aux techniques de développement utilisant la technologie d'agent. De ce fait, un nouveau thème de recherche, celui de l'ingénierie logicielle orientée agent (en anglais Agent Oriented Software Engineering "AOSE"), s'est développé. L'objectif principal de l'ingénierie logicielle orientée agent, est de proposer des méthodologies et des outils qui facilitent le développement et la maintenance des applications multi-agents [Tve01]. Dans ce contexte, il existe trois principales activités de recherche dans le domaine de l'ingénierie logicielle [Cer05a]: (i) le développement d'outils conceptuels (outils de notation ou de modélisation formelle), (ii) le développement des outils d'assistance au cours du processus du développement (les CASE tools) et (iii) les méthodologies multi-agents.

L'application des systèmes multi-agents aux environnements temps réel est une voie très intéressante de travail qui peut fournir de nouvelles solutions aux systèmes complexes et restrictifs comme les systèmes temps réel. L'utilisation du paradigme agent dans des scénarios temps réel provient des propriétés des nouveaux systèmes temps réel (tel que l'intelligence et la distribution). Ce paradigme essaye d'incorporer la flexibilité, l'adaptabilité et la distribution dans un framework temps réel. Une méthode appropriée pour le développement des Systèmes Multi-Agent Temps Réel (SMATR) doit prendre en compte les caractéristiques intrinsèques de ce type de système. Souvent, il est essentiel d'analyser les besoins dans le comportement global du SMA en relation avec les propriétés comportementales des agents, afin de développer un système avec des propriétés correctes.

Les Systèmes d'Intelligence Artificiel Temps Réel (SIATR) ont émergés comme des techniques utiles pour résoudre des problèmes complexes nécessitant de l'intelligence et une réaction temps réel [Mel05]. Le développement des systèmes temps réel quant à lui a suivi le mode de distribution, principalement parce qu'un grand nombre d'applications temps réel sont fondamentalement distribuées et dynamiques.

Les SMATR sont actuellement très largement utilisés, particulièrement pour les applications complexes nécessitant l'interaction entre plusieurs entités ayant des restrictions temporelles à respecter. Plusieurs méthodologies ont été proposées pour le développement des SMATR. Cependant, ces méthodologies qui constituent soit une extension des méthodologies orientées objet, ou une extension des méthodologies orientées-agent, demeurent insuffisantes pour la formalisation des concepts organisationnels liés aux SMATR. Par ailleurs, peu d'efforts ont été déployés en matière de standardisation des méthodes de conception des SMA. Il apparaît donc évident que le développement des SMATR reste encore un domaine ouvert. Le succès du paradigme orienté-agent exige des méthodologies systématiques pour la spécification et la conception des SMATR.

Notre travail de recherche se place naturellement dans le contexte du génie logiciel et de l'intelligence artificielle distribuée où nous nous occupons de plus près de l'aspect génie logiciel dans le développement des SMATR; dès lors, il est devenu impératif de s'investir dans les méthodologies orientées-agents.

Dans ce chapitre, nous commençons par introduire les concepts de base qui découlent du domaine de recherche que constituent les systèmes multi-agents (**section 1**), ainsi que leurs avantages et inconvénient. Ensuite, nous allons présenter les systèmes multi-agents temps réel en évoquant principalement les systèmes temps réel et leur implication dans les SMA (**section 2**). On termine par une conclusion (**section 3**).

1. Les Systèmes Multi-Agents

De nos jours, les systèmes multi-agents ont pris une place de plus en plus importante en informatique, que ce soit dans le domaine de l'intelligence artificielle, dans celui des systèmes informatiques distribués, que de la robotique et du génie logiciel. C'est une discipline qui s'intéresse aux comportements collectifs produits par les interactions de plusieurs entités autonomes et flexibles appelés agents.

Les systèmes multi-agents mettent en œuvre des agents homogènes et hétérogènes ayant des buts communs ou distincts. Un SMA est un système distribué composé d'un ensemble d'agents qui interagissent le plus souvent, selon des modes de coopération, de concurrence ou de coexistence. Selon [Cha99] un SMA est généralement caractérisé par :

- Chaque agent a des informations ou des capacités de résolution de problèmes limitées, ainsi chaque agent a un point de vue partiel;
- Il n'y a aucun contrôle global du système multi-agents;
- Les données sont décentralisées;
- Le calcul est asynchrone.

De plus, Tranier [Tra07] décrit un SMA comme étant un ensemble d'entités actives et autonomes (les agents) qui évoluent dans un environnement. Cette relation générale entre un agent et son environnement est illustrée sur la Figure 1.1.

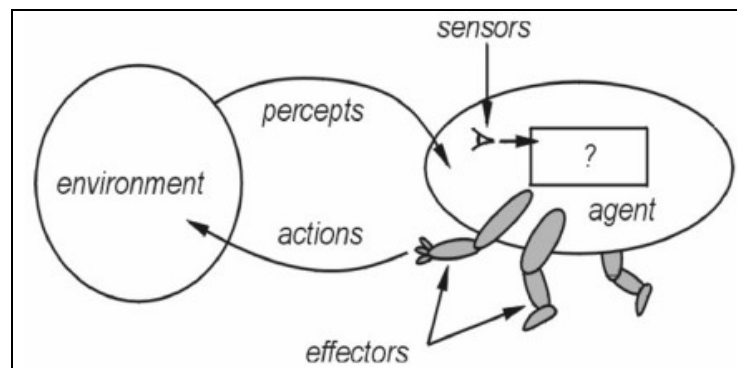


Figure 1.1. Interaction entre un agent et son environnement [Tra07].

Dans cette section nous introduisons d'abord, les notions d'agents et de systèmes multi-agents (SMA), pour détailler par la suite les SMATR.

1.1. Le Concept d'Agents

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

- Des entités clairement identifiables de résolution de problèmes avec des bornes et des interfaces bien définies;
- Situés dans un environnement particulier; ils reçoivent des entrées liées aux états de cet environnement par des capteurs et agissent sur cet environnement par des émetteurs;
- Destinés à atteindre un objectif spécifique;
- Autonomes et responsables de leur comportement;

- Capables d'adopter un comportement flexible pour résoudre des problèmes selon les objectifs de la conception; ils sont réactifs (capables de s'adapter aux changements d'état de leur environnement) et proactifs (capables d'adopter un nouvel objectif);
- Capables dans un univers multi-agents, de communiquer, coopérer, se coordonner, négocier les uns avec les autres.

Lorsqu'un agent perçoit une situation dans l'environnement, il essaie de la reconnaître. Si la situation lui est familière, il peut enclencher un processus de planification afin de résoudre le problème. Il peut aussi reconnaître la situation en terme d'action et donc, passe à l'exécution de la tâche (Reconnaissance- Exécution). Lorsque l'agent perçoit des situations qu'il connaît très bien, il peut faire intervenir son comportement réactif en passant directement à l'action (Perception-Exécution). S'il ne peut pas résoudre un problème (situation non-familière), il engage un processus de coopération pour demander de l'aide aux autres agents (Reconnaissance- Prise de décisions).

1.2. Agent versus objet

Plusieurs chercheurs définissent la notion d'agent par rapport à celle d'objet, leur argumentation se base sur le fait que la maîtrise d'une nouvelle technologie ou d'un nouveau concept est facilitée par sa référence à des technologies ou concepts voisins et d'ores et déjà maîtrisés [Ode00]. Les objets se définissent comme des entités informatiques qui encapsulent un état et qui sont capables d'accomplir des actions (ou méthodes dans cet état) tout en communiquant par échange de messages. Bien qu'il existe plusieurs similitudes entre les agents et les objets, il n'en ait pas moins qu'il existe d'importantes différences [Woo02]:

La première concerne le degré d'autonomie : les agents comme les objets peuvent encapsuler leur état interne, et agir sur cet état par le biais de leurs méthodes. Pour les objets, une invocation de méthode est assimilée parfois à un envoi de message mais une telle méthode doit être explicitement invoquée par un autre objet pour agir tandis qu'un agent recevant une information décidera par lui-même de l'action subséquente [Woo02, Jar02].

Quant à la deuxième différence entre les objets et les agents provient de la flexibilité du comportement d'un agent. Bien qu'il soit possible de construire des objets disposant de ces caractéristiques, les auteurs de [Woo02, Jar02] noteront que le modèle standard d'un objet ne dit rien à propos de ces types de comportements.

Alors que, la troisième différence entre ces deux concepts vient du fait que les agents sont censés avoir chacun son fil d'exécution (thread) [Woo02, Jar02]. Cependant, dans le modèle objet standard, il existe un fil d'exécution unique pour tout le système, bien que l'on voie apparaître du parallélisme dans la programmation orientée objet (POO) avec des langages comme Java offrant la possibilité de lancer plusieurs fils d'exécution en même temps. Toute fois, il faut noter une évolution dans la communauté des adeptes de la POO vers la notion d'objet actif possédant son propre fil d'exécution. Les objets actifs sont généralement autonomes, c'est-à-dire qu'ils peuvent exhiber un comportement sans avoir été sollicités par un autre objet [Hik03]. Odell et ses collègues [Ode00] présentent les agents comme une extension d'objets actifs possédant une prise d'initiative (la capacité à initier des actions sans intervention externe) et un libre arbitre (la capacité à refuser ou à modifier une requête externe). Ils définissent un agent comme un objet qui peut prendre des initiatives et refuser de répondre à une sollicitation.

1.3. Caractéristiques des Agents

Selon l’auteur de [Maz13], il existe de nombreuses caractéristiques relatives aux agents qui procurent la force au paradigme agent et qui les distinguent des autres paradigmes logiciels tels que les systèmes orientés objets, les systèmes distribués et les systèmes experts. Il cite les plus intéressantes afin d’avoir une meilleure vision d’un agent, qui cependant, ne possède pas forcément toutes ces caractéristiques :

- L’autonomie : l’agent est capable d’agir sans l’intervention d’un tiers (humain ou agent) et contrôle ses propres actions ainsi que son état interne ;
- La capacité représentationnelle ou la perception: l’agent peut avoir une vision très locale de son environnement, mais aussi une représentation plus large de cet environnement et notamment des agents qui l’entourent ;
- La réactivité : l’agent reçoit des informations de son environnement, et il doit être capable de réagir en conséquence ;
- L’intentionnalité ou pro-activité : un agent ne réagit pas simplement aux changements de l’environnement mais il fait preuve d’un comportement dirigé vers ses buts et prend des initiatives lorsqu’elles lui semblent appropriées ;
- La communication avec les autres agents ;
- L’anticipation : l’agent possède plus ou moins les capacités d’anticiper les événements futurs;
- La rationalité : les agents rationnels disposent de critères d’évaluation de leurs actions et de choix des meilleures pour atteindre le but ;
- L’adaptabilité ou la flexibilité: un agent adaptable est un agent capable de moduler ses aptitudes selon l’état de l’environnement. Il est capable d’apprentissage et sait résoudre un nouveau problème à partir de son expérience ;
- La sociabilité : un agent interagit avec les autres agents (logiciels et humains) quand la situation l’exige afin d’accomplir ses tâches et aide les autres à atteindre leurs buts.

1.4. Classification des Agents

Des chercheurs issus de l’intelligence artificielle distribuée comme Moulin et Chaib-Draa [Mou96] caractérisent les agents par leur capacité à résoudre les problèmes [Maz13], comme suit :

- Un agent réactif réagit aux changements de son environnement ou aux messages provenant des autres agents. Il ne possède pas de représentations de lui-même, de son environnement ou des autres agents ;
- Un agent intentionnel est une entité qui perçoit et agit sur son environnement, qui est capable de raisonner à propos de ses intentions et ses croyances, qui possède des buts explicites et qui peut concevoir et exécuter des plans d’actions ;
- Un agent social possède en plus des propriétés d’un agent intentionnel, des modèles explicites des autres agents.

Cependant, Ferber [Fer95] classe les agents en fonction de leur caractère cognitif ou réactif. Les agents cognitifs, dont la plupart sont intentionnels, disposent chacun d’une base de connaissances comprenant l’ensemble d’informations et de savoir-faire leur permettant de réaliser leur tâche et de gérer les interactions avec les autres agents et avec leur environnement. Ils sont capables

d'anticipation et peuvent planifier leur comportement. Par contre, les agents réactifs sont incapables d'anticiper ou de planifier leur comportement.

Alors que, Nwana [Nwa96] propose sept catégories d'agents :

- Les agents réactifs ;
- Les agents collaboratifs: sont des agents autonomes et qui coopèrent et négocient avec les autres agents afin d'atteindre des ententes lors de la résolution distribuée de problèmes;
- Les agents d'interface: ces agents fournissent une assistance à l'utilisateur. Un agent d'interface est capable de s'adapter aux habitudes de l'utilisateur et à ses préférences ;
- Les agents mobiles: ce sont des agents qui peuvent se déplacer d'un site à un autre en cours d'exécution pour se rapprocher des données ou des ressources;
- Les agents d'information ou d'Internet : ces agents ont pour rôle d'administrer, manipuler ou collecter les informations à partir de plusieurs sources d'informations distribuées ;
- Les agents hybrides: un agent hybride consiste en la combinaison de plusieurs caractéristiques des autres agents au sein d'un même agent tels que la mobilité, la collaboration, l'autonomie, la capacité à apprendre, etc.
- Et les agents intelligents : Nwana pense que les agents logiciels intelligents n'existent réellement pas encore mais d'après Jennings et Wooldridge [Jen98a], un agent intelligent se caractérise par son autonomie, sa réactivité, sa capacité à agir ainsi que sa sociabilité.

1.5. Approche Voyelles

Voyelles [Dem95] est un cadre conceptuel et méthodologique pour le développement des systèmes multi-agent caractérisé principalement par sa puissance d'abstraction. Dans cette approche (Figure 1.2), Demazeau [Dem01] distingue quatre dimensions orthogonales qui interviennent dans l'analyse, la conception, l'implémentation et l'exécution d'un système multi-agent : l'Agent, l'Environnement, l'Interaction et l'Organisation (AEIO) que nous présentons brièvement ci-dessous.

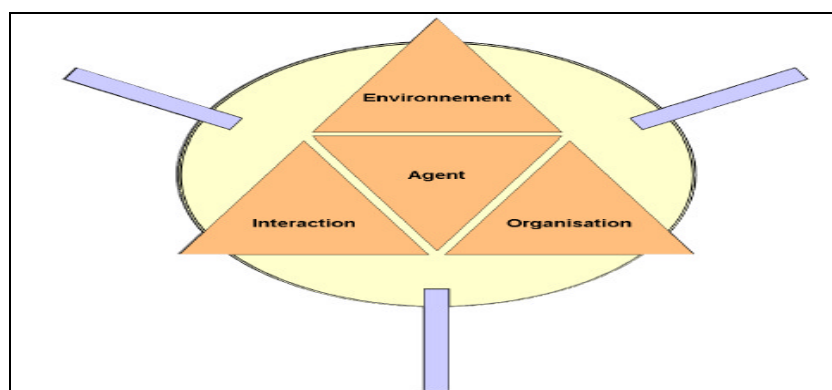


Figure 1.2. L'approche Voyelles [Tra07].

L'idée sous-tendue par cette approche comme le fait savoir Michel dans [Mic04] est que chacune de ces dimensions définit une problématique particulière qui doit être explicitement étudiée dans chaque étape de la conception, de la phase d'analyse à l'implémentation. La plate-forme de développement Volcano est basée sur cette méthodologie [Ric02]. Elle a par exemple été utilisée dans [Das02] où les auteurs appliquent la décomposition Voyelles pour l'étude de la coordination dans le contexte de la RoboCup.

L'intérêt principal de l'approche voyelles est de proposer une séparation abstraite mais avec suffisamment de généralité pour être utile dans la spécification ; conception ; et développement des SMA.

1.5.1. Agent

Le concept d'agent a été étudié en détaille dans les sous sections précédentes. Nous nous contentons de donner la définition voyelle d'agent cité dans [Her04] : "les agents sont des entités concrètes ou virtuelles qui existent dans un environnement dans lequel elles peuvent agir et communiquer entre eux, et qui présentent aussi un certain comportement autonome."

1.5.2. Environnement

Selon [Pes97], l'environnement peut être considéré comme la représentation du monde dans lequel les agents se situent. L'environnement est modifiable par les agents, soit de façon globale, soit en faisant la distinction entre objets passifs (soumis aux actions des agents) et entités actives (les agents) [Fer95].

Dans [Aza07] l'auteur présente l'environnement sous plusieurs angles :

- Environnement social (par opposition à l'environnement physique): il correspond aux autres agents avec lesquels un agent est en interaction par envoi de message. Alors que, l'environnement physique est constitué des ressources matérielles que l'agent peut percevoir ou sur lesquels il peut agir. Par exemple, dans une société de fourmis, l'environnement physique est composé de nourriture et d'obstacles.
- Environnement accessible (par opposition à "inaccessible"): le système peut obtenir une information complète, exacte et à jour sur l'état de son environnement. Par contre, dans un environnement inaccessible, seule une information partielle est disponible. Par exemple, un environnement tel que l'Internet est inaccessible car il est impossible de tout connaître à propos de lui.
- Environnement continu (par opposition à "discret"): dans le quel le nombre d'actions et de perceptions possibles est infini. A l'inverse, si ce nombre est limité, l'environnement est dit discret. Ceci est le cas dans un environnement simulé tel qu'un éco-système.
- Environnement déterministe (par opposition à "non déterministe"): une action effectuée dans un tel environnement a un effet unique et certain. De plus, l'état suivant de l'environnement est complètement déterminé par l'état courant. Dans un environnement non déterministe, une action n'a pas un effet unique garanti. Par exemple, le monde physique réel est un environnement non déterministe.
- Environnement dynamique (par opposition à "statique"): son état dépend des actions des processus appartenant à cet environnement mais aussi des actions d'autres processus externes. Aussi, les changements ne peuvent pas être prédits par le système. Par exemple, l'Internet est un environnement hautement dynamique, son changement n'est pas simplement lié aux actions d'un seul utilisateur.

1.5.3. Interaction

Ferber [Fer95] définit les interactions comme étant la mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques. L'interaction entre agents s'effectue par la communication, les actes de langages et les protocoles d'interaction. Pour atteindre son objectif ou pour améliorer la coordination des actions, un agent peut demander des services à un autre agent.

Les interactions définissent un contexte organisationnel qui définit la nature des rapports entre agents et qui influence le comportement individuel des agents. Il est donc important de représenter explicitement ces rapports qui sont sujets à un changement continu. Les rapports existants évoluent dans le temps et de nouvelles relations se créent.

Dans le travail de [Aza07] l'auteur distingue différents types d'interaction que les agents peuvent adopter comme : la communication, la coopération, la négociation et la coordination.

L'interaction se base sur des protocoles qui permettront aux agents d'échanger des messages structurés. Le fait d'interagir va permettre à l'agent de partager de l'information, d'être au courant de ce qui se passe autour de lui, d'atteindre ses buts et d'éviter autant que possible les conflits [Ghb00]. Dans ce qui suit, nous allons décrire quelques modes d'interaction ainsi que les protocoles sur lesquels ils se basent comme illustré dans [Aza07] :

La Communication

La communication entre les agents est un élément très important dans les SMA. Elle peut être directe ou indirecte. Celle indirecte est adoptée par les agents réactifs. Elle se fait à travers l'environnement ou bien via un tableau noir. Dans une communication par environnement, les agents laissent des traces ou des signaux que les autres agents peuvent percevoir. Dans ce genre de communication il n'y a pas de destinataire bien défini.

Cependant la communication directe, concerne les agents cognitifs. Elle est intentionnelle et se fait par envoi de messages à un ou plusieurs destinataires. Ce type de communication (directe) se base sur les trois éléments suivants :

- Le langage de communication qui permet de structurer les messages échangés entre les agents tels que KQML [Fin95] et FIPA ACL [Fip02].
- L'ontologie qui fournit un vocabulaire et une terminologie compréhensible par tous les agents. Elle concernera les symboles non logiques du contenu du message.
- Les mécanismes de communication quant à eux permettent de stocker, rechercher et adresser les messages aux agents. Ces mécanismes sont présents dans les plates-formes multi-agents comme Jade [Bel99] ou MadKit [Fer00].

La Coopération

Les agents coopèrent pour atteindre un but commun ou individuel. Ce besoin vient essentiellement du fait qu'un agent ne peut satisfaire son objectif individuellement mais avec l'aide des autres agents du système. Dans la coopération, une tâche sera décomposée entre plusieurs agents afin d'être réalisé par plusieurs mécanismes comme c'est le cas du Contract Net [Smi88].

La Coordination

Elle consiste à synchroniser les activités des agents ou à régler les conflits qui existent entre eux. En effet, la coordination met de l'ordre dans le processus global effectué par des agents pour éviter des problèmes dans le comportement global du système.

La Négociation

Les agents négocient entre eux en faisant des concessions ou en cherchant des alternatives lorsqu'ils se retrouvent dans un conflit : "conflits d'intérêts ou de buts, accès à des ressources ou proposition de plusieurs solutions différentes à un seul problème".

Les Protocoles d'Interaction

Les protocoles d'interaction permettent de décrire explicitement les échanges de messages entre les agents. Ils représentent un schéma commun de conversation utilisé pour exécuter une tâche ou appliquer une stratégie de haut niveau. Un protocole précise qui peut dire quoi à qui et les réactions possibles à ce qui est dit. Il décrit également les enchaînements de messages qui peuvent être possibles entre les agents. Il existe différents types de protocoles d'interaction: les protocoles de coordination, de coopération et de négociation. Parmi les protocoles de coordination les plus utilisés et les plus connus dans les SMA, il y a le protocole Contract Net [Smi88] qui permet aux agents de coordonner leurs activités en établissant des contrats afin d'atteindre leurs buts.

1.5.4. Organisation

Étant donné que les SMA peuvent être considérés comme une société d'agents coopérant ensemble pour accomplir collectivement un objectif donné, il est nécessaire de résoudre un problème d'organisation, généralement de façon dynamique.

Selon Fox [Fox81], une organisation peut être définie comme une structure décrivant comment les membres de l'organisation sont en relation et interagissent afin d'atteindre un but commun. De plus dans [Que02] l'auteur définit l'organisation comme étant la topologie d'un groupe permettant de spécialiser les agents en fonction de leurs compétences. Elle peut également contenir des liens organisationnels et des règles sociales qui contraignent le comportement des agents. L'étude d'une organisation en SMA est composée de :

- La structure organisationnelle qui décrit l'organisation en terme de rôles, tâches, responsabilités, et
- L'entité organisationnelle qui instancie cette structure. (c'est-à-dire l'affectation des agents aux rôles décrits dans la structure organisationnelle.

Querrec [Que02] ajoute aussi qu'une organisation peut être :

- Explicite: les rôles sont définis a priori. Avant la simulation, le concepteur connaît la structure organisationnelle.
- Emergente: les rôles sont observés a posteriori. Aucune structure organisationnelle n'est connue du concepteur. C'est en observant les interactions entre les agents qu'il se fait une représentation de l'organisation.
- Statique: la structure ou les entités organisationnelles sont fixées a priori et ne peuvent évoluer durant la simulation.
- Dynamique: la structure ou les entités organisationnelles peuvent évoluer au cours de l'exécution.

L'autonomie et le comportement proactif des agents constituant les SMA suggèrent que la conception de ces applications peut être réalisée en imitant le comportement et la structure des organisations humaines, car l'une des missions principales des SMA est de supporter et/ou de contrôler des organisations du monde réel. Un SMA peut, par exemple, aider à contrôler les activités du commerce électronique. Selon [Zam00], la perspective organisationnelle conduit à une caractérisation générale d'un SMA telle que décrite dans la Figure 1.3.

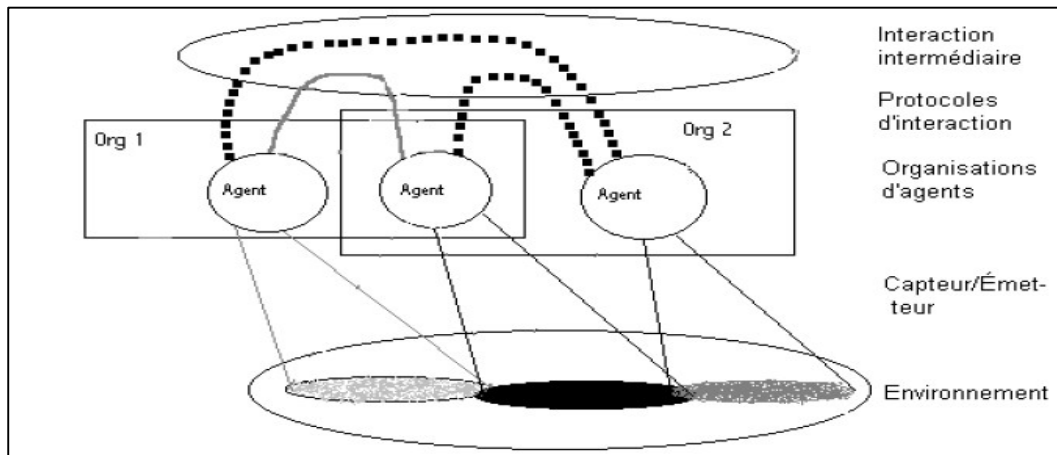


Figure 1.3. Caractérisation d'un SMA [Zam00].

Le système peut être décomposé en des sous-organisations distinctes. Un agent peut jouer un ou plusieurs rôles tout en coopérant et en respectant ses sous-organisations. Les interactions entre agents apparaissent via les capteurs et les émetteurs. Le rôle d'un agent détermine la portion de l'environnement dans laquelle il peut recevoir et émettre des données.

Dans notre approche pour la conception et la formalisation de modèle organisationnel de système multi-agents temps réel, nous avons choisie les deux modèles d'organisation, Agent-Groupe-Rôle et MOISE⁺ qui seront détaillés dans le chapitre suivant.

1.6. Points Forts & Faibles du Paradigme Orienté Agent

Bien qu'il soit né dans les années 80, le paradigme orienté agent reste encore d'actualité puisqu'il propose des mécanismes d'abstraction qui permettraient probablement de résoudre plusieurs problématiques rencontrées actuellement. En effet, les recherches menées autour de ce paradigme élaborent des mécanismes pour la construction de systèmes complexes évoluant dans un environnement dynamique et ouvert (c'est-à-dire dont la cardinalité et la topologie des composants le constituant peut être variable). Ces systèmes complexes comportent un grand nombre de composantes en interaction. L'organisation de ces composantes peut être *à priori* inconnue. Le contrôle de tels systèmes ne peut pas se faire de manière centralisée. Pour traiter ce type de système, le paradigme orienté agent propose une nouvelle philosophie de développement qui va au-delà d'une simple décomposition conceptuelle ou fonctionnelle comme il se fait avec les autres paradigmes de développement. Suivant le paradigme orienté agent, les composantes du système appelées agents sont des entités [Fer95, Pic04, Jen98b] :

- Autonomes : dont le comportement est guidé par des objectifs,
- Réactives : capables de percevoir leur environnement et de réagir face aux événements survenus dans celui-ci,
- Proactives : capables d'agir sur leur environnement,
- Sociales : capables d'interagir et de communiquer avec d'autres agents afin de coopérer en vue d'atteindre un but global.

D'autres caractéristiques peuvent être attribuées aux SMA mais selon le contexte, ces caractéristiques ne sont pas toutes nécessaires. Leur intégration dans le système final dépend des besoins fonctionnels et non fonctionnels liés à chaque domaine. Parmi ces caractéristiques, il y a :

- Le raisonnement: un agent peut décider quel but poursuivre ou à quel événement réagir, comment agir pour accomplir un but, suspendre ou abandonner un but pour se consacrer à un autre,
- L'apprentissage: l'agent peut s'adapter progressivement à des changements dans des environnements dynamiques grâce à des techniques d'apprentissage,
- La mobilité: dans certaines applications déterminées il peut être intéressant de permettre aux agents de migrer d'un nœud à un autre dans un réseau tout en préservant leur état lors de leur migration.

Dans ce qui suit, comment ces caractéristiques peuvent être avantageuses pour le développement de système complexe est expliqué. Pour ensuite, présenter les principales difficultés empêchant une utilisation à grande échelle de ce paradigme.

1.6.1. Avantages des Systèmes Multi-Agents

Les systèmes multi-agents héritent beaucoup de choses du domaine de l'intelligence artificielle comme le traitement symbolique (au niveau des connaissances), la facilité de maintenance, la réutilisation et la portabilité ainsi que des avantages traditionnels de la résolution distribuée de problèmes [Cha01, Bar03] :

- L'ouverture: c'est la capacité d'ajouter ou de retirer dynamiquement dans le système des agents ou des fonctionnalités et des services [Kle09]. Ainsi, il est impossible de savoir lors de la conception quels seront les composants du SMA ni comment ils vont interagir les uns avec les autres [Woo01].
- L'interaction: les SMA permettent de modéliser un ensemble d'agents qui interagissent. Ces agents ont une influence localement sur le comportement des autres agents, généralement sur un pied d'égalité. On rencontre de nombreuses interactions entre agents telles que la coordination (organiser la résolution d'un problème de telle sorte que les interactions nuisibles soient évitées, ou que les interactions bénéfiques soient exploitées), la négociation (parvenir à un accord acceptable pour toutes les parties concernées), la coopération (travailler ensemble à la résolution d'un but commun) [Cha01]. Cette approche est particulièrement bien adaptée à la simulation des systèmes complexes dont le fonctionnement global émerge des actions des individus. Les SMA permettent de faire vivre virtuellement des agents autonomes sur ordinateur et d'y effectuer des expériences difficiles, voire impossible à mener dans la réalité, d'où la qualification de laboratoires virtuels [Tre97].
- La souplesse de l'outil informatique: elles permettent de rendre la programmation plus simple. Elles permettent aussi de modifier le comportement des agents, d'ajouter ou de supprimer des actions possibles [Bar03].
- La décentralisation: la décentralisation des agents permettent au SMA de supporter l'échec individuel d'un des éléments ou l'ajout d'un nouveau sans dégrader le système dans son ensemble. Le fonctionnement du système ne dépend en effet d'aucun agent particulier.
- Il n'y a pas de contrôle global: le contrôle et les responsabilités sont partagés entre les différents agents, ce qui le rend fiable, robuste et flexible.
- La résolution distribuée de problèmes : un problème peut être décomposé en sous-parties, chacune pouvant être résolues de façon indépendante pour aboutir à une solution stable.

- La vitesse d'exécution: elle est principalement due au parallélisme, car plusieurs agents peuvent travailler en même temps pour la résolution d'un problème.
- La résolution plus rapide et efficace de problèmes en exploitant le parallélisme.
- La réduction des coûts de développement de logiciel : plus un logiciel est modulaire, plus la complexité et les coûts de développement diminuent.

De plus, selon l'auteur de [Zam98] le paradigme orienté agent est très approprié pour la modélisation de systèmes ouverts, flexibles et auto-adaptables. En effet, ce paradigme veut interconnecter des ressources logicielles de manière à créer un système intégré permettant de résoudre des problèmes complexes et de s'auto-adapter à différentes situations. Ainsi, les SMA aident à garantir :

- L'extensibilité : elle peut correspondre à de nouveaux objectifs pour le système, ce qui induit l'intégration de nouvelles entités qui vont coopérer avec celles qui existent (*dans ce cas, les capacités sociales des agents sont très utiles*). Il pourrait s'agir aussi d'ajouts progressifs d'hypothèses sur des comportements locaux au sein des agents (ceci est rendu possible grâce notamment aux capacités cognitives des agents c'est-à-dire les capacités de raisonnement et d'apprentissage),
- La portabilité : c'est-à-dire l'aptitude d'un logiciel à être transféré d'un environnement à un autre. Les systèmes multi-agents sont axés sur *les interactions*. Celles-ci se basent sur l'utilisation de langages de communication normalisés tels qu'ACL/KQML qui décrivent la sémantique des messages. Ainsi, les interactions dépendront de la compréhension du contenu des messages et non de la configuration des messages échangés. Ceci réduit fortement le couplage entre les composantes des systèmes,
- La performance : celle-ci est liée aux possibilités intrinsèques aux agents, d'exécution parallèle et d'asynchronisme. La technologie agent doit aussi fiabiliser l'exploitation des ressources disponibles en traitant les cas de concurrence entre agents,
- La robustesse et la tolérance aux pannes : celles-ci peuvent être garanties par les capacités d'adaptation des agents aux différentes situations qui peuvent surgir. Des études sont menées pour permettre à l'agent de diagnostiquer les pannes et d'adapter son comportement par le choix des meilleures manières d'atteindre les buts en fonction du contexte.

Donc, le paradigme agent vise à proposer des mécanismes pour simplifier l'utilisation et l'intégration de plusieurs systèmes hétérogènes. Il permet de minimiser encore plus le couplage entre les différentes entités du système car les agents ont des capacités cognitives leur permettant de gérer leurs interactions. Les agents se distinguent aussi par leur capacité de perception de l'environnement. Cette capacité n'est pas prise en compte de manière explicite dans les autres paradigmes de développement où la distinction entre entités actives du système (qui vont correspondre aux agents) et entités passives du système (qui vont correspondre aux ressources) est peu explicitée [Zam98].

1.6.2. Difficultés de l'approche orientée-agent

D'après l'auteur de [Aza07], la grande majorité des systèmes multi-agents utilisés dans les applications industrielles ont été construits de manière *ad hoc* sans utiliser des composants agents réutilisables. De ce fait, il affirme de la nécessité à développer des techniques d'ingénierie (méthodologies, méthodes, notations spécifiques, etc.) facilitant la réalisation de ces systèmes.

Dans le monde académique, plusieurs travaux sont proposés, tentant d'absorber ce problème. Parmi eux, il y a :

- Des méthodologies de développement [Igl99, Ber04, Bra97, Cos01, Ber05, Woo00b] qui permettent de guider les développeurs durant les phases de développement,
- des plates-formes orientées agents [Bel99, Col98, Fer00] qui permettent de faciliter la phase d'implémentation,
- Des langages de modélisation permettant de décrire des modèles selon les concepts orientés agents [Bau01a, Cai01, Stu03, Wag03].

Cependant, vu la diversité et le manque de maturité des travaux proposés, ils ne sont utilisés que dans le milieu académique. Le problème est lié essentiellement au manque de consensus concernant la sémantique des concepts orientés agent ainsi que leur utilisation concrète durant le cycle de développement. Par exemple, il arrive souvent que certains concepts soient supportés par certaines méthodologies ou plates-formes alors qu'ils sont complètement omis par d'autres. Ces omissions peuvent constituer un réel problème pour le développeur qui se voit obligé de modéliser ou d'implémenter certaines parties du système de manière ad hoc. A cela se rajoute l'inexistence de liens entre les méthodologies et les plates-formes d'implémentation dans pas mal de cas.

En conséquence à cela, le passage des modèles vers le code devient alors une lourde tâche réalisée pour la plupart du temps de manière manuelle par le développeur. Il devient alors nécessaire d'adopter une approche de développement plus cohérente permettant d'intégrer les différentes facettes du système multi-agent et de faciliter le travail des développeurs tout au long du cycle de vie.

Dans ce contexte, si l'un compare le développement orienté agent par rapport à l'approche orienté objet. Il trouve que le développement d'applications orienté objet est fondé sur des standards comme OMT (en anglais "Object Modeling Technique" c'est-à-dire technique de modélisation objet) et UML (Unified Modeling Language) pour la modélisation, la conception et l'implémentation. Cependant, la même chose n'est pas possible pour le développement d'application orienté agent car aucun standard n'existe encore aux niveaux des méthodologies et des techniques de développement à l'exception de la FIPA (Foundation for Intelligent Physical Agents) pour l'implémentation des SMA [Fip97]. Par contre, les standards définissent beaucoup plus le niveau architectural et les modes de communication que le niveau conceptuel qu'impliquent la modélisation et la conception. Si la création d'un SMA exige impérativement l'utilisation d'une méthodologie, il faut donc déterminer celle qui représente le mieux le type de système à développer. Un tel choix suppose une connaissance relativement bonne des différentes méthodologies disponibles, ce qui est rarement le cas [Gar03].

1.7. Domaines d'application des Systèmes Multi-Agents

Les domaines d'application des systèmes multi-agents sont particulièrement nombreux a cause de leur grande souplesse d'utilisation puisqu'il n'y a aucune limite fixée sur les comportements des agents ni sur leur manière de s'exprimer mais aussi a cause de leur pouvoir de représentation d'un univers intuitif ou chaque composante d'un système peut être modélisée en un agent [Gli05].

Parmi les applications des SMA, les auteurs de [Fer99, Cha01] citent :

- La résolution de problèmes qui concerne toutes les situations dans lesquelles des agents logiciels, purement informatiques, accomplissent des tâches utiles aux êtres humains.

- La robotique distribuée porte sur la réalisation d'un ensemble de robots qui coopèrent pour accomplir une mission et utilise des agents concrets qui se déplacent dans un environnement réel.
- La simulation multi-agents : la simulation est la démarche scientifique qui consiste à réaliser une reproduction artificielle, appelée modèle, d'un phénomène réel que l'on désire étudier, à observer le comportement de cette reproduction lorsqu'on en fait varier certains paramètres, et à en induire ce qui se passerait dans la réalité sous l'influence de variations analogues. La simulation multi-agents offre un outil puissant pour expliquer les changements au sein d'un système naturel ou artificiel en fonction des changements des individus qui le composent [Gue03]. Les phénomènes émergents peuvent être détectés et expliqués par un observateur externe qui est l'utilisateur de la simulation. La détection de ces phénomènes émergents repose souvent sur l'interprétation par l'observateur externe de résultats de simulation [Mul00]. Elle est utilisée pour l'étude de phénomènes complexes réels ou virtuels dans les domaines tels que la physique, la chimie, la biologie, l'écologie, la géographie et les sciences sociales. Elle permet de représenter et de simuler des systèmes faisant intervenir un grand nombre d'individus (écosystèmes, sociétés d'insectes, dynamiques de populations), de modéliser des environnements virtuels (jeux vidéo, cinéma). Enfin, elle permet d'expliquer l'impact de comportements individuels à un niveau global de description.

En effet, dans [Boi04] ces domaines d'application sont classés en trois familles:

- Les systèmes purement informatiques, utilisés dans le domaine des télécommunications et de la simulation, tels que: les supports d'aide à la décision, le commerce électronique, la supervision de réseaux, la simulation de systèmes sociaux et naturels, etc.
- Les systèmes sociaux nécessitant une forte interaction avec l'être humain, comme les collecticiels, la recherche d'information, l'automatisation des processus industriels ou administratifs etc.
- Les systèmes mécaniques qui intègrent des composantes mécaniques opérant dans le monde réel, tels que la robotique, les systèmes industriels, les applications embarquées et sans fil etc.

Ainsi, les SMA peuvent être utiles dans plusieurs contextes. D'ailleurs, certains systèmes industriels tels que la gestion des ressources hydrauliques [Bar02], le contrôle du trafic aérien [Lju92], les télécommunications [Esf96] ont appliqué ce paradigme. Ceci est valable aussi pour le domaine d'E-Learning [Mil01, Wil04] ce qui démontre que les SMA sont très bien adaptés à un nombre important de domaines. Cependant, il existe encore plusieurs freins empêchant leur utilisation à grande échelle. Tels que décrit ci-dessus.

En ce qui nous concerne, nous nous sommes intéressés à l'utilisation des systèmes multi-agents dans le domaine de Gestion de Chaîne Logistique ainsi qu'au Système de Gestion de Conférence que nous traitons dans les chapitres 4 et 6 respectivement.

2. Les Systèmes Multi-Agents Temps Réel

De nos jours, l'utilisation du paradigme des systèmes multi-agent dans les systèmes temps réel (STR) offre la flexibilité, l'adaptabilité et la distribution. Ces caractéristiques aident à relever les nouveaux défis dans ce domaine comparé aux applications temps réel traditionnelles.

En conséquence, le concept des Systèmes Multi-Agent Temps Réel (qu'on note SMATR) a émergé comme solution intéressante pour le développement des systèmes temps réel.

2.1. Définition

Selon les auteurs de [Bot95], un Agent Temps Réel (ATR) est un agent ayant des contraintes temporelles dans quelques une de ses responsabilités. Généralement, un ATR est un agent composé d'une série de tâches, dont quelques unes ont des contraintes temporelles. Dans ces agents, il est aussi nécessaire de prendre en compte l'exactitude temporelle, qui est exprimé au moyen d'un ensemble de restrictions temporelles imposé par l'environnement. Par conséquent l'ATR assure la satisfaction de ses restrictions temporelles. Si le non respect de ces restrictions temporelles engendre des conséquences catastrophiques, l'agent est de type *Hard*. D'autre part, si l'insatisfaction de ces mêmes restrictions n'engendre qu'une dégradation dans les réponses alors l'agent est de type *Soft*.

La définition suivante d'un SMATR proposée dans [Jul04a] est basée sur la définition ci-dessus d'un ATR : *"Un système multi-agents temps réel est un système multi-agents où au moins un de ses agents est un agent temps réel"*.

Dans ce qui suit, nous allons décrire les principaux concepts des STR pour bien comprendre la mise en œuvre du temps dans les SMA.

2.2. Les Systèmes Temps Réel

Les systèmes temps réel, sont vus comme un ensemble d'actions qui opèrent sur des données en prenant en compte l'écoulement du temps et les événements qui apparaissent au niveau du système réactif ou de son environnement. Ils sont caractérisés d'une part par leurs fréquences d'arrivées et des durées d_{min} et d_{max} , et d'autre part, par la contrainte de délai entre un événement en entrée et un événement en sortie. Les données sont caractérisées par une fréquence de mise à jour et possèdent une durée de validité.

2.2.1. Définition

Les systèmes temps réels sont une classe de systèmes informatiques réactifs dont le but est de suivre et piloter les évolutions d'un procédé dynamique avec des contraintes de temps physique [Edw97]. Le temps de réaction du système aux stimuli du procédé doit être assujéti à la dynamique du procédé. La correction d'un STR ne s'évalue donc pas uniquement du point de vue fonctionnel (justesse des résultats calculés) et temporel (enchaînement correct des événements) mais également du point de vue *temps réel* (dates d'arrivée des événements) : "un résultat juste mais hors délai est un résultat faux" [Sta88].

Ces systèmes sont aujourd'hui présents dans des domaines tels que l'automobile, l'avionique, le spatial mais aussi le contrôle de chaînes de production, de processus chimique, etc.

Le caractère critique souvent attaché aux STR résulte principalement du souci de préserver le procédé (par exemple un engin spatial), son environnement (par exemple une centrale nucléaire) et les personnes (par exemple un engin de transport). En conséquence, il est nécessaire d'appliquer des techniques de validation tout au long du processus de développement de tels systèmes [Sho98].

2.2.2. Les Contraintes Temps Réel

Pour décrire les contraintes temps réel on se réfère au travail de [Mam98] où l'auteur juge que dans le cycle de vie d'un STR, ces contraintes apparaissent dès la phase d'expression des besoins. Ainsi, lorsque sont spécifiées les fonctionnalités du système liées au contrôle du procédé, les contraintes temps réel nécessaires à la garantie de ce contrôle doivent être données. En fait, les systèmes réactifs temps réel, sont vus comme un ensemble d'*actions* qui opèrent sur des *données* en prenant en compte l'écoulement du temps et des *événements* qui apparaissent au niveau du système réactif ou de son environnement. De ce fait, les contraintes temps réel doivent être exprimées sur ces trois éléments principaux, comme présenté ci-après.

2.2.2.1. Contraintes temps réel associées aux actions

Une action peut correspondre à tout traitement ayant une fonctionnalité bien précise au niveau d'un système. Selon [Jah86] une action peut être :

- Primitive: c'est une opération consommant une quantité bornée de ressource du système,
- Composite: c'est une séquence d'actions primitives.

Et parmi les contraintes temporelles portant sur les actions, il y a :

- La date de réveil: c'est l'instant où l'action peut démarrer.
- L'instant d'exécution: représente l'instant où une action commence à s'exécuter.
- La durée d'exécution: définit le temps écoulé pendant l'exécution d'une action.
- La date de fin d'exécution : c'est l'instant où l'action se termine.

2.2.2.2. Contraintes temps réel associées aux événements

Un événement est un marqueur temporel, c'est-à-dire une occurrence qui marque un point temporel ayant son importance dans la description du comportement du système. Les événements sont classés en plusieurs types [Mam98]:

- Périodique : est un événement ponctuel, mais revenant périodiquement, par exemple tous les 5 millisecondes. Il est donc caractérisé par sa période.
- Sporadique : est un événement aléatoire, mais il existe une durée minimale et une autre maximale entre deux occurrences de deux événements successifs.
- Apériodique: est un événement aléatoire dont on ne possède pas d'informations temporelles quant à sa date d'occurrence.

Cependant, les contraintes temporelles exprimées sur les événements sont :

- Une durée séparative minimale et maximale entre l'occurrence de deux événements successifs: Dans le cas des événements périodiques une seule durée (période) caractérise l'occurrence des événements. Par contre, deux durées caractérisent l'occurrence des événements sporadiques.
- Une échéance de bout en bout (End-to-End Deadline) : représente un intervalle de temps maximum entre un événement en entrée et un événement en sortie du système. Elles sont typiquement associées à une action (celle déclenchée par l'événement entrée) et se répercutent alors sur les différentes entités qui participent à la réalisation de cette action.
- Une échéance relative qui porte sur la terminaison d'une action, soit entre les deux événements date de réveil et date de fin d'exécution. Ces contraintes apparaissent par exemple avec l'éclatement de contraintes de bout en bout correspondant au découpage du logiciel.

2.2.2.3. Contraintes temps réel associées aux données

Une donnée représente toute information utilisée par une ou plusieurs actions pour s'exécuter. Les données appartiennent aux deux types suivants :

- Les données actives : ce sont des données dont la mise à jour provoque une réaction du système et qui se matérialise par le déclenchement d'actions (mises à jour d'autres données, déclenchement d'une alarme,...). La mise à jour des données suit soit une loi périodique (par exemple, la mise à jour périodique de la vitesse d'un avion déclenche la mise à jour de l'altitude [Xia95]), soit une loi sporadique.
- Les données temporelles sont des données datées qui permettent de gérer l'historique de leur évolution [Sho98]. Ces données sont soit lues sur le procédé, soit émises vers le procédé. Pour prendre des décisions correctes en tenant compte de l'état réel de l'environnement, les données manipulées par une application temps réel doivent être cohérentes du point de vue temporel.

Dans ce contexte, les contraintes temporelles exprimées sur ces deux types de données sont :

- *La contrainte de cadence* : elle caractérise un intervalle de temps entre deux mises à jour. De telles contraintes sont caractéristiques des systèmes échantillonnés.
- *L'âge d'une donnée* est l'intervalle maximum entre la date de production par le procédé et la date de consommation par toute action du système.
- *La latence* est le délai requis par le système de contrôle pour prendre en compte une donnée (c'est à dire la date maximum entre la date de production de la donnée par le procédé et la date de consommation par toute action du système).

Selon l'auteur de [Mam98], il est usuel de classer les contraintes temps réel dans deux catégories. La première est celle des contraintes "dures", qu'il faut à tout prix respecter. Alors que la deuxième catégorie concerne les contraintes temps réel "souples" dont il faut respecter autant que possible, mais qui supportent occasionnellement d'être violées.

2.2.3. Les Caractéristiques d'un système temps réel

Le temps de réponse d'un STR est une caractéristique importante, mais il ne représente pas le seul paramètre à prendre en compte. Dans [Dod89], les auteurs ont défini les caractéristiques suivantes d'un système temps réel :

- *La Réactivité* : Elle peut être définie comme étant la capacité à réagir à des stimulations extérieures. Un STR doit être réactif face aux événements externes asynchrones de l'environnement. Il doit pouvoir classer ces événements en fonction de leur urgence et donner la priorité au plus urgent. Il doit également focaliser ses ressources en fonction des données. Dans la majorité des applications temps réel (surveillance, procédés industriels,...), ces événements asynchrones sont délivrés par des capteurs.
- *L'Adaptabilité* : Elle peut être définie comme la capacité du système à adapter son comportement à de nouvelles données. Contrairement à la réactivité, cette caractéristique n'est en général que partiellement traitée. Certains modèles ne sont pas facilement extensibles ou modifiables [Gue96].
- *Les Types d'interaction (forte/faible)* : Toute application temps réel est en interaction (forte ou faible selon les cas) avec son environnement. Ce dernier peut être un procédé industriel,

un moteur d'avion, un malade, etc. la nature de l'environnement a un impact direct sur la "criticité" des actions entreprises dans une application temps réel. La notion de "criticité" (ou "criticality en anglais) est utilisée comme critère pour pouvoir classer les applications temps réel selon la sévérité, en termes de coût engendré par le non respect des contraintes temporelles. Ainsi, les applications temps réel sont classées en deux types : *les applications à contraintes temporelles strictes* ("hard real time applications"), où le non respect des contraintes de temps peut conduire à des défaillances avec des conséquences pouvant être graves, et *les applications à contraintes temporelles relatives* ("soft real time applications"), où le dépassement des échéances est considéré comme une faute bénigne [Duv01].

- Le Déterminisme : c'est le but à atteindre pour tout STR, afin d'assurer la prévisibilité, i.e., enlever toute incertitude sur le comportement des activités (individuelles et mises ensemble). Dans *un STR dur* (à contraintes strictes), on cherche à ce que toutes les échéances soient respectées, alors que dans un *STR à contraintes relatives*, on cherche à minimiser le retard moyen des activités, ou maximiser le nombre de tâches qui respectent leurs échéances par exemple [Sad04].

Cette section nous a permis de faire un tour d'horizon sur les systèmes temps réel et leurs principales caractéristiques. Notre cadre de travail concerne à la fois le domaine temporel et le domaine multi-agent. Nous cherchons à relier ces deux aspects; ainsi, dans la section suivante, nous nous intéressons à la prise en compte de la composante temporelle dans les SMA.

2.3. Mise en Œuvre du Temps dans les SMA

Des travaux issus du domaine multi-agents abordent le temps réel selon une approche plus proche de la spécification formelle de logiciel en exprimant les propriétés dynamiques de systèmes multi-agents.

- Ainsi le langage concurrent MetateM [Bar89, Fis94] est un langage de programmation orienté agents. Il est accompagné d'une logique propositionnelle et multi-modale permettant d'exprimer à la fois le temps et les croyances pour raisonner sur les programmes. L'objectif de ces travaux est également de servir de support à un raisonnement (vérification de certaines propriétés) sur les propriétés dynamiques d'un SMA. Deux procédures de décision ont été développées s'appuyant sur des modèles (Konolige [Kon84] et sémantique des mondes possibles) et des techniques (méthode à base de tableaux et méthode à base de résolution) issues de l'IA. La topologie du temps choisie est linéaire.
- Le modèle formel de SMA développé par Singh [Sin94] a également pour but de spécifier des agents, des sociétés d'agents et d'en vérifier les propriétés. Ce formalisme incorpore le temps au moyen d'un langage formel noté L qui est une extension de la logique modale propositionnelle temporelle CTL [Eme90] enrichie de quelques opérateurs temporels. Le langage L adopte un temps ramifié dans le futur et les connaissances temporelles sont formulées par rapport à l'instant présent. Cependant aucune méthode de raisonnement n'est donnée si ce n'est la proposition d'utiliser éventuellement des techniques de "Model Checking" afin de vérifier certaines propriétés.

D'autres voies ont été explorées dans les SMA, en incorporant le temps à différents niveaux comme nous allons le voir dans ce qui suit. Nous commençons par présenter en premier deux travaux décrivant la prise en compte du temps au sein de *l'interaction*. Ensuite nous passons à d'autres travaux qui concernent la prise en compte du temps dans *l'organisation*. Tous ces travaux sont donnés dans [Car01].

Dans [Loo00] les auteurs proposent une approche permettant de combiner une programmation à la fois impérative et déclarative pour spécifier et simuler des SMA. Le langage déclaratif est ainsi basé sur la logique temporelle d'Allen qui a l'avantage de fournir ce langage de haut-niveau pour décrire l'interaction. Les logiques temporelles exécutables offrent en outre, la possibilité de mettre en œuvre un raisonnement temporel (faire des déductions et raisonner sur le temps). Des messages spécifiques (*start()*, *stop()*, *mustStart()* et *mustStop()*) sont envoyés quand une contrainte temporelle impose de donner la priorité à l'action courante sur les autres actions. Cela permet de forcer ces actions à démarrer ou s'arrêter. Les contraintes temporelles sont définies à la spécification et semblablement à [Sie97], des messages spécifiquement temporels se chargent d'empêcher un blocage à l'exécution. D'autres méthodes peuvent être utilisées pour gérer le déroulement de l'interaction et les contraintes temporelles. Parmi elles, les réseaux de Pétri comme indiqué dans le travail suivant.

Le deuxième travail concerne la gestion de chaîne de production où un SMA à base de négociation par Réseau de Pétri Coloré (noté par la suite par NRPC) a été défini dans [Che99]. Les performatifs utilisés pour définir la négociation sont issus d'un sous-ensemble d'ACL FIPA mais cet ensemble peut être étendu. Le processus de négociation est modélisé par le biais d'un réseau de Pétri coloré. Ce dernier est utilisé comme un outil de modélisation pour des systèmes dans lesquels la communication, la synchronisation et le partage des ressources jouent un rôle important. En effet, le réseau de Pétri (et ses extensions) fournit les primitives pour traiter le processus d'interaction : il offre des possibilités relativement simples pour représenter et contrôler des communications (parcours d'un jeton spécifique); pour synchroniser différentes actions (spécification d'états, de places et de transitions pour y accéder) et enfin partager des ressources (accès, quantification contrôlés par des jetons spécifiques). Dans ce SMA, le déroulement des interactions et notamment du protocole est contrôlé par un "timer" générant un jeton (token) spécial. Il existe trois sortes de places : *Inactive*, *Waiting*, and *Thinking* qui correspondent aux états possibles d'un agent durant le processus de négociation. Par exemple, quand l'acheteur décide de démarrer un processus de négociation, il passe de l'état *Inactive* à *Waiting* en prenant l'action *Send Messages* (CFP). Le type de réponse est codé par une couleur spécifique (proposal, accept-proposal, reject-proposal ou terminate). A la réception du message, l'acheteur passe à l'état *Thinking*. En cas de non-réponse, le *timer* génère un jeton spécial (non-coloré) qui contrôle la continuité du déroulement.

Nous partageons le même avis que Carron [Car01] sur ces travaux qui reflètent la prise en compte du temps au sein des différentes composantes de l'interaction comme le langage de communication, les protocoles et les conversations. Cependant, l'auteur fait remarquer qu'il n'existe pas de langage d'interaction ni les autres aspects relatifs à l'interaction (protocoles, gestion de conversations) spécifiquement temporels; c'est à dire adapté à l'explicitation de ces contraintes temporelles. Les travaux qui suivent concernent la proposition de Team Planned Activity [Kin92] et celle de Jonker, Treur et de Vries [Jon01] pour la prise en compte du temps dans l'organisation que nous présentons ci-dessous.

Dans Team planned activity [Kin92], les auteurs s'intéressent à la représentation de plans communs pour des équipes d'agents situés dans des environnements dynamiques, et en particulier à la manière dont les agents s'organisent pour former des équipes adaptées à une activité complexe et synchronisée. Les auteurs veulent définir un langage pour décrire l'activité d'une équipe. Du point de vue organisationnel, les notions de rôles et d'équipes sont positionnées grâce à des opérateurs temporels particuliers (séquence, choix non déterministe et parallélisme) mais une

dimension précise et explicite du temps n'est pas exprimée. Donc, il s'agit que d'un positionnement relatif de notions entre elles mais aucune notion numérique ou de positionnement par rapport au temps global n'est permis.

Par ailleurs, Jonker, Treur et de Vries [Jon01] (noté par la suite JTV) s'intéressent à la façon dont les spécifications de la dynamique d'une organisation en termes de groupes, de rôles à l'intérieur d'un groupe et d'interactions entre rôles peuvent constituer un lien entre les spécifications des besoins d'un SMA d'une part et celles des agents individuels d'autre part. Le but est de permettre aux agents d'anticiper le comportement futur des agents selon des comportements prévus pour un rôle ou à partir des motivations personnelles des autres agents. Ils s'appuient sur le fait que d'un point de vue dynamique, une structure organisationnelle apporte des spécifications de contraintes sur les interactions et le comportement des rôles. Le modèle de structure organisationnelle est similaire à celui proposé dans MadKit [Fer00]. *Les contraintes temporelles* qu'ils définissent sont principalement basées sur les relations temporelles entre les "entrées" et les "sorties" des instances de rôles, que ce soit pour les spécifications de comportement de rôle, d'interactions au sein de groupes ou d'interactions inter-groupes. La référence au temps est faite de manière formelle : le temps est discret et borné et la primitive temporelle est l'instant. Le langage temporel défini se rapproche du calcul de situation [McC69] par son formalisme logique. Alors que le raisonnement temporel s'appuie également sur l'approche BDI [Rao95] pour représenter l'état des agents. L'intérêt du travail mené dans JTV est d'améliorer l'étendue de la coordination par la prise en compte de la composante temporelle au sein de la dimension Organisation.

De plus, selon le même auteur [Car01], d'autres travaux ont même introduit le temps dans la formalisation d'obligations et de permissions qui sont deux notions fondamentales pour formaliser et spécifier des structures organisationnelles tel que proposé dans [Dig97].

3. Conclusion

Les systèmes multi-agents représentent aujourd'hui un nouveau paradigme de conception qui se trouve à l'intersection entre l'intelligence artificielle distribuée (IAD) et la vie artificielle. Ils offrent une décentralisation des données et du contrôle, apportant ainsi de nouvelles solutions pour résoudre les problèmes complexes, mais il y a une perte de contrôle au niveau l'ensemble du système car le résultat d'un SMA n'est pas prévisible. Heureusement que les SMA sont utilisés principalement dans des environnements dynamiques, distribués, et imprévisibles.

Dans ce chapitre, nous avons présenté les principaux concepts utilisés tout au long de cette thèse, on commençant par les notions d'agents, de systèmes multi-agents, avec leurs caractéristiques respectifs afin de montrer en particulier l'importance des interactions et surtout de l'organisation dans ces systèmes. Pour aborder ensuite les systèmes multi-agents temps réel en les définissant et en présentant les principaux travaux qui les considèrent.

Ce chapitre a pour but d'aider le lecteur à comprendre le contexte dans le quel se situe notre travail ainsi que le sens précis des différents concepts employés par la suite.

Chapitre 2

La Vue Organisationnelle dans Les Systèmes Multi-Agents

Introduction	30
1. Concepts de Base	32
2. Les Méthodologies de développement des SMA	33
2.1. Les Méthodologies Orientées Agent	33
2.2. Les Méthodologies Orientées Organisation	34
3. Les Caractéristiques Organisationnelles	34
3.1. Les Topologies de l'Organisation	35
3.2. L'Emergence	37
3.3. Réorganisation et Auto-Organisation	38
4. Une étude Comparative des Modèles Organisationnels SMA	39
4.1. Modèles Organisationnels	39
4.2. Synthèse	47
4.3. Etude Comparative	49
5. Conclusion.....	52

Introduction

D'après [Gut99], le concepteur d'un SMA ne s'occupe que des actions individuelles des agents, et suppose que les structures sociales se formeront à partir des interactions. De plus, il est communément admis que le comportement d'un agent est une conséquence directe de son état mental: croyances, désirs, intentions, buts, engagements, etc. Même les travaux sur les agents réactifs [Fer96] conservent souvent cette prééminence de l'agent en considérant que les structures vont émerger des interactions entre individus, même si ces individus ne sont pas supposés avoir d'états mentaux au sens strict.

Cette multitude d'interactions, si elle permet la résolution de problèmes complexes, amène également des situations conflictuelles (conflit d'actions, accès à une ressource, etc.) et des situations où des actions redondantes sont entreprises par les agents. Il devient alors nécessaire de coordonner les interactions existant entre les agents, afin de structurer leurs activités [Foi98]. Ce problème est abordé au travers de la notion d'organisation qui est un autre aspect fondamental du paradigme multi-agents. Cette notion se préoccupe de définir, de gérer et de modifier les relations entre les individus. Elle correspond donc à un concept liant la description d'une société d'agents et celle des activités de contrôle et de coordination. Cette description se fait en termes de définition de rôles, d'assignation de tâches, de comportements, de relations d'autorité, etc. [Gas92]. Elle permet de contrôler l'ensemble des agents pour qu'ils résolvent ensemble un problème, ceci en restreignant les comportements individuels pour limiter les conflits potentiels et pour gérer le problème de cohérence globale dans le système [Foi98].

Mais l'inconvénient de cette optique centrée sur l'agent comme le voient les auteurs de [Gut99] est qu'il est souvent difficile de construire des systèmes complexes de grande taille, pour les raisons suivantes :

- Hétérogénéité des langages : La multiplicité des langages et protocoles d'interaction rend difficile la fédération d'agents non basés sur le même formalisme. L'importance de ce problème a d'ailleurs déclenché divers efforts de normalisation (KQML, FIPA, ...)
- Hétérogénéité des modèles d'agents : De très nombreuses architectures et modèles de comportement d'agent ont été proposés. Les travaux portent généralement sur un SMA conçu avec un seul modèle, objet de l'étude. La réunion, au sein d'un même système, d'agents conçus selon des architectures diverses est le plus souvent délicat.

Nous partageons la même vue des auteurs de [Gut99], qui soutiennent que la prise en compte de concepts organisationnels en tant que concepts primitifs est un point clé de la construction de systèmes complexes, et une solution possible au problème de l'hétérogénéité. En adaptant [Gas92], ces mêmes auteurs de [Gut99] définissent l'organisation comme "un Framework d'activité et d'interaction, mis en place par la définition de groupes, rôles et de leurs relations".

En évitant de centrer leurs propos sur l'agent, les auteurs de [Gut99] considèrent l'organisation comme des relations structurelles dans un ensemble d'agents. Elle peut être décrite uniquement par sa structure, c'est à dire par la manière dont sont définis et liés groupes et rôles, sans se concerner du comportement effectif des agents.

De plus, en s'appuyant sur [Aza07], cette organisation permet de structurer les différentes composantes du système ainsi que leur mode de fonctionnement global. L'organisation

s'intéresse aussi aux dimensions sociales des agents [Hab68]. Elle peut être considérée comme une structure décrivant les interactions et autres relations qui existent entre les membres de la dite organisation [Fox81] (dans le but d'accomplir un objectif commun). L'organisation peut donc apparaître comme une structure de coordination et de communication [Mal87].

D'après le travail de [Cou05], il existe deux principales visions du concept organisation. Dans la première vision, l'organisation est une entité à part entière qui est composée par un groupe d'agents. La structure organisationnelle ainsi que le comportement global sont préétablis. Plusieurs contraintes peuvent être exprimées au niveau de la topologie, des protocoles d'interaction, des normes utilisées ainsi que le déroulement des activités. Dans la deuxième vision, l'organisation émerge du comportement global des différentes composantes du système. Les différents éléments faisant partie de l'organisation sont préalablement connus et leur comportement individuel est spécifié. Cependant, le comportement global est inconnu. Ces cas sont surtout rencontrés dans les applications d'ingénierie sociale qui consistent à étudier la dynamique des structures sociales. L'objectif de ces applications est de découvrir le modèle organisationnel généré suite aux activités et décisions des différents acteurs sociaux ainsi que leur interaction, Comme c'est le cas des sociétés de fourmis [Rou98]. Dans cette vision, il est intéressant de spécifier de manière explicite au niveau de l'organisation, les règles de gestion de conflits ou d'interférences générés suite aux activités de chacun des agents [Aza07].

En ce qui nous concerne, nous allons essayer de concentrer notre effort sur l'aspect fonctionnel des agents au sein de l'organisation sans entrer dans les détails des architectures d'agents et des modèles de comportements. Cela n'implique pas comme la fait savoir les auteurs de [Gut99] que de telles structures doivent être statiques, ni que les groupes d'agent doivent être complètement indépendants. Au contraire, la puissance des structures d'organisations vient de la capacité des agents à se joindre à de tels groupes et à y acquérir de nouvelles aptitudes qu'ils n'auraient pas obtenues sinon et qu'ils utiliseront dans d'autres groupes.

Nous commençons ce chapitre par présenter les concepts de base liés à l'organisation tels que la structure organisationnelle (aspect statique), le processus d'organisation (aspect dynamique), et la relation entre les deux, afin de bien la situer dans **la section 1**. **La section 2**, permet la présentation des principales catégories de méthodologies de développement. Ensuite, dans **la section 3**, nous donnons de manière brève les différents aspects étudiés au niveau des organisations qui sont principalement: la topologie, l'émergence, la réorganisation et l'auto-organisation. La **section 4** est consacrée à l'étude des principaux modèles organisationnelles en les comparants. A la fin, une conclusion sur les concepts abordés tout au long de ce chapitre est donnée dans **la section 5**.

1. Concepts de Base

D'après Hübner [Hub07] un système multi-agent est généralement composé d'un ensemble d'agents autonomes qui doivent coordonner leurs activités afin d'accomplir un objectif globale. Cependant, cette autonomie des agents est essentielle au SMA mais peut engendrer une perte de la cohérence globale. L'organisation du SMA est utilisée pour gérer les intra- et inter propriétés des agents. Dans ce sens que Foisel [Foi98] affirme aussi que le concept d'organisation est un thème majeur du paradigme multi-agents. Il a fait l'objet de nombreux travaux [Str95], [Che93], [Mar96], [Bae96].... Ces travaux se sont souvent inspirés des ouvrages de sociologie [Dup92], psychosociologie [Pet84], économie [Min79], éthologie [Len94] ou encore sciences cognitives.

Pour le domaine des systèmes multi-agents, nous donnons la définition de Dignum [Dig04]: "Les organisations d'agents peuvent être vues comme des ensembles d'entités et leurs interactions, régulées par des mécanismes d'ordre sociaux et créées par des acteurs autonomes pour accomplir un but commun". Cette définition souligne l'aspect social des organisations d'agents tout en spécifiant le devoir pour les agents de tenir compte des objectifs du groupe pour atteindre leurs propres buts.

Dans [Fer03b], et [Sic05] les auteurs identifient les principes de bases des organisations au sein des agents suivants :

- Les interactions entre agents peuvent augmenter les structures organisationnelles. Chaque fois que les mêmes modèles (patterns) d'interaction sont répétés plusieurs fois, impliquant les mêmes agents.
- Les organisations d'agent réduisent les interactions au sein des agents, afin d'optimiser la réalisation des buts globaux. Par conséquent, le comportement collectif devient plus efficace dès que la formation d'organisation est faite en priori.
- Une organisation peut être divisée en différents groupes. Ces groupes définissent des frontières. Alors que la structure du groupe "A" peut être connue par tous les agents de ce groupe, elle est cachée aux autres agents ne faisant pas parti du groupe 'A'. Ainsi, les groupes sont obscures les uns aux autres et n'assume pas une standardisation générale des interactions et des architectures agent.
- Il est possible de construire des systèmes sécurisés en utilisant les groupes comme "des boîtes noires (black boxes)" parce que ce qui arrive dans le groupe ne peut pas être vue par les agents externe à ce groupe. Il est aussi possible de définir des politiques de sécurités qui laissent les agents indésirables en dehors du groupe.

Selon [Fox81], une *structure organisationnelle* est un modèle ("pattern") décrivant les relations entre les membres d'un système afin d'atteindre un but commun. Qu'elle soit décrite ou observée, cette structure traduit la mise en place de contraintes sociales qui restreignent l'activité individuelle pour assurer une cohérence globale de la résolution du système. Cette structuration correspond à des relations entre les agents de la société et à un ensemble de rôles affectés à ces agents [Gas92]. En effet, comme l'affirme Ferber dans [Fer95], l'adoption de rôles et de conventions est la base de toute structure organisationnelle.

Cependant, le *processus d'organisation* correspond à un savoir permettant d'aboutir ou de changer de structure organisationnelle [Foi98]. Dans le contexte des SMA, le processus d'organisation varie en fonction du type d'agents qui le composent [Str95]. Les agents de

type cognitif ou communicant composent généralement des organisations à structure imposée dès la conception du système et relativement stables. Cependant, les agents réactifs forment diverses organisations dites émergentes.

Selon [Foi98], l'étude des types d'organisation stable proposée par V. Chevrier [Che93] a montré que l'existence **d'une structure** optimale d'organisation est peu probable. Une structure d'organisation efficace face à un type de problème peut être inefficace et inopérante face à un autre. Il faut alors que les SMA soient dotés d'une structure adaptée au problème à résoudre. C'est pourquoi l'auteur conclut que l'aspect statique est insuffisant. Concernant l'aspect dynamique, les travaux de Lestrugéon [Str95] considèrent que le *processus d'organisation* correspond à une *réorganisation* globale du système permettant d'adapter le système au contexte de la résolution du problème.

En résumé, l'organisation quelle soit statique ou dynamique, il est de toute façon nécessaire de fournir aux agents une représentation de la structure de celle-ci qui reste stable ou qui est modifiée par les agents. En effet, l'organisation peut donc être considérée dans le cadre de la conception de systèmes multi-agents, comme statique et dynamique à la fois.

Pour conclure, parmi les avantages de l'approche organisationnelle pour le domaine des systèmes multi-agents cités par Ferber [Fer03a], il y a :

- La modularité: les organisations peuvent être vues comme des unités spécifiant le comportement de leurs membres. Ainsi, la conception du SMA est facilitée par la définition de règles pour chaque unité ;
- Les architectures multiples: l'approche organisationnelle ne restreint pas l'architecture interne de l'agent, ce qui permet de pouvoir implémenter tout type de modèle au niveau de l'agent ;
- L'hétérogénéité des langages: les groupes d'agents sont considérés comme des espaces d'interactions. A l'intérieur de ces groupes, les moyens spécifiques de communication peuvent être utilisés sans modifier l'architecture globale du SMA ;
- La sécurité des applications: tous les agents d'une organisation communiquent entre eux sans intervention du milieu extérieur. Cette caractéristique permet d'éviter les problèmes liés à la sécurité dans le SMA.

2. Les Méthodologies de développement des SMA

D'après les travaux de [Dig02] et [Fer03a], il existe deux catégories de méthodologies de développement SMA, celles orientées organisation et d'autres orientées agent. Dans ce qui suit, nous allons nous baser beaucoup plus sur le travail de [Arg04].

2.1. Les Méthodologies Orientées Agent

Dans les méthodologies orientées agent, le concepteur se base sur les actions individuelles des agents, de ce fait les SMA sont conçus en terme des états mentaux des agents (croyances, intentions, buts, engagements). Les structures sociales quant à elles ne sont pas spécifiées et sont supposées émerger des interactions des agents.

D'après [Arg04], les principales propriétés "des méthodologies orientées agent" sont : (i) un but commun est défini entre les agents ; (ii) les agents sont supposés être bénévoles et

coopératifs ; (iii) elles sont appropriées aux systèmes fermés et non aux systèmes ouverts et (iv) elles sont basées normalement sur une architecture agent spécifique (ex. JACK [How01]).

La plupart des méthodologies orientées agent sont des extensions de méthodologies existantes (comme les méthodologies orientées objet et celles à base de connaissance), qui prennent en charge les aspects agents. Dans [Arg04], les auteurs citent à titre d'exemple les méthodologies Gaia [Woo00b], Tropos [Giu02], Prometheus [Pad02], MAS-CommonKADS, MASSIVE [Lin99], et MaSE [Woo00c]. Plusieurs d'entre elles couvrent les phases d'analyse, de conception et d'implémentation, en utilisant quelques CASE tools.

D'ailleurs, quelques-une de ces méthodologies offre un modèle organisationnel, qui est 'une description simple des rôles sans explicité leurs relations et hiérarchie'. En outre, ce modèle organisationnel est quelque fois obtenu à travers les agents et les modèles de coordination, donc l'organisation n'est pas considérée comme une base pour l'analyse mais comme un résultat.

2.2. Les Méthodologies Orientées Organisation

Dans les méthodologies orientées organisation, le concepteur de SMA se concentre sur l'organisation du système. La perspective rationnelle sur les organisations, adoptée par Scott [Sco02], décrit l'organisation par "ses buts, structures et normes sociales". Les *buts* de l'organisation montrent comment la structure de celle-ci doit être conçue. La *structure* de l'organisation est formalisée de tel sorte que les règles qui orientent son comportement soient bien formulées et les rôles soient définis indépendamment des attributs et des relations des personnes occupant une position particulière dans la structure de l'organisation. En fin, les *normes sociales* décrivent les comportements attendus des membres et les sanctions qui doivent être appliquées en cas d'actions indésirables.

Donc, les méthodologies orientées agent ont été les premières à être développées et elles s'intéressent aux comportements individuels des agents, tout en supposant que les structures sociales émergent des interactions des agents. Alors que celles orientées organisation s'intéressent aux aspects organisationnels (structures, objectifs et normes).

Dans la section qui suit nous allons essayer de présenter quelques caractéristiques liées au concept d'organisation qui sont principalement: la topologie, l'émergence, la réorganisation et l'auto-organisation. On s'est basé principalement sur le travail de Azaiez [Aza07].

3. Les Caractéristiques Organisationnelles

Dans [Foi98] l'auteur présente les principales caractéristiques des organisations dans les systèmes multi-agents en citant :

- La dualité entre la structure statique de description de société abstraite et dynamique du processus d'organisation ;
- L'origine non définie du processus d'organisation, qui peut être imposé par le concepteur ou initié, construit et adapté par le système lui-même ;
- La nécessité d'adaptation d'une organisation, due à l'absence d'une structure optimale: la société d'agents doit pouvoir remettre en cause son fonctionnement pour mieux s'adapter à un environnement changeant.

De plus d'après [Aza07], quand on parle d'organisation au niveau des systèmes multi-agents, différents aspects sont étudiés en profondeur. Ce sont principalement : la topologie, l'émergence, l'auto-organisation et on en parle aussi de la réorganisation. Ces derniers seront présentés dans les paragraphes ci-après.

3.1. Les Topologies de l'Organisation

Les organisations multi-agents peuvent être décrites selon une certaine topologie. Ces topologies décrivent les structures organisationnelles et permettent d'imposer certaines contraintes au niveau des interactions entre les agents appartenant à l'organisation [Str95].

Dans ce qui suit, nous présentons la première classification proposée par [Hor05], qui est la plus intéressante, car elle identifie les structures les plus utilisées dans les SMA.

3.1.1. Hiérarchies

Dans une hiérarchie les agents sont organisés dans une structure d'arbre où l'agent au sommet de l'arbre a une vue plus globale que les autres se trouvant au dessous de lui (Figure 2.1). Cette approche divise un problème en un ensemble de sous problèmes simple jusqu'à ce que la solution soit évidente. Ces inconvénients sont sa fragilité, et sa lenteur (le délai nécessaire pour passer l'information par les différents niveaux).

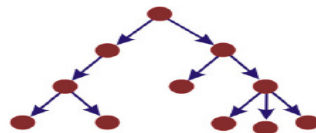


Figure 2.1. Topologie à structure de hiérarchie.

3.1.2. Holarchies

Une holarchie est une hiérarchie d'Holon autorégulés. Ces organisations permettent la conception de système complexe qui n'arrive pas à utiliser efficacement leurs ressources, très résistant aux perturbations (à la fois interne et externe), et adaptables aux changements de l'environnement dont elles faites partis. Les Holons sont autonomes, auto-reliés et coopèrent entre eux.

Les holarchies sont utiles pour les problèmes où les buts peuvent être décomposés récursivement en sous-tâches affectées à des holons individuels (Figure 2.2).

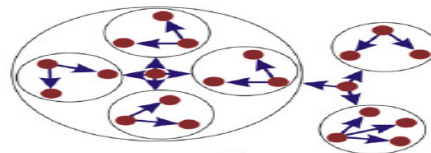


Figure 2.2. Topologie à structure de Holarchie.

3.1.3. Coalitions

Une coalition est un sous ensemble d'agents formé pour un objectif (orientée but) et dissous lorsqu'on n'aura pas besoin. Comme illustré dans la Figure 2.3, la coalition crée des partitions d'agents, et dans chaque partition les agents coordonnent leurs activités.

Son principal avantage est sa capacité à résoudre des buts avec des contraintes que n'importe quel agent ne peut offrir seul. Par contre, son inconvénient est la croissance de la complexité si le partitionnement des agents n'est pas disjoint [Cha09].



Figure 2.3. Topologie à structure de coalition.

3.1.4. Equipes (Teams)

Une équipe (team) est un ensemble d'agents coopératifs qui s'est mis d'accord à travailler pour atteindre un but commun (Figure 2.4). Par rapport aux coalitions, les équipes (teams) essaient de maximiser leurs utilités [And04]. Les agents peuvent changer leurs rôles plusieurs fois afin d'atteindre le but global.

Cette structure est une solution pour de nombreux problèmes, sa capacité à faire face à des contraintes globales, et la réalisation d'économie. Son principal défaut est le grand volume de communication.

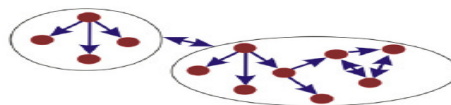


Figure 2.4. Topologie à structure d'équipe.

3.1.5. Congrégations

Les congrégations sont des groupes d'agents liés afin d'offrir d'autres services (Figure 2.5). À la différence des autres paradigmes, les congrégations sont créées pour durer plus longtemps et ne pas réaliser un seul but spécifique.

Chaque congrégation doit contenir une collection d'agents qui veulent interagir entre eux. Une fois l'existence de ces groupes, des algorithmes d'allocation (comme les ventes aux enchères 'auctions') ou des coalitions peuvent être utilisés dans chaque congrégation pour décider quel agent doit accomplir telle tâche ou échanger tel produit. Les agents de différente congrégation ne communiquent pas, mais les groupes ne sont pas forcément disjoints.

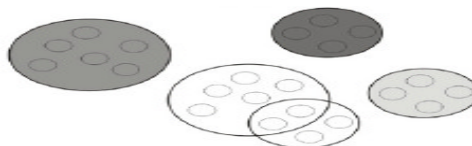


Figure 2.5. Topologie à structure de congrégation.

3.1.6. Sociétés

Les agents peuvent avoir différents buts, des niveaux de rationalités variés et des capacités hétérogènes, et se différencient par l'information qu'ils ont. La structure sociale leur offre un espace commun où ils peuvent agir et communiquer (Figure 2.6). Un ensemble de contraintes (règles sociale ou normes) est imposé sur le comportement des agents [Arc05]. Il ya deux aspects liés au problème de formation d'une société. Le premier est de définir les rôles, les protocoles et les règles sociales qui forment la base d'une société. Le deuxième problème est de déterminer comment les agents entrent et sortent de la société.

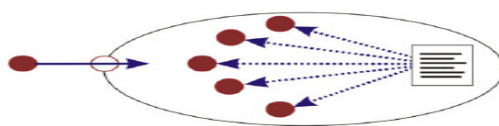


Figure 2.6. Topologie à structure de société.

3.1.7. Fédérations

Les fédérations consistent en groupes d'agents qui ont cédés un peu de leurs autonomies à un délégué qui va représenter le groupe (appelé facilitateur, médiateur, traducteur, ou courtier) (Figure 2.7). Les membres de groupes interagissent seulement avec cet agent (délégué), qui joue l'intermédiaire entre les groupes et le monde extérieur [Saw07].

Les agents intermédiaires fournissent un niveau d'abstraction pour toutes les parties liées et permettent la décomposition du problème en plusieurs sous-tâches qui peuvent être exécutées dans les différents groupes. Si la fédération est bien conçue, les agents se comportent sans le besoin d'avoir d'information sur les autres groupes.

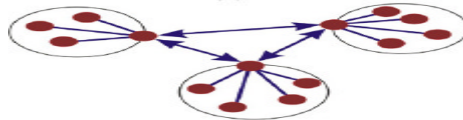


Figure 2.7. Topologie à structure de fédération.

3.1.8. Les Structures de Type Marché

Les structures de type marché (Markets) sont des collections d'agents avec deux rôles : acheteur et vendeur (illustré dans la Figure 2.8, en rouge et blanc respectivement). Les acheteurs envoient leurs offres pour un ensemble d'objets (ressources, tâches, services, ..) qui vont être reçues par les vendeurs, ces derniers les analysent pour choisir le gagnant. Dans cette structure les agents sont auto-intéressés et guidés par leurs propres buts. Les interactions émergent des communications et négociations [Gim06].

Ces structures (Markets) sont similaires aux fédérations par le fait qu'un agent se charge de la coordination des activités du groupe, mais elles sont plus compétitives et ne cèdent pas leurs autorités à d'autres individus.

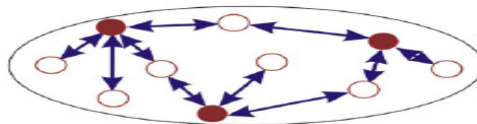


Figure 2.8. Topologie à structure de marché.

3.2. L'Émergence

Müller dans [Mul02] considère que le concept d'émergence est important pour décrire comment un phénomène non-compositionnel au niveau macro repose sur une structure d'interactions à un niveau micro (dans le sens que les descriptions au niveau macro ne sont pas une simple composition des propriétés individuelles au niveau micro). L'avantage est la possibilité pour le micro-niveau de s'auto-organiser fournissant ainsi à la fois flexibilité et robustesse malgré la possible simplicité des composants au niveau micro. De plus, Müller affirme qu'un phénomène est émergent seulement si [Mul02]:

- Il y a un ensemble d'entités en interaction dont la dynamique est exprimée dans un vocabulaire ou théorie D distincte du phénomène émergent à produire ;
- La dynamique de ces entités interagissantes produit un phénomène global qui peut être un état structuré stable ou même la trace d'exécution ;
- Ce phénomène global peut être observé et décrit dans un vocabulaire ou théorie D' distincte de la dynamique sous-jacente.

Toute fois, pour que la dernière condition soit possible, le phénomène global doit être globalement perçu et, donc, inscrit sur un support. Dans les systèmes naturels, l'environnement joue ce rôle important de médium d'inscription. C'est la même chose dans les organisations humaines où les bases de données et de connaissances sur les ordinateurs sont là pour fournir des traces des processus en cours et simultanément pour les piloter.

De nombreuses disciplines manipulent le terme " émergence", tels que la Vie Artificielle, les Réseaux Neuronaux, les Automates Cellulaires, les Systèmes Multi-Agents, la Simulation (de systèmes physiques, chimiques, biologiques, sociaux), la Robotique, l'Intelligence Artificielle Distribuée,...etc. Si on prend le cas des systèmes multi-agents, un SMA est un ensemble d'agents situés dans un même environnement et qui interagissent. De ces interactions peuvent émerger différents phénomènes. L'émergence traite donc de l'apparition soudaine non programmée et irréversible de phénomènes dans un système. Cette notion se base sur la maxime grecque "le tout est plus que la somme de chaque partie". Les phénomènes qui apparaissent peuvent être classés en trois catégories [Mar97] : l'émergence structurelle, l'émergence de comportements et l'émergence de propriétés.

3.3. Réorganisation et Auto-Organisation

Dans sa thèse [Gut01] Gutknecht considère qu'il est très peu probable d'avoir une organisation idéale de SMA pour un problème, car l'environnement qui est le contexte des agents n'est jamais complètement maîtrisé, et l'adaptation est obligatoire. De ce fait, la dynamique des organisations est toujours évoquée dans les modèles sociaux d'agents. En effet, certains travaux présentent même cette capacité de restructuration comme essentielle, à travers l'émergence de structures sociales ou des *réorganisations* permanentes.

En revanche, d'autres travaux étudient la capacité d'auto-adaptation dans les systèmes multi-agents. Ces derniers considèrent que le système doit s'adapter à son environnement par auto-organisation des agents. L'organisation finale observée est alors la plus adaptée à l'état courant de l'environnement. L'auto-organisation a été initialement étudiée et appliquée dans les secteurs de la biologie, de la physique et de la chimie, son objectif est de permettre l'évolution dynamique de l'existant en fonction du contexte, de façon à assurer la viabilité du système [Cam94]. Elle est un mécanisme très intéressant permettant de réaliser des systèmes tolérants aux pannes. Ce mécanisme permet aussi aux composantes du système de s'adapter à leur environnement soit par spécialisation des fonctions (apprentissage), soit par modification de la topologie de l'organisation. Les études sur l'auto-organisation portent sur la définition des critères de réorganisation [Aza07].

Bien que point majeur des systèmes multi-agents, peu de travaux ont été axés sur les modèles d'organisations pour SMA. En particulier les concepts de rôle, de groupe, d'organisation ou de société sont généralement utilisés d'une manière très informelle et les approches centrées sur l'organisation se sont avérées relativement peu courantes par rapport à l'ensemble des travaux [Gut99]. Mais les avantages de ces approches organisationnelles ont été repris dans les méthodologies de conception de SMA telle que Gaia [Woo99]. De plus, plusieurs (méta)-modèles organisationnels ont été proposés tels qu'Aalaadin [Fer99], Moise+ [Han02].

Pour terminer ce chapitre, nous allons étudier quelques modèles organisationnels les plus connus en se basant sur le travail de Gâteau [Gat07a].

4. Une étude Comparative des Modèles Organisationnels SMA

Les méthodologies SMA choisies pour l'analyse comparative sont considérées comme les méthodologies d'ingénierie logicielle les plus complètes, les plus largement référencées et les mieux documentées par rapport aux autres travaux existants. Chacune de ces méthodologies offrent un ensemble de concepts, de techniques et/ou de modèles pour l'analyse et la conception de SMA, mais aucune d'entre elles ne prend en compte la dimension temporelle conjointement avec les concepts organisationnels. Cette section se focalise surtout sur la prise en compte de la dimension temporelle dans les modèles organisationnels et à leur formalisations.

4.1. Modèles Organisationnels

Les modèles organisationnels tels que mentionnés dans [Gat07a] spécifient la structure et les fonctionnalités d'une société d'agents, c'est-à-dire un ensemble d'agents évoluant au sein du même environnement. Ces modèles sont plus ou moins orientés autour de la spécification de la fonctionnalité du système et/ou de la structure de l'ensemble d'agents. Dans [Gat07a] l'auteur commence par les systèmes exclusivement orientés organisation structurelle pour terminer par les spécifications fonctionnelles permettant même la coordination entre agents.

Selon [Arg08], des modèles organisationnels ont été utilisés dans la théorie des agents pour assurer l'ordre social dans les applications SMA [Dig02, Sic05]. En effet, il y a un nombre croissant de méthodologies SMA orientées vers les organisations qui essaient de faire du concept d'organisation la base du développement SMA. Ici, nous décrirons les approches SMA les plus populaires orientées vers les organisations.

Plusieurs méthodes pour modéliser les SMA centrés organisation ont été développées, telles que Gaia [Woo00b, Zam05], AGR [Fer03a, Fer05, Man07], MOISE+ [Hub02a, Boi05, Hub07], et INGENIAS [Gom05, Fue10]. Ces approches sont principalement axées sur les rôles, les groupes et les règles sociales. D'autres méthodes, telles que RT-Message [Jul04] prennent également en compte l'aspect temporel du système mais tous n'ont pas de modèle organisationnel temporel. De plus, nous souhaitons bénéficier de ces méthodes pour proposer un modèle organisationnel facilitant le développement de SMATR.

Dans ce qui suit, nous analysons et comparons ces cinq modèles organisationnels, en mettant l'accent sur leurs composants structurels, méta-modèles, et les structures organisationnelles supportées par chacun d'entre eux. Ces structures ont été tirées du travail de David [Dav11].

4.1.1. Modèles d'analyse au sein de GAIA

Gaia est utilisée dans la modélisation organisationnelle conceptuelle et suggère que les développeurs envisagent de construire des systèmes basés sur des agents comme un processus de conception organisationnelle [Lin07]. Le concept de rôle est le modèle clé à modéliser dans Gaia. Les rôles et les ressources sont réglementés par des règles organisationnelles. Les modèles préliminaires sont extraits des exigences, ce qui aide à

postuler des objectifs implicites sur les parties organisationnelles, l'environnement, les rôles et les règles d'interaction.

La première version de la méthodologie Gaia modélise les agents d'un point de vue orienté objet, mais elle a été révisée trois ans plus tard par les mêmes auteurs dans le but de représenter les SMA comme une société d'individus organisés [Cer04a, Zam05]. Les agents jouent des rôles sociaux et interagissent avec les autres suivant des protocoles associés à ces rôles. Dans cette approche, le comportement global du système est vu sur deux niveaux micro et macro. Le premier explique comment les agents se comportent suivant leurs rôles, et le second explique le modèle du comportement de ces agents. Ces contraintes sont étiquetées "règles organisationnelles et structures organisationnelles" respectivement.

L'élément central du méta-modèle de Gaia est un agent qui peut jouer un ou plusieurs rôles (Figure 2.9). Un rôle est un comportement particulier à jouer par un agent, il est défini à travers des permissions, responsabilités, activités et interactions. Quand un agent joue un rôle, il met à jour son comportement en termes de services qui peuvent être activés selon certaines pré- et postes-conditions. En plus, le rôle est décomposé en plusieurs protocoles quand l'agent aura besoin de communiquer quelques données. L'environnement décrit toutes les entités et ressources qu'un SMA peut interagir avec, tout en mettant des restrictions sur les interactions par le moyen d'actions autorisées.

Le méta-modèle de Gaia étendu ajoute quelques concepts organisationnels. L'organisation elle-même est représentée par une entité qui modélise une certaine structure (ou topologie). Les règles organisationnelles représentent les responsabilités de l'organisation, on trouve des règles de sécurité (le temps invariant que l'organisation doit respecter) et des règles de vivacité (qui définissent comment la dynamique de l'organisation doit évoluer avec le temps). Vu la relation d'agrégation qui existe depuis l'agent vers l'organisation et depuis celle-ci vers la structure d'organisation, Gaia permet la conception d'une structure hiérarchique d'agents. D'un point de vue organisationnel, les agents forment une équipe du fait qu'ils appartiennent à une seule organisation, et ils peuvent communiquer explicitement avec les autres agents au sein de la même organisation par le moyen des collaborations, quand aux organisations, ces dernières communiquent entre elles via les interactions (Figure 2.10). Si la communication inter-organisation n'existe pas, alors les coalitions et congrégations peuvent aussi être modélisées.

Dans [Lin07], les auteurs donnent une critique de la méthode Gaia:

- Les objectifs coïncident implicitement avec les subdivisions du système, et il n'y a pas de ligne directrice claire sur la manière de dériver les rôles du modèle organisationnel.
- Il est difficile de modéliser les agents entrant et sortant des sous-organisations.
- La métaphore organisationnelle est une abstraction fortement intégrée codée dans la méthodologie Gaia.

Par conséquent, Gaia considère un SMA comme une société ou une organisation, avec les éléments de cette société définis par des rôles. Cependant, il n'y a pas de manière concrète pour déterminer les rôles et leurs relations dans l'organisation [Che02].

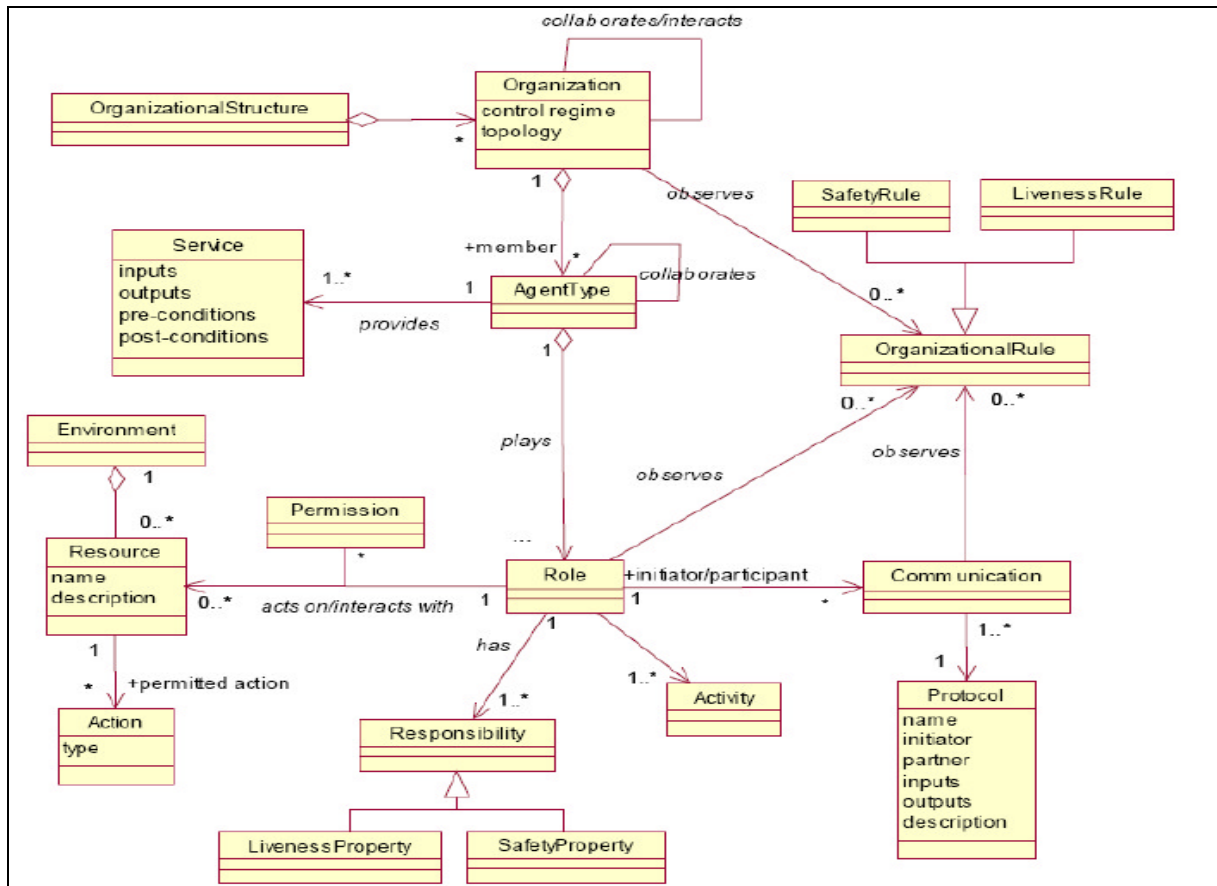


Figure 2.9. Méta-modèle de Gaia étendu [Aza07].



Figure 2.10. Topologie organisationnelle supportée dans Gaia étendu.

4.1.2. Modèle Agent-Groupe-Rôle

Le modèle organisationnel AGR pour Agent, Groupe et Rôle [Fer03a] est une évolution d'Aalaadin [Fer98] qui est un méta-modèle pour décrire les organisations de type hiérarchie ou markets. AGR [Fer03b] est basée sur les trois concepts :

- L'Agent qui est une entité active et communicante jouant des rôles au sein d'un ou plusieurs groupes. Aucune contrainte n'est définie sur son architecture interne ou ses états mentaux.
- Le groupe qui est un ensemble d'agents ayant des caractéristiques communes. Il est utilisé comme un contexte pour les modèles (patterns) d'activités, et pour créer des partitions dans l'organisation. Deux agents ne peuvent communiquer seulement s'ils font partis du même groupe, mais un agent peut appartenir à plusieurs groupes. Cette propriété facilite la conception des structures organisationnelles (e.g. markets et hiérarchie) en définissant des modèles de communication entre les rôles [Fer98].

- Le Rôle est une représentation abstraite d'une position fonctionnelle d'un agent dans un groupe. Un agent doit jouer au moins un rôle dans le groupe, toutefois il peut jouer plusieurs rôles. Ces derniers sont locaux aux groupes, et le rôle doit être joué par un agent, mais il peut être joué par plusieurs agents.

La Figure 2.11, donne un exemple de représentation graphique du niveau agent d'une modélisation AGR.

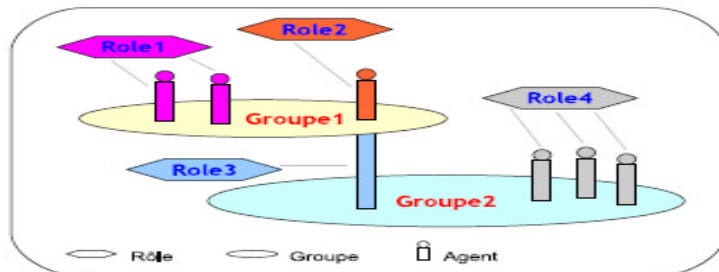


Figure 2.11. Modèle agent [Gat07a].

En effet, AGR spécifie la structure d'une organisation en termes de groupes, de rôles et d'interactions entre rôles, mais rien ne définit la fonctionnalité de cette structure et des rôles en particulier. De plus, les liens entre les rôles structurent de façon minimaliste les organisations imposées aux agents (pas de notion de hiérarchie ou d'héritage entre rôles). La description de l'organisation est abstraite et n'est plus accessible aux agents une fois l'organisation instanciée. Ce modèle permet toutefois d'organiser un ensemble d'agents de n'importe quelle taille.

Comme illustré dans la Figure 2.12, les concepts agent, groupe, et rôle sont les principaux composants du méta-modèle de la méthodologie, permettant de définir les caractéristiques et les interrelations de celle-ci.

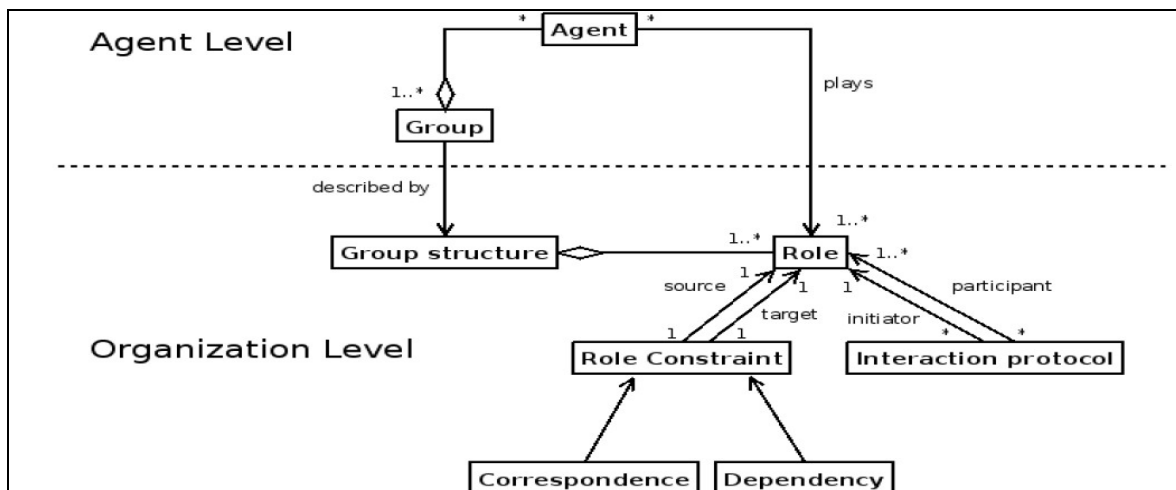


Figure 2.12. Méta-modèle d'Agent Groupe Rôle [Fer03a].

Cependant, AGR propose une vue stricte de l'organisation, dans le sens qu'elle est spécifiée comme une structure de groupe de rôle imposé sur les agents ce qui permet de définir des congrégations et coalitions d'agents, dans lesquelles les agents de même groupe sont capables de communiquer entre eux mais pas avec d'autres agents appartenant à des groupes disjoints (Figure 2.13). Un agent doit appartenir à un ou plusieurs groupes, mais il peut exister des groupes sans agents à l'intérieur. Les interactions entre les agents sont

spécifiées dans le méta-modèle à travers les protocoles d'interactions associés aux rôles joués par les agents. Les rôles dépendent socialement des structures de groupes qui définissent les groupes auquel appartiennent les agents.

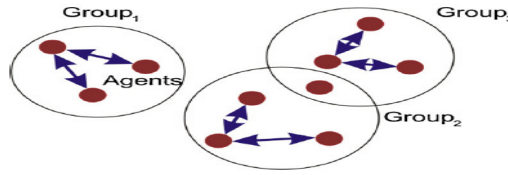


Figure 2.13. Topologie organisationnelle supportée dans AGR.

Dans AGR le comportement des agents peut être guidé par des protocoles d'interactions, mais la nature et les primitives qui décrivent de tel protocole sont ouvertes [Cou05]. Les fondateurs d'AGR proposent d'utiliser la méthodologie Gaia [Woo00b] pour réaliser les rôles et les relier à la structure générale, et ils fournissent la plate forme MadKit comme un outil d'implémentation.

Des extensions ont été établies sur le modèle AGR donnant: AGRE [Fer05] qui prend en compte l'environnement dans la modélisation et AGREEN [Bae05] qui tente d'intégrer à la fois la dimension environnementale et normative des organisations multi-agents.

4.1.3. Moise+

Moise+ (Model of Organization for multi-agent SystEms) [Hub02a] est un modèle organisationnel qui distingue explicitement trois aspects dans la modélisation d'une organisation: les aspects structuraux, fonctionnels et déontiques. L'aspect structurel définit les relations des agents à travers les notions de rôles, de groupes et de liens. L'aspect fonctionnel décrit comment un SMA atteint généralement ses objectifs globaux, c'est-à-dire comment ces objectifs sont décomposés (par des plans) et distribués aux agents (par des missions). Enfin, l'aspect déontique décrit les permissions et les obligations des rôles pour les missions.

Le méta-modèle de Moise+ (Figure 2.14), est composé de deux parties: l'agent et l'organisation. Un agent peut jouer plusieurs rôles durant son cycle de vie. L'organisation définit quant à elle un ensemble de tâches à accomplir, en spécifiant les actions, ressources et les autorisations d'exécutions de chacune. L'agent est responsable d'une partie de tâche appelée mission. L'ensemble des rôles et de mission est appelé groupe auquel l'agent va appartenir. L'exécution d'une mission suit un but que l'agent doit accomplir.

Dans Moise+, les agents et les organisations sont modélisés indépendamment l'un de l'autre. Comme illustré dans la Figure 2.15, Moise+ permet implicitement la décomposition de l'organisation en tâches, avec des parties individuelles associées aux agents. Dans ce cas, les agents ne sont pas structurellement dépendant de l'organisation du fait que les sous-tâches affectées peuvent changer durant l'exécution. Les agents d'autre part, peuvent être organisés en groupes (indépendamment des organisations) suivant leurs rôles et missions, qui peuvent aussi changer durant l'exécution. Donc, Moise+ permet la définition de coalitions orientées tâches.

En comparant Moise+ avec AGR, on constate qu'AGR ne correspond qu'à la dimension structurelle de Moise+. La dimension structurelle de Moise+ étend également les concepts trouvés dans AGR en ajoutant les notions de: rôles abstraits; héritage; composition entre les rôles; liens de communication et d'autorité. La dimension fonctionnelle (schémas sociales) de

Moise+ est une sorte de description de tâche / activité / processus basée sur la décomposition de l'objectif. Lorsque des missions (collections de buts) sont assignées à des rôles au moyen de liens déontiques (permission, obligation) le résultat est une sorte de description de protocole d'interaction [Luc05].

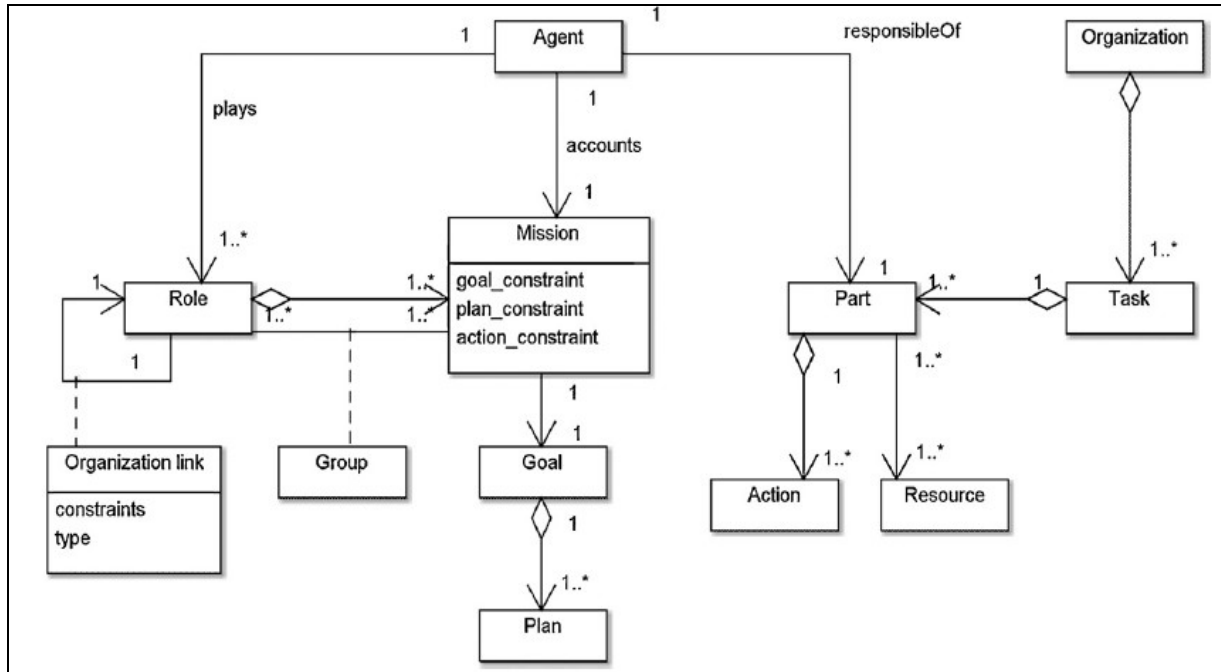


Figure 2.14. Méta-modèle de Moise+ [Dav11].

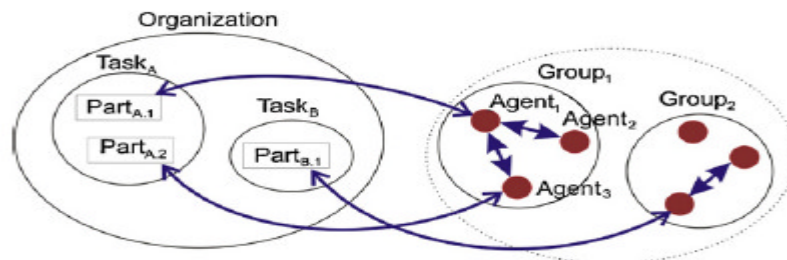


Figure 2.15. Topologie organisationnelle supportée dans Moise+.

4.1.4. INGENIAS

La méthodologie INGENIAS fournit une notation pour guider le processus de développement des SMA depuis l'analyse à l'implémentation [Fue10, Pav05].

INGENIAS ne modélise pas explicitement les normes sociales, cependant elles sont implicites du point de vue organisationnel. La dynamique organisationnelle n'est pas considérée c.à.d. comment les agents entrent ou sortent du système, comment ils forment les groupes dynamiquement, comment leurs cycle de vie est fait, etc. [Arg06]. Les auteurs ont développé un outil logiciel orienté agent (agent-oriented software tool) appelé "INGENIAS development Kit (IDK)" qui permet d'éditer des modèles consistant (selon la spécification d'INGENIAS) et de générer du code en différents langages comme JADE, Robocode, servlets ou Grasp agents.

INGENIAS fournit un ensemble de méta-modèle pour analyser tous les systèmes en utilisant le langage GOPRR (Graph, Object, Property, Role et Relationship) à partir de différentes perspectives [Kel96], qui sont comme suit (Figure 2.16) :

- L'organisation : qui est un Framework où les agents, ressources, tâches et buts coexistent. Elle est définie par sa structure, fonctionnalité et relations sociales. Elle propose la décomposition du SMA en groupes et workflows.
- L'agent : est un programme qui suit un comportement rationnel et défini à travers un objectif, ses responsabilités et ses capacités. Son comportement consiste en ses états mentaux (information qui aide l'agent à prendre des décisions), son gestionnaire d'états mentaux 'mental state manager' (opérations sur les éléments de ses états mentaux et leurs relations), et son processeur d'états mentaux 'mental state processor' (sélection d'une décision pour exécuter une tâche).
- Tâches /Buts : c'est la décomposition des buts et tâches, le résultat de l'exécution d'une tâche et la raison de son exécution.
- L'interaction : c'est l'échange de l'information ou des requêtes entre agents (ou entre les agents et l'utilisateur). Ses éléments sont les acteurs, sa spécification, son contexte et sa nature.
- L'environnement : représente les entités avec lesquels le SMA interagit (ressources, autres agents ou applications).

En ce qui concerne les structures organisationnelles, INGENIAS décompose les organisations selon deux points de vue. D'une part, les tâches se trouvent dans les workflows, et ces derniers appartiennent aux organisations. D'autre part, les agents appartiennent aux groupes se trouvant dans les organisations. Les groupes peuvent aussi appartenir à d'autres groupes. Ainsi, INGENIAS permet la définition récursive de groupes d'agents non disjoints qui peuvent interagir entre eux (Figure 2.17). En conséquence, de cette topologie et vue l'absence de contrainte dans l'association entre les agents ; groupes ; et organisation, cette méthodologie permet l'implémentation des hiérarchies, équipes orientées buts, fédérations d'agents, et application basée markets. De plus, de simple congrégation et coalitions peuvent être modélisées si les communications inter-groupes ne sont pas considérées.

INGENIAS [Gom05] suit un processus de développement itératif basé sur Rational Unified Process (RUP). Il est soutenu par de puissants outils de modélisation, de conception et de génération de code. Il utilise le langage de notation UML pour décrire leurs méta-modèles. Cependant, il manque une description normative spécifique et une analyse plus approfondie de la dynamique du système.

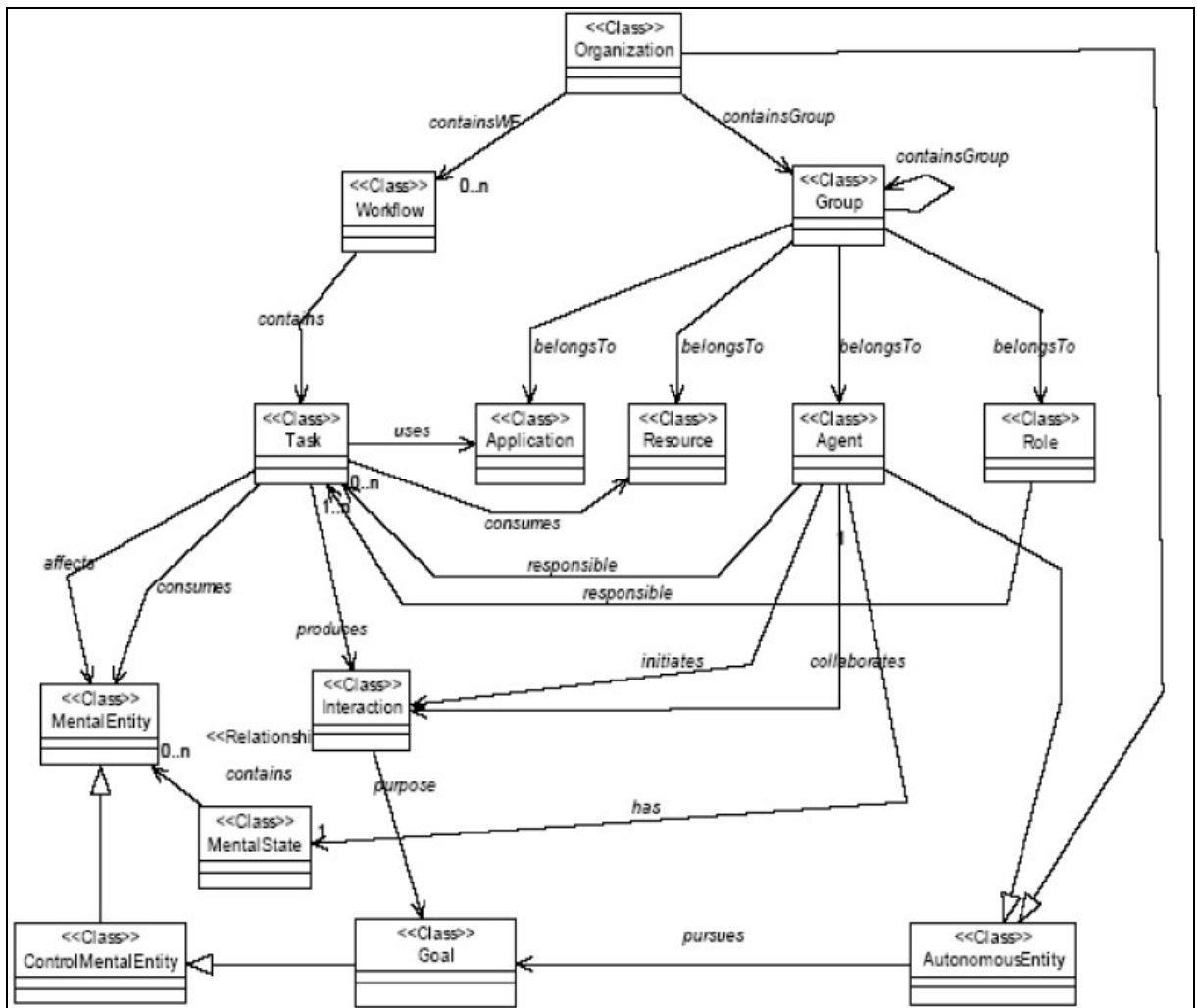


Figure 2.16. Un métamodèle Ingenias résumé [Aza07].

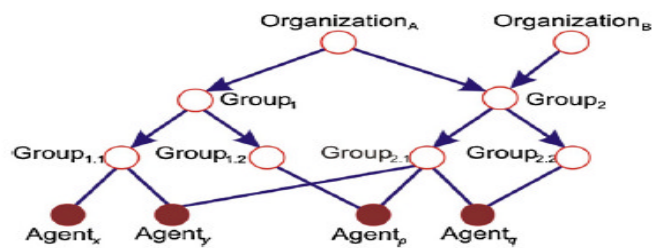


Figure 2.17. Topologie organisationnelle supportée dans INGENIAS.

4.1.5. RT-Message

RT-MESSAGE [Jul04], est un ensemble d'idées flexibles et adaptables, qui couvre l'analyse, la conception et la mise en œuvre de SMATR. La méthode est basée sur la méthodologie MESSAGE [Cai01]. Fondamentalement, RT-MESSAGE utilise le même modèle que MESSAGE dans l'activité d'analyse. La méthode ajoute des extensions temps réel afin de spécifier des comportements avec des restrictions temporelles.

Dans RT-MESSAGE, le modèle d'organisation est d'abord construit en visualisant le système en tant qu'entité unique. Le modèle montre ses interactions au sein de l'environnement, en identifiant les événements auxquels il doit réagir. Il est également nécessaire d'estimer les temps de réponse approximatifs pour tous les événements identifiés.

Deuxièmement, comme mentionné dans [Jul04], il est nécessaire de développer le modèle objectif / tâche, de trouver une correspondance entre les sous-objectifs et les buts identifiés dans le modèle d'organisation. L'identification de l'objectif doit prendre en compte les événements précédemment identifiés qui ont des restrictions temporelles à gérer par le système. La spécification de comportement de tâche doit démarrer à l'aide de diagrammes incorporant des marques temporelles.

Ainsi, il est nécessaire de développer le modèle d'agent, en générant les schémas correspondants, en accordant une attention particulière aux aspects temporels de ces agents avec des responsabilités temps réel. Après, le modèle d'interaction peut être démarré. Ce modèle permet de spécifier les processus de coopération possibles entre les agents ou les rôles identifiés dans les modèles précédents. Dans ce cas, il est nécessaire de spécifier un schéma pour chaque interaction identifiée, en indiquant les éventuelles restrictions temporelles qui lui sont associées.

Le modèle de domaine est ensuite construit parallèlement à l'ensemble du processus. Il est possible de démarrer ce modèle au début du processus, mais auparavant il est important de déterminer les événements d'environnement que le système doit contrôler, en spécifiant leurs caractéristiques temporelles possibles, et d'avoir une première idée des entités nécessaires dans le système. Ce modèle peut être démarré une fois que le modèle d'organisation a été défini.

À notre avis, ces modèles sont abstraits, en particulier le modèle organisationnel qui devrait être décrit plus clairement et plus profondément.

4.2. Synthèse

D'après l'analyse effectuée sur les modèles organisationnels présentés ci-dessus, on constate que chacun offre différents niveaux d'expressions et de possibilités de modélisation. Par conséquent, les développeurs agents doivent prendre en compte ces différences afin de choisir le modèle qui répond le mieux à leurs besoins. Cependant, il est possible de trouver des points en communs dans leurs méta-modèles.

Dans cette section, nous allons beaucoup plus présenter le travail fait par [Dav11] qui résume les principales propriétés, similitudes, et différences entre ces modèles organisationnels.

Selon [Dav11], si on analyse la portée (scope) des approches, la plus part d'entre elles sont des méthodologies complètes car elles définissent à la fois un processus du cycle de vie et un langage de modélisation. L'exception est AGR, qui ne peut pas être considérée comme une méthodologie complète parce qu'elle utilise le langage de modélisation proposé par Gaia.

Bien que, ces méthodologies définissent un processus de développement en utilisant un langage de modélisation particulier, quelques unes adoptent des standards de représentation existants pour augmenter leurs interopérabilités et niveau d'expression. Quelques diagrammes UML¹ comme les diagrammes de classe, de cas d'utilisation et de collaboration sont largement adoptés, mais ils sont étendus par d'autres diagrammes proposés spécialement par chaque approche pour répondre à ses spécificités.

¹ Unified Modelling Language: <http://www.uml.org/index.htm>

Pour un développeur agent, la possibilité d'utilisation "d'un outil de développement graphique" associé à la méthodologie peut faciliter son adoption. Par rapport aux outils supportés, Moise⁺, AgentTool et INGENIAS sont des outils de développement (development Kits) en pleines croissance (les dernières versions datent de 2009 et 2010). Seulement MadKit est considéré comme totalement développé, sa dernière version date de 2008. En plus, quelques outils de développement permettent à la fois la génération de code et son exécution, tel que Moise⁺.

Cependant, JADE [Bel99] qui est l'un des langages de programmation orienté agent les plus utilisés, est seulement exploité par la méthodologie INGENIAS. D'autres part, il ya différents langages utilisés par différents outils pour générer des prototypes ce qui constitue l'un de leurs inconvénients d'un point de vue réutilisabilité.

En ce qui concerne l'analyse de leurs méta-modèles, les auteurs de [Dav11] constatent que toutes les méthodologies ont en commun un certain nombre d'éléments : organisation, agent, rôle, tâche, interaction et contraintes ; cependant, la façon par laquelle ces éléments sont présentés diffère d'une méthode à l'autre. Bien sûr, toutes les méthodes représentent explicitement la notion de classe d'agent comme un élément atomique d'exécution auquel les buts des tâches ou rôles lui sont associés. La façon avec laquelle les agents sont liés dans le méta-modèle détermine les structures sociales supportées par la méthode, aussi l'agrégation est la relation la plus définie. Le degré de généralité de ces éléments définit la complexité des structures sociales prises en compte. Dans certains cas, les agents entrent directement dans les organisations (comme il est le cas pour Gaia Etendu) ; dans d'autres cas ils peuvent être organisés dans différentes structures, tels que des groupes (dans le cas d'AGR, et INGENIAS).

Dans d'autres méthodes telle que Moise⁺, la relation entre les agents et les organisations est décrite d'une façon indirecte, selon les tâches qu'ils exécutent ; même si quelques structures agents peuvent être définies de manière implicite, ceci entrave le niveau d'expression de la méthode d'un point de vue sociale. Les tâches sont associées aux agents suivant leurs rôles (AGR, Gaia Etendu, et INGENIAS). Seulement Moise⁺ affecte explicitement les tâches aux agents, néanmoins dans ce cas les tâches sont interprétées comme étant des problèmes globaux à résoudre par une communauté d'agents, où chaque agent exécute individuellement une partie de tâche. Dans certains cas, la classe tâche (task class) n'est pas incluse dans le méta-modèle, mais peut être implicite à la mission, au but ou plan d'agent. Les rôles et tâches sont dérivés naturellement dans les comportements agent implémentés au niveau du code, et qui résument ce que l'agent est capable de faire.

Les caractéristiques ou associations liées à l'interaction sont aussi un aspect important qui influence sur le degré d'expression des communications inter-agent. Dans la majorité des cas, les protocoles d'interaction et les messages structurés sont explicitement supportés (tels que dans Gaia Etendu, et INGENIAS). Ils sont fondamentaux lorsqu'on veut implémenter des standards et des modèles de communication interopérable comme ceux proposés par la FIPA [Fip02].

A notre avis, l'identification des points communs qui existe entre les méta-modèles associés aux différents modèles organisationnels étudiés peut aider à trouver des liens entre eux, afin de proposer un méta-modèle utilisant un ensemble d'éléments unifiés. Cependant, plusieurs tentatives ont été faites dans ce sens, pour proposer un modèle globale mais les

résultats sont restés faible. De ce fait, il est intéressant de noter que dans certain cas, le nouveau modèle organisationnel peut tenir en compte les caractéristiques de ceux qui le précèdent, en mettant à jour quelques unes de leurs propriétés et en ajoutant d'autres aspects. Dans ce sens, que nous allons proposer et valider notre modèle.

Pour les structures sociales supportées par chaque méthodologie, les auteurs de [Dav11], constatent que les hiérarchies qui sont de simple holarchie sont considérées par AGR, Gaia étendu, et INGENIAS. Néanmoins, de ces trois méthodes, seulement INGENIAS définit les hiérarchies en plusieurs niveaux de détails ; alors que dans les deux autres cas, les détails des hiérarchies sont limités à la définition de groupes d'agents.

Les coalitions, équipes (teams) et congrégations sont des structures sociales similaires, mais elles se distinguent par les possibilités de communication inter-agents qu'offre chacune (seulement pour le cas des équipes) ou non. La majorité des méthodes décrivent la notion de groupe d'agents (ou une autre notation tels que holon ou sous-organisation), et prennent en compte ces trois structures, uniquement AGR ne prend pas en charge la définition des communications inter-agent de ce fait la structure d'équipe ne peut pas être modélisée directement.

Cependant, toutes ces méthodologies (Gaia [Woo00b], AGR [Fer03a], Moise+ [Hub02a], et INGENIAS [Gom05]) ne tiennent pas en compte les spécificités des SMATR en particulier la représentation de la dimension temporelle. Pour cette raison, nous avons aussi analysé la méthodologie RT-Message [Jul04] qui aborde directement la conception de SMATR en utilisant différents modèles parmi eux on trouve le modèle organisationnel mais qui reste encore en grande partie non résolu.

En effet, la modélisation de SMATR est fréquemment liée à des concepts d'organisation car ils fournissent une base pour décrire les propriétés de l'agent, en appliquant un ensemble de techniques de génie logiciel. Parmi ces dernières, nous pensons que les diagrammes étendus AUML [Lao10] sont bien adaptés pour modéliser les concepts organisationnels de SMATR.

A la fin, cette analyse des modèles organisationnels de SMA montre qu'il est possible de concevoir un SMA en utilisant uniquement des concepts organisationnels tels que: rôles (fonction, ou position), groupes (ou communautés), tâches (ou activités), objectifs, normes et protocoles d'interaction (ou dialogue), structure. Cela offre en effet la possibilité de créer des organisations en tant que cadres dans lesquels les agents peuvent interagir. Néanmoins, ces modèles ne se concentrent pas sur la caractérisation organisationnelle de SMATR. Nous suggérons qu'ils seront complétés par des méthodes qui encouragent fortement la spécification des exigences organisationnelles temporelles capturées dès l'analyse pour les phases amont du processus d'ingénierie logicielle de SMATR.

4.3. Etude Comparative

La littérature sur les modèles organisationnels est vaste. Les modèles que nous avons décrits jusqu'à présent ont été choisis parce qu'ils touchent à la dimension clé de la modélisation organisationnelle de SMA. Nous présentons ici une étude comparative qui résume les dimensions traitées par ces modèles organisationnels. Cette analyse est basée sur les travaux de [Arg06, Gar09, Dav11], notre contribution principale est illustrée à partir des deux dimensions: le domaine du problème et le modèle de critères.

Dans ce contexte, Argente, Julian et Botti [Arg06] affirment que les méthodologies de développement de SMA orientées organisation doivent prendre en compte: les objectifs organisationnels, la structure organisationnelle (topologie, hiérarchie des rôles, interactions et normes sociales), les dynamiques organisationnelles (agents rejoignant l'organisation, implémentation de rôle), cycle de vie des agents, contrôle du comportement, et les aspects normatifs. Dans ce travail, nous insistons sur le fait que "les restrictions temporelles définies par le SMATR doivent être prises en compte conjointement avec les concepts organisationnels".

Dans le tableau 2.1, nous illustrons les principaux problèmes pour le développement de modèles organisationnels qui capturent tous les aspects critiques de la spécification de SMATR. Nous présentons également un processus préliminaire d'analyse des cinq méthodologies (Gaia [Woo00b], AGR [Fer03a], Moise+ [Hub02a], INGENIAS [Gom05] et RT-Message [Jul04]). En conséquence, dans le cadre de cette comparaison trois dimensions majeures sont identifiées, à savoir la dimension de domaine de problème, la dimension de modélisation et la dimension organisationnelle, pour fournir une évaluation complète. Les deux premières dimensions présentent notre contribution principale, tandis que dans la dimension organisationnelle, une vue structurelle, dynamique et des composantes sont développées selon les travaux analysés [Arg06, Gar09, Dav11].

Nous concluons que ces approches (Gaia [Woo00b], AGR [Fer03a], Moise+ [Hub02a], et INGENIAS [Gom05]) sont intéressantes pour spécifier l'interaction de l'agent, l'affectation du rôle et les aspects organisationnels. Elles ont modifié la construction de SMA en donnant naissance à de nouveaux concepts organisationnels, mais sans tenir compte du comportement temporel des agents du SMATR. En effet, elles ne spécifient pas les exigences fonctionnelles centrées organisation, ni utilisent les cas d'utilisation/scénario et la concurrence.

De plus, ces modèles organisationnels ne peuvent pas supporter et représenter les contraintes temporelles que les agents temps réel du SMATR doivent respecter. Pour cette raison, nous analysons aussi trois méthodologies qui abordent directement la conception de systèmes multi-agents temps réel. Nous nous sommes intéressés aux: méthodologie RT-Message [Jul04], méthodologie BDI-ASDP étendue pour le temps réel [Mel05] et méthode de développement de Zhang Lichen [Zha06]. Ces trois méthodologies utilisent différents modèles pour la description des agents temps réel, à savoir: le modèle de domaine, le modèle de rôle, le modèle temporisé et le modèle organisationnel, comme décrit dans le Tableau 2.2.

Dans RT-Message, un modèle d'organisation est créé pour visualiser le système en tant qu'entité unique. Le modèle montre ses interactions au sein de l'environnement, en identifiant les événements auxquels il doit réagir. Il est également nécessaire d'estimer les temps de réponse approximatifs pour tous les événements identifiés. À notre avis, ce modèle est abstrait et reste en grande partie non résolu, il devrait être décrit plus clairement et plus profondément. De plus, le concept de rôle n'est présent que dans la méthodologie RT-Message [Jul04], où les rôles sont identifiés indépendamment des agents du système.

Concernant la modélisation des contraintes temporelles des agents temps réel, chaque méthodologie propose une approche: pour le cas de RT-Message, les extensions réalisées sur les différents modèles importés de la méthodologie MESSAGE [Eva01] permettent d'analyser le SMATR. Par exemple, "le modèle Objectif/Tâche" a été modifié pour incorporer une taxonomie d'objectifs qui prend en compte des critères temporels.

Tableau 2.1. Comparaison des caractéristiques organisationnelles de différentes approches.

Méthodes		RT-Message	AGR	Moise+	INGENIAS	Ext.Gaia
Critères (concepts)						
Domaine du problème	Exigences Fonctionnelles	/	/	/	/	/
	Cas d'utilisation / scénarios	/	/	/	/	/
Critères de modèles	Modèle organisationnel	/	/	/	Modèle très simple	+
	Temps Réel	+	/	/	/	/
	Concurrence	+	/	/	+	/
	Formalisation	/	/	/	/	+
Caractéristiques organisationnelles :Structure	Rôles	/	Associés à des groupes et joué par des agents	Affectés aux agents, composés par missions	Appartiennent à des groupes, assignés aux agents et aux tâches	Les rôles définis pour les structures organisationnelles
	Relations sociales	Liens de communication; liens d'autorité	/	/	Liens de connaissances; Communication, Liens d'autorité	/
	Dépendances de rôles	Communication ; autorité ; coordination	/	/	Communication; autorité ; coordination	/
	Normes sociales	/	/	/	/	/
Caractéristiques organisationnelles :Dynamique	Adhésion d'Agents	/	/	/	/	/
	Promulgation Rôle	+	/	/	/	/
	Contrôle du comportement	/	/	/	/	/
	Interaction	Conception AUML	Via protocoles de communication à l'intérieur de chaque groupe	Défini en tant qu'interactions de rôle	Défini les agents participants, les spécifications et le contexte	Le modèle d'interaction utilise des actes de communication et des protocoles
Composants organisationnels du méta-modèles	Organisation	/	Représenté en tant que groupe de rôles	L'organisation définit les tâches à compléter.	Décomposé en groupes d'agents et workflows (flux de travail)	Composé de structures organisationnelles
	Agent	/	Affecté à des groupes, Jouer un ou plusieurs rôles	Exécute les parties de tâches via Les rôles	Agents basés sur des objectifs	Les types d'agent fournissent des services
	Tâche	/	Pas explicitement prise en charge. Inhérent aux rôles	Les tâches sont divisées en parties (actions)	Tâches comprises dans les workflows	Les activités affectées aux rôles joués par les agents
	Contraintes	/	Inclus dans les rôles attribués aux agents en tant que dépendances	Contraintes d'action autorisés	Associés aux ressources de l'environnement et des applications	Les contraintes affectées aux ressources et aux actions

(/ : pas d'information ; + : Concept présent)

Tableau 2.2. Approches d'identification des modèles d'agent temps réel.

	Modèle de Domaine	Modèle de Rôle	Modèle Temporisé	Modèle Organisationnel
Méthodologie RT-Message [Jul04]	+	+	+	Modèle très simple
Méthodologie BDI-ASDP étendue pour le temps réel [Mel05]	+	-	+	-
Méthode de développement de Zhang [Zha06]	+	-	+	-

(+ : Modèle existe ; - : Modèle n'existe pas)

Les artefacts obtenus sont un ensemble de "diagrammes d'implications" montrant la relation entre les objectifs et les tâches. Cependant, dans la méthodologie BDI-ASDP étendue pour le temps réel, proposé par Melián et ses collègues [Mel05], la modélisation des contraintes temporelles se fait à travers "les chronogrammes" spécifiés dans UML 2.0. Alors que dans Zhang [Zha06] l'auteur a proposé d'étendre UML en introduisant un nouveau stéréotype, appelé <<agents>> et Les caractéristiques de synchronisation sont spécifiées en tant qu'instance d'un stéréotype appelé << TimeAspect >>. Mais malheureusement, du point de vue organisationnel, aucune de ces trois méthodologies ne prend en compte la dimension temporelle conjointement avec les concepts organisationnels (Tableau 2.2).

En effet, bien que ces méthodologies apportent des réponses importantes dans le processus de développement et en particulier pour décrire les exigences de SMATR, elles ne prennent pas en compte les concepts organisationnels ou n'offrent que des descriptions informelles ou semi-formelles pour représenter les exigences organisationnelles de SMATR. De ce fait, nous voulons par notre approche offrir une représentation conjointe des deux vues (organisationnelle et temporelle) tout en profitant des avantages des approches semi-formelles et formelles qui permettent de réduire les risques de confusion et d'incompréhension entre les développeurs et les utilisateurs.

Donc, il serait intéressant de combiner les deux dimensions (organisationnelle et temporelle) en une seule technique lors de la spécification du modèle organisationnel, de sorte qu'une meilleure analyse du système puisse être faite. Nous proposons que les diagrammes étendus AUML [Lao10], puissent être appliqué dans le domaine de l'organisation des agents et pourrait servir de base pour décrire les rôles, groupes, contraintes temporelles ... etc., qui sont intrinsèquement liés aux agents temps réel de l'organisation. Le modèle organisationnel que nous voulons proposer diffère des autres approches de modélisation par l'utilisation des extensions du formalisme AUML pour décrire, dans un premier temps, les exigences organisationnelles de SMATR.

5. Conclusion

L'organisation est un concept complexe qui est associé globalement à un système et qui intervient au niveau des agents, comprenant un aspect statique et un aspect dynamique, qui doivent assurer et maintenir la cohérence globale, et contribuer à une amélioration des performances collectives [Foi98].

Dans ce chapitre nous avons présenté les principes et concepts de base permettant de mieux comprendre la vue d'organisation au sein des systèmes multi-agents. Les théories de l'organisation et l'étude de ce concept dans le cadre des SMA ont montré la dualité entre la structure organisationnelle et le processus d'organisation pour définir une organisation. Cette approche duale de l'organisation permet d'assurer et de maintenir la cohérence globale, ainsi que l'efficacité dans les SMA.

Une structure organisationnelle est un concept abstrait qui prend concrètement place à travers un ensemble de relations (des structures concrètes potentielles définies à partir de rôles et de conventions) permettant aux agents de structurer leurs activités. Cette structure permet la mise en place d'une coordination globale des relations entre les individus. Elle se décrit et s'observe à un niveau global, mais selon le principe d'autonomie défini pour le fonctionnement des agents, elle doit être pilotée à un niveau local par les agents. Concernant le processus d'organisation, il peut être entrepris par le concepteur au niveau de la conception du système, par le système lui-même lors de son démarrage ou tout au long de la vie de celui-ci. L'aspect dynamique correspond alors à l'adaptation de l'organisation de la société par les agents en remettant en cause la structure dans laquelle ils ont été placés ; ceci en fonction des situations relationnelles qu'ils rencontrent et afin de maintenir la cohérence et l'efficacité de fonctionnement du système.

Nous avons aussi analysé et comparé quelques modèles organisationnels (Gaïa [Woo00b, Zam05], AGR [Fer03a, Fer05, Man07], MOISE+ [Hub02a, Boi05, Hub07], et INGENIAS [Gom05, Fue10]), en mettant l'accent sur leurs composants structurels, méta-modèles, et les structures organisationnelles supportées par chacun d'entre eux afin de proposer un modèle organisationnel facilitant le développement de SMATR. Ces modèles sont principalement axés sur les rôles, les groupes et les règles sociales. D'autres méthodologies, telles que RT-Message [Jul04], BDI-ASDP étendue pour le temps réel [Mel05] et la méthode de développement de Zhang [Zha06] prennent également en compte l'aspect temporel du système mais tous ne permettent pas d'offrir une représentation conjointe des deux vues (organisationnelle et temporelle) c'est-à-dire n'ont pas de modèle organisationnel temporel.

Une fois de plus, notre préoccupation principale est de proposer un modèle organisationnel pour la description des SMATR. Nous voulons donc nous situer à un niveau d'abstraction supérieur à celui exprimant les systèmes et leurs entités, en se positionnant au niveau des structures sociales.

Chapitre 3

Langages de Spécification et Conception des Systèmes Multi-Agents Temps Réel

Introduction	55
1. Langages de Modélisation orientée agent	56
1.1. Unified Modeling Language (UML)	57
1.2. Agent UML	58
2. Langages de Spécification Formelle pour SMATR	63
2.1. Approches Formelles	64
2.2. Maude	66
3. Synthèse	82
4. Conclusion	85

Introduction

Le génie logiciel orienté agent (Agent Oriented Software Engineering “AOSE”) a émergé comme une discipline consacrée à l’engineering des systèmes logiciels complexes basés sur le paradigme des systèmes multi-agents. La recherche dans le domaine d’AOSE inclut l’identification et le développement d’outils conceptuels (e.g., modélisation formelle) ainsi que d’outils pratique (e.g., infrastructures basé agents) pour supporter les ingénieurs logiciels (“software engineers”) et les programmeurs dans l’analyse, la conception et le développement des systèmes multi-agents. Cependant, un grand défi à relever dans le domaine du génie logiciel orienté agent (AOSE) consiste à la définition de méthodologies pour guider le processus de développement des SMA [Gat07b]. En effet les méthodologies proposées jusqu’ici ne tiennent pas en compte la formalisation des concepts organisationnels du futur système.

Selon [Bau05], une méthodologie de développement est typiquement caractérisée par un “*langage de modélisation*” utilisé pour la description des modèles, la définition des éléments du modèle avec une syntaxe spécifique et une sémantique associée, et “*un processus logiciel*” définissant les activités de développement, les interrelations parmi les activités, et comment les différentes activités sont exécutées. Chaque activité résulte en un produit livrable ou plus, tel que les documents de spécification, les modèles d’analyse, les spécifications de conception, le code, les tests, les rapports de test, les rapports d’évaluation de performance, etc. qui servent comme entré aux activités amonts.

Au niveau de la spécification, la description des systèmes s’appuie classiquement sur des langages qui permettent de les spécifier sous différents aspects (les types de données, les fonctionnalités, les protocoles de communication, etc.). Dans ces langages de spécification différents types de notation existent, elles vont des plus naturelles aux plus formelles. Dans ce sens nous présentons les trois niveaux d’abstraction définie dans [Bow03] :

- Le niveau non formel s’appuie sur des descriptions faites en langage naturel. C’est le niveau le moins précis, qui fait apparaître le plus d’ambiguïté ;
- Le niveau semi-formel permet aux différents participants à un projet de communiquer facilement et efficacement. Parmi ses représentations, il y a l’ensemble des notations utilisées dans UML (diagramme de cas d’utilisation, de séquence, de collaboration, d’états-transitions, etc.). On peut aussi le caractériser par les langages de programmation actuels tels C++ ou Java. Bien que plus précis, ils laissent certains aspects ambigus ;
- Le niveau formel se base sur les mathématiques, il est le niveau d’abstraction le plus élevé, qui demande le plus grand effort d’apprentissage et de compréhension, mais qui ne laisse pratiquement rien d’imprécis dans ce qui sera spécifié. Les langages formels sont composés, en premier lieu, d’un alphabet (une série de symboles reconnus du langage). La seconde composante essentielle est une grammaire, c’est-à-dire une série de règles qui permettront d’assembler les symboles en des phrases correctes définissant ainsi le domaine syntaxique du langage. Le sens ou encore l’interprétation des formules sont donnés par le domaine sémantique. Ce domaine est composé d’une série de règles qui préciseront quel objet sert à remplir une certaine spécification donnée. Parmi les représentants des notations formelles il y a les langages: B [Abr91], Z [Spi92] et Maude [Cla05]. De plus, certaines descriptions non formelles peuvent être étendues par des langages formels pour l’expression des contraintes, comme c’est le cas d’OCL [Bot00] pour UML.

Afin d'arriver à un grand usage du paradigme agent, il faut l'associer à la technologie antérieure la plus proche à savoir l'approche orientée Object comme l'a mentionné Bauer dans [Bau01b], et d'utiliser les différents artefacts afin de supporter l'environnement de développement durant tous le cycle de vie du système. Ces deux besoins sont pris en compte en utilisant Agent UML (AUML), qui est un ensemble de dialectes et d'extensions d'UML. Le langage AUML (langage semi-formel de modélisation graphique) est à notre avis le futur standard de l'industrie pour le développement des systèmes orientés agents. Il permet de modéliser et de décrire plusieurs vues complémentaires d'un même système. Cependant, AUML comme UML souffre d'un manque de sémantique formelle. Les modèles AUML développés pourront donc contenir des incohérences ou des inconsistances difficiles (voire impossibles dans le cas de certains systèmes complexes) à détecter manuellement.

Les méthodes formelles représentent une solution intéressante à ce problème. Les spécifications formelles auront pour effet d'éliminer les ambiguïtés au niveau de l'interprétation des modèles. Dans ce contexte, et comme le fait savoir l'auteur de [Alk04], des travaux de recherche existent portant sur l'utilisation conjointe d'UML et d'un langage formel possédant une sémantique temps réel. Dans ce cadre, des travaux ont défini une passerelle entre les diagrammes d'états transitions et les diagrammes des collaborations et les automates temporisés et UPPAAL [Amn01a]. Cette passerelle permet la vérification formelle d'une collaboration d'objets temporisés [Gue00]. D'autres travaux ont été proposés avec l'utilisation des réseaux de Petri [Pal99], SMV [Mcm92], Z [Dup00], et E-LOTOS [Bab01].

Nous suivrons le même chemin en faisant la combinaison d'Agent UML avec un langage de spécification formelle éprouvé comme Real time Maude (RT-Maude) ce qui permettra de valider formellement les modèles AUML développés. Dans ce chapitre, seuls les langages de modélisation et de spécification formelle, supports indispensables à notre approche, seront étudiés. En particulier AUML et RT-Maude pour modéliser et formaliser les concepts organisationnels des Systèmes multi-agents Temps Réel (SMATR), Comme cela a été précisé au début de cette thèse.

Nous commençons par les langages de modélisation orientée agent, particulièrement par UML du fait qu'il est à la base d'AUML pour passer aux détails de ce dernier (**section 1**). Ensuite, on va présenter quelques langages de spécification formelle en donnant plus d'intérêt au langage Maude qui est basé sur la logique de réécriture et son extension Real Time Maude pour les systèmes temps réel (**section 2**). Une synthèse expliquant nos choix est donnée (**section 3**) avant de terminer par une conclusion (**section 4**).

1. Langages de Modélisation orientée agent

Plusieurs langages de modélisation ont été proposés dans le contexte de l'ingénierie orientée agent, parmi elles, on trouve les langages basés sur une extension d'UML, tels que AUML [Bau01a], AML [Cer04b], et d'autres réalisés par création de profil UML comme AORML [Wag03].

Dans cette section, nous présentons brièvement UML et ses extensions AML et AORML pour détailler par la suite le formalisme AUML.

1.1. Unified Modeling Language (UML)

UML est le langage le plus connu et le plus utilisé des langages de modélisation graphique orientée-objet. Il est considéré comme un standard pour les méthodes OO.

Selon Alkhodre [Alk04] UML voit le système à travers plusieurs niveaux d'abstraction exprimés par divers diagrammes. Le diagramme de cas d'utilisation est utilisé pour décrire les interactions entre l'environnement et l'application. Ce diagramme se compose de l'ensemble des acteurs, du système et des cas d'utilisation eux-mêmes. Chaque acteur représente un rôle joué par une personne ou un élément qui interagit avec le système. Le cas d'utilisation regroupe la famille des séquences d'interaction (les scénarios) du point de vue de l'utilisateur. Le diagramme de classe lui, exprime la structure statique d'un système, en termes de classes et de relations entre ces classes. Chaque classe consiste à un ensemble d'attributs qui représente les données et un ensemble des méthodes qui représente les services. Le diagramme d'instance décrit les objets qui constituent le système. Pour présenter le comportement dynamique d'une classe, le diagramme d'état transition (Statechart) définit les méthodes à exécuter en fonction de l'état courant et des messages reçus. Alors que, le diagramme d'activité permet de décrire un enchaînement d'actions. Les diagrammes de séquence et de collaboration permettent de modéliser les interactions entre les objets au cours de scénarios donnés. Ils s'appuient sur les cas d'utilisation, les diagrammes de classe et d'instance. Tous ces modèles peuvent être "décorés" de contraintes non fonctionnelles (temps, précedence, etc.) à l'aide de mécanismes d'extension d'UML et du langage de description de contraintes OCL (Object Constraint Language specification). Les architectures matérielles et logicielles d'un système peuvent être décrites en UML via les diagrammes de déploiement et de composants respectivement.

Cependant, UML ne fournit pas toutes les fonctionnalités nécessaires pour la modélisation orientée-agent mais fournit tout un ensemble d'outils pour faciliter son extension (les profils qui intègrent des stéréotypes particuliers, des contraintes, etc.). Ces outils peuvent ensuite être exploités pour intégrer les spécificités de la modélisation orientée-agent au sein du langage. Dans ce qui suit nous dérivons deux exemples: AML et AORML.

AML (Agent Modeling Language) [Cer04b] [Cer05b] est un langage de modélisation des systèmes multi-agents basés sur une extension d'UML 2.0. Il permet de décrire différents aspects tels que la modélisation des ontologies, des aspects sociaux et mentaux, des interactions, du comportement ainsi que la décomposition du comportement sur plusieurs agents etc. De plus, comme mentionner dans [Gau07] AML définit trois types d'éléments semi-abstraits nommés semi-entités :

- Les semi-entités comportementales représentent des éléments possédant des capacités propres et capables d'agir, d'observer ou de percevoir leur environnement.
- Les semi-entités sociales qui peuvent former des sociétés, avoir des relations sociales et posséder des propriétés sociales propres.
- Enfin, les semi-entités mentales qui représentent des éléments pouvant être caractérisés en termes d'états mentaux : croyances, objectifs, besoins, désirs, buts, etc.

Ces semi-entités peuvent ensuite être surchargées par héritage pour intégrer les besoins d'approches plus spécifiques. Les entités de base d'un SMA telles les agents, les ressources et l'environnement disposent également de leurs définitions dans ce langage.

AORML (Agent-Object Relationship Modeling Language) [Wag03] est un profil UML pour l'analyse et la conception de systèmes d'information organisationnels basés sur les agents. D'après l'auteur de [Gau07], dans cette approche, deux points de vue différents (externe et interne) sont adoptés sur la notion d'agent. Le point de vue externe décrit les agents, leurs interactions dans le domaine de l'application, leurs croyances sur les objets qu'ils manipulent ainsi que les relations qui relient ces différents éléments. Ces différents points sont représentés par les diagrammes d'Agent ("Agent Diagram"), de fenêtre d'interaction ("Interaction Frame Diagram"), de séquence d'interaction ("Interaction Sequence Diagram") et de schéma d'interaction ("Interaction Pattern Diagram"). Le point de vue interne décrit quant à lui les différentes composantes internes d'un agent. Ce point de vue est associé aux diagrammes de fenêtre de réaction ("Reaction Frame Diagram"), de séquence de réaction ("Reaction Sequence Diagram") et de schéma de réaction ("Reaction Pattern Diagram").

1.2. Agent UML

1.2.1. Rappel

UML a unifié et formalisé les méthodes de plusieurs approches, incluant les méthodes de Booch, Rumbaugh, Jacobson, et Odell. Bauer a noté dans [Bau01b] qu'UML souffre d'insuffisance pour la modélisation des systèmes à base d'agents [Bau00]. Principalement, cela est dû à deux raisons majeurs : premièrement, comparé aux objets, les agents sont actifs parce qu'ils peuvent prendre l'initiative et ont un contrôle globale sur leurs comportements. Deuxièmement, les agents n'interagissent pas de façon individuel mais plutôt en coopération ou en coordination avec d'autres agents. Les SMA sont des communautés sociales de membres interdépendants qui agissent individuellement. De ce fait, plusieurs langages de modélisation des systèmes agents sont proposés, parmi eux Agent UML [Hah08].

AUML (Agent UML) est un langage offrant de nombreux artefacts en vue de la construction d'un système logiciel. Avec ses vues et ses divers types de diagrammes, il offre la possibilité de modéliser la structure statique d'un système, ses besoins fonctionnels, en plus de son comportement collectif et individuel [Gag07]. AUML a été proposé comme une extension du langage UML. Jusqu'à maintenant il n'y a pas de standard reconnu pour la modélisation des SMA et AUML a émergé comme un candidat pour assumer une telle position. Il utilise les caractéristiques de "décomposition", "d'abstraction", et "d'organisation", qui réduisent la complexité de développement de logiciel. AUML décompose le système en petites parties d'objets, de modèles, de cas d'utilisation ou de classes, et d'actions opérationnelles. Concernant "l'abstraction", elle offre une vue spécialisée abstraite de la modélisation (classe, cas d'utilisation, diagramme, interface, etc.).

"L'organisation" orientée agent définit un ensemble d'éléments et de notations comme la spécification des besoins pour la modélisation du domaine. Son objectif est de fournir un modèle et une architecture interne d'un système agent. Elle offre généralement quelques Framework (classe, diagramme, interface, etc.) pour voir comment les agents peuvent être construits dans un système. La partie principale d'AUML est les mécanismes de modélisation des protocoles d'interaction des SMA. Cela est réalisé par l'introduction d'une nouvelle classe de diagramme à UML : "diagramme de Protocole". Ces diagrammes étendent les diagrammes de séquence en incluant : 'les rôles d'agents', 'multithreaded lifelines', 'protocole templates', etc. [Gag07].

Dans le volet des insuffisances, AUML tout comme UML [Ake02] offre la possibilité de décrire plusieurs "vues" d'un même système. Cependant, ces différentes vues peuvent être en conflit puisqu'elles peuvent présenter des incohérences. De plus, l'expérience montre que de très nombreuses fautes de modélisation sont perceptibles à travers la détection d'incohérences [Mus03, Reg99, Mor00, Ast98]. Selon [Mal05], une incohérence est la violation d'une propriété associée au langage AUML qui doit être respectée par tout modèle AUML. En effet, la cohérence peut concerner un seul diagramme (on parle de cohérence intra-diagramme) ou plusieurs diagrammes (on parle de cohérence inter-diagrammes).

Parmi les incohérences intra-diagramme, on peut mentionner la présence de cycles dans un graphe d'héritage pour un diagramme de classe, l'existence d'une transition dont l'état source est un état final dans un diagramme d'états-transitions et le non respect de l'ordre d'envoi de message d'un point de synchronisation dans un diagramme de communication (par exemple, le numéro du message bloqué est inférieur au numéro du message bloquant). La représentation des systèmes complexes, en utilisant les différents diagrammes cités précédemment, peut engendrer des incohérences inter-diagrammes. On peut citer, entre autres, un événement appel d'un diagramme d'états-transitions qui ne figure pas comme méthode dans la classe correspondante. Aussi, un envoi de message (appel de procédure) dans un diagramme de communication ne figurant pas comme une méthode à visibilité appropriée dans la classe de l'objet destinataire. Par ailleurs, l'état final d'un système fini ne se compose pas de différents états finaux des différents objets impliqués dans le diagramme de communication [Gag07].

Dans la section suivante nous allons présenter quatre des principaux diagrammes d'AUML, ainsi que les travaux qui les considèrent.

1.2.2. Les Principales Représentations d'Agent UML

Comme nous l'avons indiqué dans l'introduction, Agent UML est une extension d'UML afin de prendre en compte les spécificités des systèmes orientés agent. Il hérite des représentations proposées par UML telles que les Diagrammes: "de séquence ; de collaboration ; d'activité ; d'état transition ; de cas d'utilisation ; de classe ; d'objets ; Packages ; de composants ; et de déploiement". Les cinq premières représentations correspondent à des diagrammes dynamiques alors que les quatre dernières correspondent à des diagrammes statiques. Nous rappelons que notre but est de déterminer quelles représentations ont besoin d'être modifiées ou améliorées pour prendre en charge les spécificités des systèmes multi-agent temps réel. Les diagrammes de séquence ont déjà été modifiés puisqu'ils portent maintenant le nom de diagrammes de protocole [Ode00, Bau00]. Ces diagrammes de protocole ont été eux aussi améliorés par Huget [Hug02b], et correspondent à la représentation des protocoles d'interaction. Les diagrammes de classe ont aussi été modifiés. Il ne faut pas oublier qu'un agent diffère d'un objet par conséquent le diagramme de classe qui permet de représenter les classes constituant le système sera aussi modifié.

Nous trouvons deux approches pour ces diagrammes de classes: celle de Bauer [Bau01b] et l'autre de Huget [Hug02c]. La première approche [Bau01b], propose d'interpréter l'acceptation des actes de communication dans les diagrammes de classe agent. Alors que, pour Huget [Hug02c], cette approche est irréaliste dès que les agents englobent plusieurs protocoles complexes. De ce fait, nous nous sommes basés sur la deuxième proposition de Huget [Hug02c] jugée intéressante et qui sera décrite ci-dessous.

Dans les paragraphes qui suivent nous rappelons brièvement à quoi servent ces représentations (diagrammes de cas d'utilisation, de classe, de protocole, et d'état transition) et nous présentons les principaux travaux qui les considèrent.

1.2.2.1. Les Diagrammes de Cas D'utilisation AUML

Les diagrammes de cas d'utilisation sont utilisés pour définir des cas d'utilisation du système et donnent une analyse du système. Ils représentent les cas d'utilisation, les acteurs, et les relations qui existent entre les acteurs et les cas d'utilisation. Un cas d'utilisation peut être vu comme un scénario dans le système, par exemple quand un utilisateur essaye de se connecter au système et un mot de passe est nécessaire. Dans les SMA, les cas d'utilisation sont intéressants lorsqu'il s'agit de réaliser l'analyse des besoins (requirement analysis). Ils sont utiles durant le dialogue entre les utilisateurs finaux et les concepteurs puisqu'ils sont graphiques, et permettent aux utilisateurs de saisir les différents éléments du système. L'avantage principal des cas d'utilisation est de se concentrer sur le quoi et non sur le comment, c.à.d. sur le comportement du système et non sur comment le système est implémenté.

La Figure 3.1 présente le diagramme de cas d'utilisation d'un agent client situé au sein d'une entreprise de production. Nous avons un seul acteur sur cette figure : l'agent client. Les cas d'utilisation sont annotés par un cercle : placer ordre, modifier ordre, et annuler ordre. Les lignes entre l'acteur et les cas d'utilisation dénotent que l'acteur utilise ces scénarios.

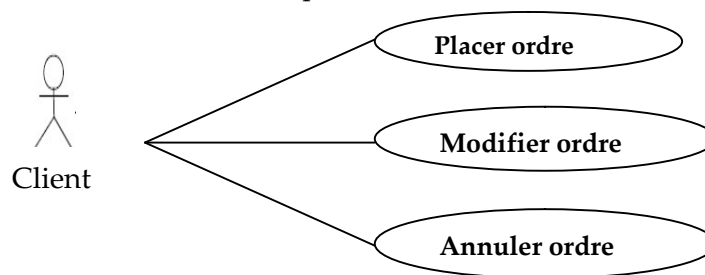


Figure 3.1. Diagramme de cas d'utilisation d'un agent client.

Peu de travaux considèrent les diagrammes de cas d'utilisation pour la modélisation des SMA, parmi eux nous trouvons les propositions de Michael Papasimeon et Clinton Heinze [Pap00], Clinton Heinze, Michael Papasimeon, et Simon Goss [Hei00], Bernhard Bauer et James Odell [Bau05]. Et beaucoup plus pour la description des systèmes multi-agent temps réel à l'instar du travail présenté dans [Lao10]. Un besoin qui nous a motivé à exploiter cette voie de recherche, pour en proposer une solution sous forme d'extensions à ce type de diagramme.

Particulièrement employé au cours des phases amont du processus de développement, ce type de modèle est un outil de base pour la formalisation de besoins fonctionnels et une aide précieuse dans le dialogue avec les utilisateurs.

1.2.2.2. Les Diagrammes de Classe AUML Etendus

Les diagrammes de classes dans UML permettent de représenter les différentes classes utilisées dans le système. Ils donnent la structure du système et les relations qui unissent les classes. Ce type de diagrammes semble être intéressant pour représenter les différents rôles d'agents qui se trouvent dans le système et les relations entre ces rôles. Mais vu que ces diagrammes ne considèrent que les attributs et les opérations pour les objets, ceci se révèle être insuffisant pour représenter les agents, car ces derniers proposent des services, ont des capacités, utilisent des protocoles d'interaction, ont plusieurs rôles, etc. En conséquence à cela, une première proposition d'extension des diagrammes de classes a été faite dans [Bau01b], suivie d'une autre suggestion, apparue dans [Hug02c], où l'auteur considère deux niveaux d'abstraction. "Un niveau conceptuel" et "un niveau d'implémentation", le premier niveau ne fait pas intervenir les classes

reliées aux agents mais uniquement ceux-ci, alors que le second niveau donne une vue détaillée des agents, des classes et des différentes relations.

Le diagramme de classe proposé par Hugot [Hug02c], est constitué d'un ensemble de compartiments. "*Le premier compartiment*" donne le nom de la classe de l'agent. Ce nom est préfixé par le stéréotype « agent » afin de permettre de différencier un agent d'un objet, cependant le nom de l'agent est insuffisant pour représenter son rôle dans le SMA, d'où l'existence du "*deuxième compartiment*" qui donne l'ensemble des rôles joués par cette classe d'agent. Cette information est ensuite utilisée pour les protocoles d'interaction et les organisations. "*Le compartiment des statecharts*" correspond au comportement de la classe. Il ne s'agit pas d'insérer le diagramme d'états au sein du diagramme de l'agent mais de donner son nom.

Le compartiment suivant appelé "*attributs*" est équivalent à celui présent dans les diagrammes de classes UML. Il donne l'ensemble des attributs de la classe d'agent. A chaque fois, le type des attributs est précisé. Les points de suspension à la fin du compartiment indiquent que tous les attributs ne sont pas présents sur le diagramme.

Le compartiment qui est au dessous du précédent est celui "*des opérations*" qui est aussi présent dans les diagrammes de classes d'UML. Mais ces opérations correspondent aux actions exécutées par les agents. Elles sont divisées en trois catégories : "*internes*", "*pro-actives*" et "*réactives*". Les opérations internes ne sont pas visibles par les autres agents, par exemple si les agents ont un module d'interaction spécifique pour la communication [Hug02d], de telles opérations doivent être utilisées pour récupérer les messages. Les opérations pro-actives correspondent aux opérations déclenchées pour définir des conditions comme un timer, des exceptions, ou des conditions. Un exemple courant est celui d'un serveur qui informe les agents de valeurs spécifiques chaque t unités de temps. Les opérations réactives sont des actions liées aux modifications faites sur l'environnement ou en réaction aux actions des autres agents.

Il y a des opérations qui ont des paramètres et d'autres ont des pré-conditions qui doivent être évaluées à vrai pour que l'opération soit effectuée. Le format des conditions est libre : il peut s'agir d'un texte ou d'une condition au format "OCL". Une fois de plus, il y a des points de suspension puisqu'il n'est pas précisé des opérations portant sur la gestion des messages par exemple. Ces trois types d'opérations sont distingués dans les diagrammes de classe par les mots clé : 'int' pour opérations interne, 'pro' pour opérations pro-active et 'reac' pour opérations réactives.

Le compartiment qui vient après donne "*les protocoles d'interactions*" utilisés par la classe d'agent tout en précisant les rôles joués par celle-ci dans ces protocoles. Une fois de plus, il ne s'agit pas d'insérer le diagramme de protocoles au sein du diagramme de classe mais de donner le nom de celui-ci.

Les deux derniers compartiments concernent la "*vue organisationnelle*" des agents. En effet plusieurs informations sont fournies selon cette vue : 'les capacités', 'les services' et 'les organisations' dans lesquelles l'agent est impliqué.

Les capacités décrivent ce qui est l'agent capable de faire. Ces capacités sont écrites sous forme textuelle et sont stockées dans "*le compartiment Capacité*". Les capacités sont dérivées comme des services aux agents. Ces services ne sont pas représentés dans le diagramme de classe agent mais interconnecté. La dernière information dans cette vue est la liste des organisations dans lesquelles l'agent appartient. Cette liste est placée dans "*le compartiment organisation*". Les agents jouent des rôles dans les organisations, de ce fait ces rôles seront insérés pour chaque organisation. Cependant, L'entrée et la sortie des organisations sont contraintes par des conditions.

1.2.2.3. Les Diagrammes de Protocole AUML Etendus

Nous avons vu que les interactions entre les agents sont le concept primordial permettant aux systèmes multi-agents de fonctionner. La notation AUML est fortement centrée sur les protocoles d'interactions entre agents. Les diagrammes de protocole AUML sont très similaires aux diagrammes de séquence dans UML, mais avec l'addition de nouveaux artefacts graphiques avec leurs sémantiques associées. Ces opérateurs sont principalement AND (qui représente le parallélisme), OR (choix non déterministe entre une ou plusieurs alternatives) et XOR (un seul message parmi une liste à envoyer). Ils ont été ajoutés pour essayer de modéliser et d'exprimer d'une façon plus complète les interactions entre les agents (Figure 3.2).

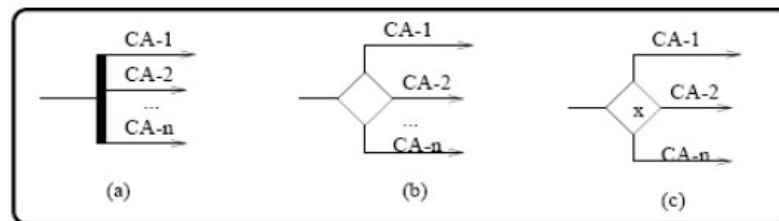


Figure. 3.2 Les branchements définis dans AUML.

Les diagrammes de protocole sont les plus communs pour décrire les protocoles d'interaction utilisés par les agents. Ils exploitent la richesse des communications entre agents et représentent les échanges de messages entre ces derniers. Ces agents vont être représentés par leurs 'rôles' dans l'interaction. En effet, les messages ne sont pas échangés d'un agent à un autre mais plutôt d'un rôle à un autre. L'avantage de cette approche est de réduire la taille du diagramme. Le concepteur ne représente pas le message envoyé par le client Smith à la compagnie ACME mais du rôle client au rôle vendeur [Hug02d]. Ces diagrammes sont certainement les plus achevés dans Agent UML puisque plusieurs travaux les considèrent [Ode00, Bau00], pour n'en citer que quelques uns. On peut également citer le travail de Huget [Hug02b] où il propose des solutions pour qu'Agent UML puisse supporter d'autres propriétés tel que : 'le broadcasting', 'la synchronisation', 'le traitement des exceptions', 'la gestion du temps', etc.

1.2.2.4. Les Diagrammes d'Etat Transition AUML

Les diagrammes d'états-transitions d'UML décrivent le comportement interne d'un objet à l'aide d'un automate à états finis. Ils présentent les séquences possibles d'états et d'actions qu'une instance de classe peut traiter au cours de son cycle de vie en réaction à des événements discrets.

Le diagramme d'états-transitions est le seul diagramme, de la norme UML, à offrir une vision complète et non ambiguë de l'ensemble des comportements de l'élément auquel il est attaché. En effet, un diagramme d'interaction n'offre qu'une vue partielle correspondant à un scénario sans spécifier comment les différents scénarii interagissent entre eux. La vision globale du système n'apparaît pas sur ce type de diagramme puisqu'ils ne s'intéressent qu'à un seul élément du système indépendamment de son environnement.

Bien entendu, le modèle dynamique du système comprend plusieurs diagrammes d'états-transitions. Il est souhaitable de construire un diagramme d'états-transitions pour chaque classeur (qui, le plus souvent, est une classe) possédant un comportement dynamique important. Un diagramme d'états transitions ne peut être associé qu'à un seul classeur. Tous les automates à états finis des diagrammes d'états-transitions d'un système s'exécutent concurremment et peuvent donc changer d'état de façon indépendante [Aud06].

Selon [Mul00], les diagrammes d'états-transitions sont en réalité des graphes orientés. Leur rôle principal est de décrire le cycle de vie des objets, les événements qu'ils subissent, leurs transitions et leurs états. Un diagramme d'états-transitions est composé de plusieurs éléments : les états (ou noeuds du graphe), ainsi que les événements et les transitions (les arcs du graphe).

Un diagramme d'états-transitions est habituellement constitué d'un état initial, d'états intermédiaires (zéro ou plusieurs) et d'un ou plusieurs états finaux (zéro ou plusieurs). Le second concept d'importance face aux diagrammes d'états est la transition. Une transition est le lien qui existe entre deux différents états. C'est par ce lien que l'objet peut changer d'état. Lorsqu'un événement se produit, il peut entraîner un changement d'état face à un objet, mais conditionnellement à ce que l'événement en question ne soit associé à une transition. Notons qu'il est impossible de transiter entre un état donné et un autre sans qu'une transition existe entre les deux. Avec UML, une transition est représentée par une flèche entre deux états. Par ailleurs, une transition peut être gardée. Les gardes sont évaluées dès l'occurrence de l'événement de déclenchement. Elles permettent de maintenir l'aspect déterministe d'un automate d'états finis, même lorsque le même événement est susceptible de déclencher plusieurs transitions. Les événements, quant à eux, pourraient être définis par quelque chose de remarquable par un objet.

Cet événement se produit à un instant quelconque lors de l'exécution du système. La Figure 3.3 montre un exemple d'un événement déclenchant une transition. Cet événement peut provenir du système ou de son interface, et peut être de trois natures différentes. Un événement dit *externe*, ou événement système, est un événement causé par un des acteurs. Un événement dit *interne* est un événement causé par une entité du système même. On pourrait citer comme exemple l'invocation d'une méthode de l'objet par l'envoi d'un message par un autre objet. Enfin, *un événement temporel* est lié à un temps écoulé ou à une date et une heure spécifique. Par exemple, on pourrait faire déclencher une méthode quelconque après qu'un laps de temps déterminé se soit écoulé [Gag07].



Figure. 3.3. Exemple d'un événement déclenchant une transition [Gag07].

Après avoir étudié les langages de modélisation, nous passons aux langages de spécification formelle où nous présentons plus particulièrement le langage Maude et son extension RT-Maude pour la description des systèmes temps réel sans oublier de parler sur la technique de vérification de modèle (modèle checker).

2. Langages de Spécification Formelle

Selon l'auteur de [Alk04], une technique est qualifiée de formelle lorsque les différentes opérations qu'elle supporte, les objets manipulés par ces opérations ainsi que le système de preuve qu'elle possède sont exprimés dans un langage à syntaxe et à sémantique formelles, à base mathématiques. En conséquence, les techniques formelles relèvent de la sémantique, c'est-à-dire de la représentation mathématique que l'on peut associer à un programme. Elles permettent de nombreux développements de programmes et de nombreuses vérifications de propriétés. Par contre, des efforts doivent être faits pour les rendre utilisables, à grande échelle, dans la pratique. Dans cette section, nous tentons d'exposer les techniques les plus pertinentes pour spécifier formellement un système logiciel, en particulier le langage Maude et son extension Real Time Maude pour les systèmes temps réel.

2.1. Approches Formelles

Azaiez dans son travail [Aza07] affirme que les notations formelles se basent sur les mathématiques pour éliminer les ambiguïtés des représentations informelles ou semi-formelles et les mauvaises interprétations qui en découlent. Les langages formels se caractérisent par une syntaxe qui définit la notation utilisée ; une sémantique permettant de lever l'ambiguïté de cette syntaxe, et d'un ensemble de relations explicitant les règles donnant les propriétés des objets mathématiques satisfaisant la spécification. Dans ce sens, Hernandez [Her04] est convaincu du grand intérêt des méthodes de spécification formelle pour réduire la nécessité de tout essayer de prouver "à la main", et de remédier aux problèmes classiques de manque de compréhension des spécifications par le biais d'un langage non ambigu pour faire le modèle.

Évidemment les détails sur la manière dont un processus formel de spécifications se déroule changent d'une technique à l'autre, mais la plupart des méthodes formelles utilisent des machines d'état fini pour le représenter. La plupart des vérificateurs ont généralement un comportement itératif, où l'outil qui valide prend des spécifications d'entrée et détermine si quelques propriétés appropriées sont satisfaites par elles, comme la sécurité et la vivacité. Au cas où il y aurait des propriétés qui sont non-satisfaites, les spécifications d'entrée doivent être modifiées jusqu'à ce qu'elles le deviennent [Her04].

Étant donné que le domaine des méthodes formelles est plutôt large, on va parler ici de quelques exemples de langages formels qui ont été utilisés avec succès pour modéliser et valider des systèmes logiciel, comme le langage Z et Object-Z [Spi92], le langage B [Can06], Promela/SPIN [Ger97], les réseaux de Petri [Car62, Car66] et Maude [Cla05]. De plus, certaines descriptions non formelles peuvent être étendues par des langages formels pour l'expression des contraintes, comme c'est le cas d'OCL [Bot00] dans le cas d'UML.

Z et Object-Z sont probablement parmi les approches les plus connues. D'après Amiguet [Ami00] Z est un langage de spécification formelle orienté modèle, ce qui veut dire que sa sémantique est composée de la définition d'un modèle en termes d'ensembles, de relations et de fonctions. Il ajoute qu'une notation logique peut être utilisée pour exprimer des contraintes sur ces notions ensemblistes. Cependant, pour la description des grands projets le langage Z lui manquait des moyens de structuration. De ce fait, le langage Object Z est apparu pour rajouter à Z les notions classiques de classe, instanciation, héritage, etc. Il offre aussi comme mentionné dans [Gag07] un vérificateur de type (Type Checker). Cet outil, expliqué en détail dans [Joh96], s'appelle Wizard et permet la vérification de la conformité des types des paramètres ou valeurs de retour des fonctions des classes.

Gagnon dans [Gag07] présente OCL (Object Constraint Language) [War99] comme étant un langage de spécification formelle qui permet d'associer une ou plusieurs contraintes aux éléments d'un modèle orienté-objet comme c'est le cas d'UML. Du fait qu'il est de type textuel, OCL se distingue vis à vis des autres langages tel que Z qui sont basés sur la syntaxe symbolique des mathématiques. Dans le contexte de l'utilisation d'UML, une contrainte est une condition qui est imposée à un ou plusieurs éléments d'un modèle orienté objet. Plus précisément, OCL permet de formuler des conditions, sous forme d'expressions booléennes, qui peuvent être vraies ou fausses. La spécification d'OCL n'impose pas de règle stricte sur l'endroit où peut être placé les contraintes (sous la forme d'invariants, préconditions et postconditions) mais il est préférable de les mettre sur les modèles UML.

La méthode B [Can06], développée par Abrial, est conçue pour la spécification, le design et le codage de systèmes logiciels. Tout comme Object-Z, B est basée sur des notations purement mathématiques. Ceci, une fois de plus, permet d'obtenir des spécifications du système en développement sans ambiguïté. La méthode B a aussi l'avantage d'être très répandue en Europe, et a fait l'objet de nombreux travaux. Elle est aujourd'hui supportée par plusieurs outils, notamment Atelier B. Z et B partagent de nombreuses racines communes, tout particulièrement en ce qui concerne les concepts mathématiques utilisés [Gag07].

En plus des trois langages décrits ci-dessus, nous en parlant aussi des deux méthodes formelles Promela/Spin et les réseaux de petri tels que définit dans la thèse de Hernandez [Her04]. Promela (Process Meta Language) est le langage accepté par l'outil de validation formelle SPIN [Ger97]. Un des aspects les plus intéressants de Promela/SPIN est l'approche "ingénieur" c'est-à-dire le langage donne des constructions qui sont familières à ceux qui connaissent le langage C, sans pour autant perdre de vue qu'il s'agit d'un langage de spécification formelle et pas un langage de programmation. En Promela il existe la notion de variable, qui a une signification équivalente à celle de C (une variable est une valeur symbolique qui existe dans un certain contexte, qui contient une valeur et qui peut être manipulé avec les opérations appropriées). De son côté SPIN permet de modéliser et de valider des systèmes distribués et parallèles en utilisant une approche appelé "on the fly", qui consiste principalement à générer un évaluateur automatisé. Ce dernier une fois compilé et exécuté, produit une représentation très compacte du système basée sur des automates de Boochi, sur laquelle il est possible d'effectuer un nombre de preuves classiques dans le domaine du model-checking mais aussi de vérifier la validité de formules logiques, exprimées par le développeur en logique temporelle linéaire. Ce qui rend le model compact créé par SPIN différent, et qu'il est créé "sur le champ" (dont vient le nom "on the fly"), à la place de créer tout l'espace d'états qui peut être potentiellement gigantesque dans le modèle de taille conséquente. Ceci a comme résultat de manipuler des modèles plus grands et plus complexes avec moins d'espace mémoire et de temps d'exécution.

Les réseaux de Petri sont une notation formelle et visuelle utilisée pour représenter des systèmes discrets concurrents. Ils ont été proposés originalement par Carl Adam Petri dans [Car62] et présentés en détail dans sa thèse doctorale [Car66]. Depuis ils ont été adaptés, étendus et modifiés pour modéliser des systèmes à caractéristiques particulières qui ne pouvaient pas être représentés de façon satisfaisante avec le modèle originale. Les réseaux de Petri ont plusieurs avantages pour la représentation des systèmes concurrents. Pour commencer, ils peuvent représenter le parallélisme de façon beaucoup plus compacte que les automates à état fini. Ils sont en réalité une généralisation des automates, avec le même pouvoir expressif. Il y a cependant des extensions de la notation originale qui sont encore plus puissantes, et qui sont en fait des constructions formelles Turing complètes (c'est à dire, elles peuvent représenter n'importe quel algorithme représentable dans une machine de Turing). Un autre avantage significatif des réseaux de Petri, est le fait qu'ils disposent d'une notation visuelle qui est assez intuitive pour les modèles de taille modérée en termes de nombre d'états et de transitions, ce qui les rend attrayants pour ceux qui s'intéressent à la modélisation visuelle.

Dans sa thèse, Sadoun [Sad14] décrit Maude [Cla02, Cla11] comme un langage de spécification et de programmation basé sur la logique de réécriture [Mes90] qui intègre un ensemble d'outils de validation et de vérification [Cla07b] dont un modèle checker [Eke04]. Martí-Oliet et Meseguer [Mar02] ont montré que Maude représente un très bon cadre logique et sémantique dans lequel différentes logiques et différents formalismes peuvent être exprimés et exécutés. De plus, il est

particulièrement bien adapté pour la spécification de systèmes concurrents orientés objet [Rom07] et permet de décrire l'état d'un système et son comportement de façon mathématique et rigoureuse. En outre, divers travaux [Rom07, Riv09, Rus13] ont montré que Maude offre un moyen simple, naturel et précis de spécifier les modèles et les méta-modèles. Au fil des années, Maude s'est avéré être un bon environnement pour le prototypage rapide [Mes10] et l'aide au développement et à la validation de modèles UML/OCL [Cla06b]. De nombreux travaux ont également exploré son utilisation pour la spécification de modèles décrits par des langages du web sémantique [Ver05, Sen05, Ele08, Hua09, Son12, Gue12] tels que RDF ou OWL.

Dans la section suivante, nous allons détailler le langage Maude et son extension pour le temps réel Real Time Maude, vu leurs importances dans notre travail.

2.2. Maude

Le système Maude est un environnement de programmation basé sur la logique de réécriture, celle-ci a été introduite dans le but de modéliser des systèmes informatiques, et plus particulièrement les systèmes concurrentiels. Maude étant un méta-langage, utilisé pour développer d'autres langages. En effet Real-Time Maude (RT-Maude) est implémenté en Maude comme une extension de Full Maude. RT-Maude est un langage et un outil qui supporte la spécification et l'analyse des systèmes temps réel, et plus particulièrement les systèmes temps réel orientés-objet. Il constitue un candidat parfait pour la translation des diagrammes AUML vers une notation formelle.

Dans les deux sections qui suivent, la logique de réécriture et le langage Maude feront l'objet d'une étude un peu plus détaillée. Par la suite, le langage Real Time Maude proprement dit sera discuté. L'ensemble des exemples Maude présents ci-dessous ont été écrits dans le plug-in "Maude Development Tools" qu'on a intégré dans la plateforme "Java Eclipse".

2.2.1. Fondement de Maude : Logique de Réécriture

La logique en général est une méthode de raisonnement correct pour quelques classes d'entités. Pour la logique équationnelle, les entités sont les ensembles, les fonctions entre ces ensembles et la relation d'identité entre les éléments. Pour la logique de réécriture, les entités sont les systèmes concurrents ayant des états, et qui évoluent à travers des transitions. Elle est considérée également comme une logique de changement concurrent qui traite l'état et les calculs concurrents [Meg04].

Dans ce qui suit, une brève introduction à la logique de réécriture [Maa04] sera donnée afin de mieux saisir son fonctionnement lorsque viendra le temps de la mettre en application. La logique de réécriture a été proposée par J.Messeguer comme une sémantique concurrente des systèmes de réécritures de termes où les opérateurs peuvent être associatifs, commutatifs, idempotents ou dotés d'un élément neutre à gauche ou à droite. La logique de réécriture formalise le processus de la réécriture, par opposition à la logique équationnelle qui considère les règles de réécritures comme des égalités et utilise la réécriture pour calculer un représentant unique d'une classe d'équivalence. La logique de réécriture utilise les règles de réécriture pour calculer une relation de "réécrivabilité" entre des termes. Elle permet d'exprimer le fait qu'un terme peut être réécrit en un autre.

Cette logique exprime aussi une équivalence essentielle entre logique et calcul dans une représentation simple. Un axiome logique de réécriture de la forme, $(t \rightarrow t')$, a deux lectures.

Dans l'esprit calculatoire, un fragment t d'états de système, peut se transformer en t' , en concurrence avec d'autres changements d'états, c'est une transition local concurrente. Dans l'esprit logique, on peut dériver la forme t' à partir de t , on l'appelle une règle d'inférence.

La logique de réécriture, utilisée comme une structure sémantique, supporte un large spectre d'applications. Premièrement ; elle est utilisée pour spécifier et unifier beaucoup de modèles de calculs concurrent. Deuxièmement ; la logique de réécriture est aussi une structure sémantique pour les langages de programmation. Elle peut être vue comme un langage déclaratif de spécification où les programmes parallèles ou séquentiels peuvent être spécifiés par des règles de réécriture.

Les éléments définis ci-dessous sont tirés d'un texte de Meseguer [Mes02]. La première définition introduit le concept de la théorie de réécriture :

Une théorie de réécriture est un triple $R = (\Sigma; E; R)$ où :

- $(\Sigma; E)$ est une théorie d'équations ensemblistes ;
- R est un ensemble de règles de réécritures conditionnelles nommées ayant la forme suivante :

$$r : t \rightarrow t' \Leftarrow \left(\bigwedge_i u_i = u'_i \right) \wedge \left(\bigwedge_j v_j : s_j \right) \wedge \left(\bigwedge_l w_l \rightarrow w'_l \right)$$

pour lequel r est un identificateur pour la règle, et où il existe un ensemble fini X de variables pour lequel t, t' appartient à $T_\Sigma(X)_k$ pour un genre (kind) k , ainsi que tous les Σ termes $u_i; u'_i; v_j; w_l; w'_l$ dans la condition sont également dans l'ensemble $T_\Sigma(X)_k$. La condition est la conjonction d'équations, d'équations ensemblistes et de réécritures.

Lorsqu'une règle de réécriture ne comporte aucune condition, cette règle est alors appelée inconditionnelle. Ainsi, on peut introduire la définition d'une théorie de réécriture générale comme suit : une théorie de réécriture générale est un :

quadruplet $\mathcal{R} = (\Sigma, E, \phi, R)$ où :

- (Σ, E) est une théorie d'équations ensemblistes avec, disons, les genres K , les sortes S et les opérations Σ ;
- $\phi : \Sigma \rightarrow \mathcal{P}_{fin}(\mathbb{N})$ est une famille indexée de fonctions de $K_* \times K$ qui assigne à chaque $f : k_1 \dots k_n \rightarrow k$ dans E l'ensemble fini $\phi(f) \subseteq \{1, \dots, n\}$ de positions d'arguments figés ;
- R est l'ensemble des règles conditionnelles nommées de réécritures (quantifiées universellement) ayant la forme (avec les paires t, t' et w_l, w'_l ayant le même genre) :

$$r : t \rightarrow t' \Leftarrow \left(\bigwedge_i u_i = u'_i \right) \wedge \left(\bigwedge_j v_j : s_j \right) \wedge \left(\bigwedge_l w_l \rightarrow w'_l \right)$$

De façon similaire à une 'théorie d'équations ensemblistes', 'une théorie de réécriture' possède une série de règles d'inférence. Elles sont la Réflexivité, l'Égalité, la Congruence, le Remplacement et la Transitivité. La Figure 3.4 permet de visualiser chacune de ces règles.

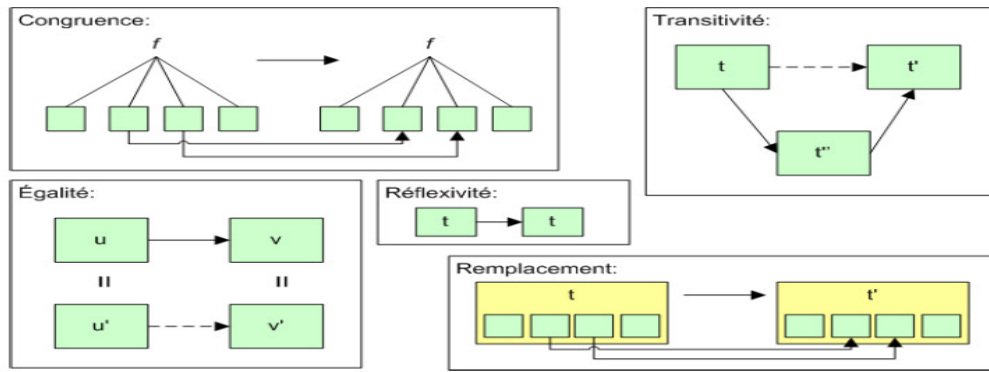


Figure 3.4. Visualisation des règles d'inférence d'une théorie de réécriture [Gag07].

La logique de réécriture supporte un large spectre d'applications. Premièrement ; elle est utilisée pour spécifier et unifier beaucoup de modèles de calculs concurrents. Deuxièmement ; la logique de réécriture est aussi une structure sémantique pour les langages de programmation où les programmes parallèles ou séquentiels peuvent être spécifiés par des règles de réécriture [Gar02]; de nombreux langages de programmation ont été conçus pour exploiter les concepts de la réécriture, Maude [Ekr02] est l'un d'eux.

Les principales notions mathématiques pertinentes à la logique de réécriture ayant été discutées brièvement depuis le travail de Gagnon [Gag07], maintenant l'environnement Maude peut être introduit.

2.2.2. Structure de Maude

L'unité de base de spécification et de programmation dans Maude est le module, en effet trois types de modules sont définis dans le langage Maude. Les modules fonctionnels (**fmod**) permettent de définir les sortes de données et les opérations sur ces données à travers la théorie des équations. Les modules systèmes (**mod**) définissent le comportement dynamique d'un système. Ce type de module étend les modules fonctionnels en introduisant les règles de réécriture, qui permettent d'offrir un degré maximal de concurrence. Finalement, il y a les modules orientés-objets (**omod**), par rapport aux modules systèmes, ils offrent une syntaxe plus appropriée pour décrire les entités de base du paradigme objet comme : classe, objets, messages. Ces modules doivent être déclarés respectivement en respectant les syntaxes suivantes:



Les points représentent les déclarations et les expressions qui peuvent apparaître dans le module.

Il existe actuellement deux niveaux séparés de Maude : *Core Maude* et *Full Maude*, tel que n'importe quelle spécification dans *Core Maude* est également dans *Full Maude*, mais non pas l'inverse. Les Modules systèmes et les modules fonctionnels sont les deux définis dans *Core Maude*, mais non pas les modules orientés-objet. Pour utiliser *Full Maude*, Tous les commandes et les modules doivent être déclarés entre parenthèses, donc la déclaration d'un module orienté-objet sera comme suit:



2.2.2.1. Caractéristiques de Maude

- Basé sur la logique de réécriture : Où les programmes sont des théories, et les règles de déduction logique de réécriture correspond exactement au calcul concurrent.
- Spectre large (wide spectrume) : La spécification est rapide et efficace pour l'exécution parallèle et distribuée, aussi bien que la transformation formelle des spécifications des programmes.
- Multi-paradigmes : Toutes les applications basées sur la logique équationnelle sont supportées facilement, donc la programmation fonctionnelle style équationnelle est naturellement représentée par un sous-langage. Le style déclaratif de programmation concurrente, orientée-objets est aussi supporté avec une sémantique logique simple.
- Stratégies internes : Cette propriété permet de guider le processus de réécriture dont la sémantique peut être définie à l'intérieur de la logique par des règles de réécriture.

Dans ce qui suit nous introduisons d'abord les éléments communs aux modules fonctionnels, systèmes et aux modules orientés-objet : sortes, sous-sortes, opérations et variables. Nous présenterons ensuite la définition de la syntaxe et de la sémantique spécifique à chacun des types de modules. Toutes les notions citées dans cette section sont rédigées à partir du document [Cla05].

2.2.2.2. Concepts de base

Sortes.

La première chose à déclarer dans une spécification est les *sortes* de données et les opérations correspondantes. Les sortes sont partiellement ordonnées via une relation de sous-sort. Une sorte est déclarée en utilisant le mot-clé **sort** suivi d'un identificateur (le nom de sorte) suivi d'un espace et un point : **sort** <Sort> .

Les sortes multiples seront déclarées en utilisant le mot-clé **sorts** : **sorts** <Sort1> . . . <Sortn> .

Sous-sortes

La relation de sous-sort **subsort** sur les sortes est équivalente à une relation de sous-ensemble sur un ensemble d'éléments (relation d'inclusion). Les sous-sortes sont déclarées en utilisant le mot-clé **subsort** ou **subsorts**. La déclaration : **subsort** <Sort1> <Sort2> .

Signifie que la sorte <Sort1> est une sous-sort de <Sort2>. Par exemple, les déclarations :

subsort Zero < Nat . **subsort** NzNat < Nat .

Signifient que les sortes *Zero* (contenant seulement la constante 0) et *NzNat* (les nombres naturels non nuls) sont des sous-sortes de *Nat* (les nombres naturels). On peut déclarer plusieurs relations de sous-sort en utilisant **subsorts** : **subsorts** <Sort1> . . . <Sort2> < . . . <Sortn> .

Par conséquent les déclarations de l'exemple précédent peuvent être faites en une seule ligne :

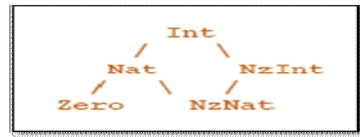
subsorts Zero NzNat < Nat .

On peut également exprimer la relation **subsort** comme suit :

sorts NzInt Int . **subsorts** NzNat < NzInt Nat < Int .

Où *NzNat* et *Int* sont les ensembles respectivement des entiers naturels non nuls et des entiers.

Un ensemble de déclarations de sous-sortes définit un ordre partiel sur l'ensemble de sortes. Toutefois, pour que cette déclaration soit correcte, l'utilisateur doit éviter les cycles dans les déclarations de sous-sortes. Graphiquement, nous pouvons visualiser cet ordre partiel de la façon suivante :



Opérations

Dans un module, une opération est déclarée à l'aide du mot-clé **op** suivi de son nom, suivi de deux points, suivis d'une série de sortes qui constituent ses arguments (arité), suivis de $\rightarrow$, suivi des sortes des résultats (co-arité), optionnellement suivis de la déclaration d'attributs (présentés plus loin) suivis d'un espace et un point. Le schéma général a la forme suivante :

op <OpName> : <Sort0> ... <Sortk> $\rightarrow$ <Sort> [<OperatorAttributes>] .

Voici quelques exemples de déclarations d'opérations :

op zero : $\rightarrow$ Zero .

op s_ : Nat $\rightarrow$ NzNat .

op sd : Nat Nat $\rightarrow$ Nat .

ops _+_*_ : Nat Nat $\rightarrow$ Nat .

Si l'arité des arguments est vide, l'opération est appelée une constante comme c'est le cas pour le *Zero*. Le nom d'opération est une chaîne de caractères. Les espaces soulignés () jouent un rôle spécial dans ces chaînes de caractères. En effet, si aucun espace souligné ne se produit dans la chaîne de caractères de l'opération comme dans le cas de *sd* alors l'opérateur est déclaré sous la forme *préfixée*. Si les espaces soulignés se produisent dans la chaîne de caractères, alors leur nombre doit coïncider avec le nombre de sortes déclarées comme arguments de l'opérateur. L'opérateur est alors sous la forme dite *mixfixée* où le nième espace souligné indique la position de l'argument de la nième sorte dans les expressions formées avec cet opérateur. Dans l'exemple précédent les opérateurs *s_*, *_+_* et *_** sont sous la forme *mixfixée*.

Il peut y avoir tous les autres caractères avant ou après n'importe lequel de ces espaces soulignés. Si aucun caractère n'apparaît, nous dirons que l'opérateur a été déclaré avec la syntaxe vide. Par exemple, nous pourrions déclarer une sorte *NatSeq* séquence des nombres naturels avec la syntaxe vide comme suit :

sort NatSeq .

subsort Nat < NatSeq .

op _ _ : NatSeq NatSeq $\rightarrow$ NatSeq [**assoc**] .

L'attribut *assoc* dans l'exemple ci-dessus est un attribut déclarant que la concaténation est associative. Avec cette déclaration d'opérateur nous pouvons écrire des termes comme : zero (s zero) (s s zero).

Des opérations ayant les mêmes arité et co-arité peuvent être déclarées simultanément en employant **ops** comme mot-clé et en donnant les sortes non vides après le mot-clé **ops** et avant deux points, comme pour les déclarations de *_+_* et *_** dans l'exemple plus haut.

Variables

Les variables peuvent être déclarées à tout moment avec une syntaxe comportant un identificateur (le nom de la variable), deux points et un autre identificateur (sa sorte).

Par exemple, *N: Nat* déclare une variable nommée N de sorte *Nat*.

Une variable doit être accompagnée de sa sorte. Une variable peut également être déclarée dans un module en utilisant le mot clé **var** suivi d'un identificateur (le nom de variable) suivi des deux points avec l'espace avant et après les deux points, suivi d'un identificateur (sa sorte), suivi d'un espace et d'un point : **var** N : Nat .

Des variables de la même sorte peuvent être déclarées en utilisant le mot-clé **vars**: **vars** M N : Nat .

2.2.2.3. Les Modules Fonctionnels

Les modules fonctionnels définissent les sortes de données et les opérations sur ces données à travers des théories équationnelles. Les sortes de données se composent des éléments qui peuvent être appelés par des termes. Un module fonctionnel est déclaré selon la syntaxe suivante :

fmod <MODULE-NAME> **is** [<DeclarationsAndStatements>] **endfm**

Par exemple: **fmod** NUMBERS **is** ... **endfm**

déclare un module fonctionnel appelé *NUMBERS*. Les points représentent les déclarations et les expressions qui peuvent apparaître dans le module. Les déclarations incluent l'importation d'autres modules fonctionnels, les sortes, les sous-sortes, et des déclarations d'opérations. Les expressions incluent des axiomes équationnels et d'appartenance.

Équations inconditionnelles

Les équations sont déclarées en utilisant le mot-clé **eq**, suivi par un terme (la partie gauche), le signe d'égalité, puis un terme (la partie droite), et optionnellement suivi par des attributs encadrés par des crochets, terminés par un espace et un point. Ainsi le schéma général est le suivant :

eq <Term1> = <Term2> [<StatementAttributes>] .

Les termes *t* et *t'* dans une équation $t = t'$ doivent être de même sorte. Pour que l'équation s'exécute, chaque variable qui apparaît dans *t'* doit apparaître dans *t*. Nous pouvons par exemple ajouter des équations relatives à l'addition dans le module *NUMBERS* :

vars N M : Nat . **eq** N + zero = N .

eq N + s M = s (N + M) . //Avec *s* défini précédemment comme opérateur unaire.

Équations conditionnelle

Les conditions dans les équations conditionnelles se composent de différentes équations $t = t_0$. Une condition peut être une équation simple, ou une conjonction d'équations en utilisant le "**and**" qui est associative.

Ainsi la forme générale des équations conditionnelles (**ceq**) est la suivante :

ceq <Term1> = <Term2>

if <EqCondition-1> **and** ... **and** <EqCondition-k> [<StatementAttributes>] .

En plus, la syntaxe concrète de la condition équationnelle "**EqCondition**" est peut être une équation booléenne dite abrégée de la forme *t*, avec *t* un terme dans la sorte *Bool* abrégant l'équation $t = true$. Ainsi, ces conditions peuvent apparaître dans une équation : $(N == zero) = true$.

$(M \neq s zero) = true$. $(N > zero \text{ or } M \neq s zero) = true$.

Attributs équationnels

Les déclarations d'opérateur peuvent inclure des attributs qui fournissent des informations additionnelles à l'opérateur : sémantique, syntaxique, pragmatique, etc. Tous ces attributs sont déclarés dans une seule paire de crochets, après la sorte du résultat et avant le point.

- **assoc** (associativité),
- **comm** (commutativité),
- **id**: <Term> (identité).

Ces attributs sont permis seulement pour les opérateurs binaires dont les arguments appartiennent au même composant. Un opérateur peut être déclaré avec n'importe quelle combinaison de ces attributs qui peuvent apparaître dans n'importe quel ordre dans la déclaration.

Dans notre exemple de nombres nous pouvons ajouter *nil* pour exprimer le vide et raffiner la déclaration de la concaténation des séquences des nombres, de sorte que la concaténation soit associative avec l'identité *nil*.

op nil : $\rightarrow \text{NatSeq}$. **op** _ : $\text{NatSeq NatSeq} \rightarrow \text{NatSeq}$ [**assoc id**: nil] .

2.2.2.4. Les Modules Systèmes

Un module système spécifie une théorie de réécriture. Une théorie de réécriture a des *sortes*, et des *opérations*, et peut avoir des : *équations* et *règles qui peuvent être conditionnelles*.

Les règles spécifient les transitions concourantes locales qui peuvent avoir lieu dans un système si le patron dans le côté gauche de la règle correspond à un fragment de l'état de système et la condition de la règle est satisfaite. Dans ce cas, la transition indiquée par la règle peut avoir lieu, et le fragment de l'état correspondant est transformé en l'instance correspondante du côté droit. Par exemple, Comme cela a été déjà mentionné auparavant, un module système "BANK-ACCOUNT" peut être déclaré comme suit : **mod** BANK-ACCOUNT **is** ... **endm**

Où les points de suspension correspondent à toutes les déclarations et expressions dans un module:

- | | |
|---------------------------|---|
| 1. importations, | 5. équations (conditionnels et non conditionnels), |
| 2. sortes et sous-sortes, | 6. règles de réécriture (conditionnelles et non conditionnelles). |
| 3. opérations, | |
| 4. variables, | |

Les déclarations 1-5 sont exactement les mêmes que celles dans les modules fonctionnels. Les modules systèmes rajoutent les règles de réécriture que nous présentons.

Règles de réécriture

Il n'y a (mathématiquement) aucun comportement dynamique dans des spécifications équationnelles. En effet, dans le modèle mathématique des spécifications équationnelles de même que dans la logique équationnelle, deux expressions sont soit équivalentes soit aucune relation n'existe entre elles. Hors pour les systèmes dynamiques, qui évoluent avec le temps, cet arrangement équationnel n'a pas de sens mathématique ou logique. Pour illustrer un système dynamique, regardons le modèle simplifié de la vie d'un être humain.

Dans cet exemple, un terme : *person('Peter, 32)*

Dénote une personne avec le nom 'Peter qui a 32 ans. Pour simuler sa vie (et par la suite son vieillissement), le prochain état devrait être : *person('Peter, 33)*

Une façon de modéliser un système de simulation d'âge serait l'équation :

$$person(X,N) = person(X,N+1)$$

Dans un tel système il serait logiquement vrai que : *person('Peter,32) = person('Peter,33)*

Ce changement est réversible – de part la propriété de symétrie de la logique équationnelle – de sorte qu'on aura aussi: *person('Peter,33) = person('Peter,32)*, Et *person('Peter,32) = person('Peter,20)*

Cependant dans le système que nous modélisons, cette propriété intuitivement ne devrait pas se tenir car on ne peut malheureusement diminuer notre âge. Le changement n'est habituellement pas réversible dans les systèmes dynamiques. En d'autres termes, pour spécifier (ou changer/évoluer) un système dynamique, nous ne pouvons pas employer seulement des équations. En revanche, nous employons la logique de réécriture, qui prolonge des techniques algébriques de spécifications équationnelles pour modéliser les systèmes dynamiques.

Règles non Conditionnelles de Réécriture

Le problème avec l'utilisation des équations pour modéliser le comportement dynamique est, comme indiqué auparavant, le fait que l'égalité soit symétrique. En logique de réécriture, le comportement dynamique est modélisé par des règles de réécriture. Une règle de réécriture est essentiellement une équation "à sens unique", et la logique de réécriture est de ce fait une logique équationnelle sans symétrie. Ainsi, le comportement dynamique d'une personne dans notre exemple peut être modélisé par une règle de réécriture : $person(X, N) \Rightarrow person(X, N+1)$

Pour des variables appropriées X et N . Dans la logique de réécriture $t \Rightarrow t'$, est prévu pour signifier que l'état t peut changer en un état t' . Les règles de réécriture donnent les changements d'états locaux et atomiques comme dans l'exemple précédent.

Les règles non conditionnelles sont déclarées avec la syntaxe:

rl [<étiquette >] : < Term1 > $\Rightarrow$ < Term2 > .

Une règle de réécriture peut avoir une étiquette qui appelle l'action ou l'événement qui change l'état. Dans notre exemple, une règle étiquetée s'écrit avec la syntaxe:

rl [birth_day] : person(X,N) $\Rightarrow$ person(X,N+1) .

Ou

rl [getting_older] : person(X,N) $\Rightarrow$ person(X,N+1) .

L'étiquette n'influence pas le calcul/déduction dans la logique de réécriture.

Règles Conditionnelles de Réécriture

Les règles conditionnelles de réécriture peuvent avoir des conditions très générales impliquant des équations et autre réécritures.

Dans leur représentation en Maude, les règles conditionnelles sont déclarées avec la syntaxe :

crl [<étiquette >] : < Term1 > $\Rightarrow$ < Term2 > **if** <Condition-1> and .. and <Condition-k> .

2.2.2.5. Les Modules Orientés-Objet

Les modules orientés objets peuvent être réduits aux modules systèmes. Par rapport aux modules systèmes, les modules orientés objets offrent une syntaxe plus appropriée pour décrire les entités de base du paradigme objet comme, parmi autres: classes, objets, messages et configuration.

- Dans un système orienté objet concurrent, l'état concurrent, appelé une configuration (ligne 1), a typiquement la structure d'un multi-ensemble, composé d'objets et des messages (ligne 2). Donc, nous pouvons voir la création des configurations par un opérateur binaire d'union de multi-ensembles que nous pouvons le représenter par la syntaxe de la cinquième ligne :

- 1) **sort** Configuration .
- 2) **sorts** Object Msg .
- 3) **subsorts** Object Msg < Configuration .
- 4) **op** null : -> Configuration .
- 5) **op** _ : Configuration Configuration -> Configuration [assoc comm id : null] .

Où l'opérateur d'union des multi-ensembles est déclaré pour satisfaire les lois structurelles de l'associativité et la commutativité et avoir le *null* comme identité. La déclaration de *subsort* à la troisième ligne ci-dessus déclare que les états de l'objet et des messages sont des configurations singletons.

- Un objet dans un état donné est représenté comme un terme : $\langle O : C \mid a_1 : v_1, \dots, a_n : v_n \rangle$

Où O est le nom de l'objet ou l'identificateur, C est sa classe, a_i sont les attributs de l'objet et les v_i sont les valeurs correspondantes.

- Chaque classe est déclarée par la syntaxe : **class** $C \mid a_1 : S_1, \dots, a_n : S_n$.

Où C : nom de la classe, a_i des attributs, S_i : type de chaque attribut.

- La déclaration d'un message est défini par la syntaxe: **msg** $m : p_1, \dots, p_n \rightarrow \text{Msg}$.

Où m : nom de message, p_i : sortes des attributs associés.

Des messages de mêmes sortes peuvent être déclarés en utilisant le mot-clé **msgs**.

- Maude permet aussi l'héritage de classe : une déclaration de sous-classe est définie par la syntaxe : **subclass** $C' < C$.

Tous les attributs, messages, et règles de la superclasse C , aussi bien que les attributs récemment définis, messages et règles de la sous-classe caractérisent la structure et le comportement de la sous-classe C' . L'héritage multiple est supporté par Maude.

- Une seule règle (conditionnelle) de réécriture permet d'exprimer la consommation de certains messages, l'envoi de nouveaux messages, la destruction d'objets, la création de nouveaux objets, le changement de l'état de certains objets, etc.

La forme générale de ces règles est donnée comme suit :

$$\begin{aligned} & M_1 M_2 \dots M_n \langle O_1 : C_1 / \text{ListeAt}_1 \rangle \dots \langle O_m : C_m / \text{ListeAt}_m \rangle \\ \Rightarrow & \langle O_1 : C_1 / \text{ListeAt}'_1 \rangle \dots \langle O_k : C_k / \text{ListeAt}'_k \rangle \langle O_l : C_l / \text{ListeAt}''_1 \rangle \dots \langle O_p : C_p / \text{ListeAt}''_p \rangle \\ & M'_1 M'_2 \dots M'_q \end{aligned}$$

- Les messages $M_1 M_2 \dots M_n$ sont supprimés après l'exécution de la règle.
- Les états des objets $O_1, \dots, O_k$ sont modifiés.
- Les objets $O_{k+1}, \dots, O_m$ qui n'apparaissent que dans la partie gauche sont supprimés.
- De nouveaux objets $O_l, \dots, O_p$ sont créés.
- De nouveaux messages $M'_1 M'_2 \dots M'_q$ sont créés.

La Figure 3.5 illustre le module système *BANK-ACCOUNT* qui définit un objet compte bancaire A et les deux opérations capables d'affecter le contenu de son (crédit et débit) en exécutant la règle de réécriture définie dans ce module. La Figure 3.6 représente le même module *BANK-ACCOUNT* mais avec une syntaxe orienté objet plus appropriée.

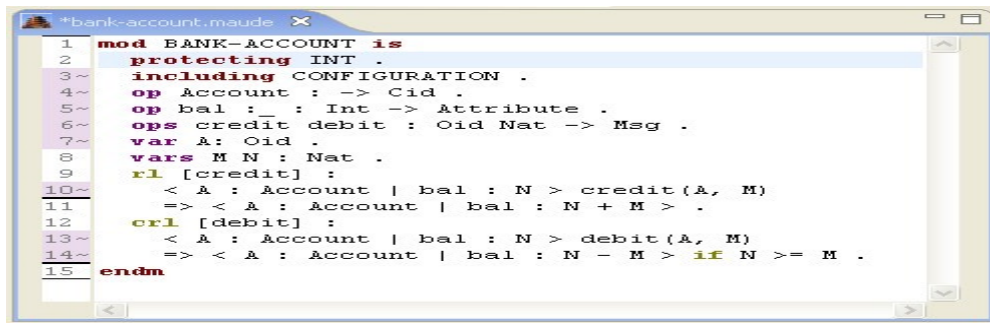


Figure 3.5. Le module système BANK-ACCOUNT.

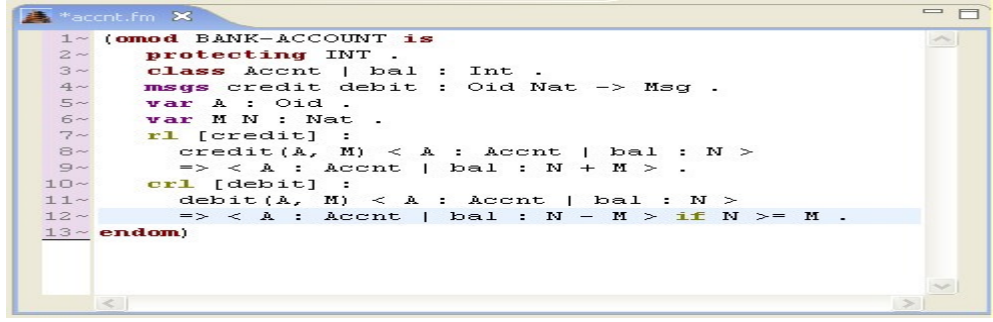


Figure 3.6. Le module orienté-objet BANK-ACCOUNT.

Nous notons, qu'après l'exécution de la règle inconditionnelle *[credit]*, le message *credit* (*A*, *M*) est consommé et le contenu de compte *A* est augmenté de *M*. De la même façon l'exécution de la règle conditionnelle *[debit]* exige que la condition ($N \geq M$) soit vérifiée. L'exécution de telle règle produit la consommation du message *debit* (*A*, *M*) et la réduction du contenu de compte *A* (Figure 3.7).

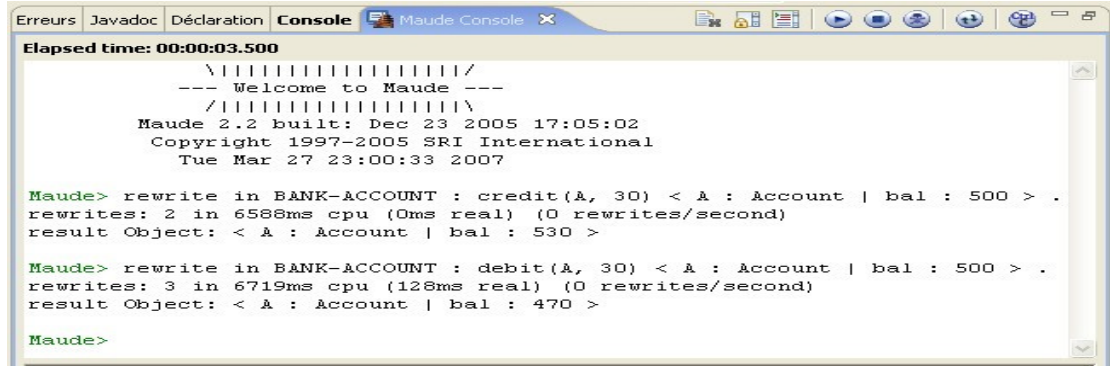


Figure 3.7. L'exécution du module BANK-ACCOUNT.

Pour plus de détail sur les notions de bases inhérentes à Maude le lecteur pourra se référer à [Cla05]. Maude est considéré comme un méta-langage pour développer d'autres langages, en effet Real-Time Maude (RT-Maude) est implémenté en Maude comme une extension de Full Maude.

2.2.3. Extension Maude : Real Time-Maude

Une théorie de réécriture RT-Maude est une théorie de réécriture Maude, qui contient en plus la spécification [Olv06]:

- D'une sorte *Time* pour décrire le domaine du temps qui peut être discret ou continue,
- d'une sorte *GlobalSystem* avec un constructeur " $\{_ \}$ ": $\{_ \} : System \rightarrow GlobalSystem$
- Et un ensemble de règles *tick* qui modélisent le temps écoulé dans le système qui ont la forme suivante : $\{t\} \Rightarrow \{t'\} \text{ in time } u \text{ if cond}$

Où *u* est un terme qui peut contenir des variables de sorte *Time* qui dénote la durée de la règle, les termes *t* et *t'* sont des termes de sorte *System* qui dénote l'état du système. Les règles de réécriture qui ne sont pas des règles *tick* sont des règles *instantanées* supposées prendre un temps zéro.

L'état initial doit toujours avoir la forme $\{t''\}$, où *t''* est un terme de sorte *System*, afin que la forme des règles *tick* assure que le temps s'écoule uniformément dans toutes les parties du système.

Les théories de réécriture temps réel sont spécifiées en RT-Maude comme des modules temporisés "*tmod*" (timed modules) ou des modules temporisés orientés-objet "*tomod*" (timed object-oriented modules).

Real-Time Maude est implémenté en Maude comme une extension de Full Maude, donc tous les commandes et les modules doivent être déclarés entre parenthèses:

```
(tmod NAME is ... endtm)
(tomod NAME is ... endtom)
```

Dans ce qui suit nous introduisons d'abord les *modules temporisés (Timed modules)*. Nous présenterons ensuite quelques exemples spécifiques à chacun des types de modules (temporisés et temporisés orientés-objet), plus de détails pourront être trouvés dans [Hui00].

2.2.3.1. Les Modules Temporisés (Timed modules)

Chaque module temporisé importe automatiquement le module fonctionnel *TIMED-PRELUDE* qui inclue les déclarations cruciales :

```
Sorts System GlobalSystem .
op {} : System -> GlobalSystem .
```

L'état d'un système doit être représenté par un terme de sorte *System*. Un module temporisé importe aussi le squelette du domaine de temps qui est décrit ci-dessous.

Les domaines de temps (time domains)

Chaque module temporisé importe automatiquement le module *TIME* suivant :

```
fmod TIME is
sorts Time NzTime . subsort NzTime < Time . op zero : -> Time .
op _plus_ : Time Time -> Time [assoc comm] . op _monus_ : Time Time -> Time .
ops _le_ _lt_ _ge_ _gt_ : Time Time -> Bool. eq zero plus R:Time = R:Time .
eq R:Time le R':Time = (R:Time lt R':Time) or (R:Time == R':Time) .
eq R:Time ge R':Time = R':Time le R:Time . eq R:Time gt R':Time = R':Time lt R:Time .
endfm
```

Ce module définit un squelette du domaine de temps, avec la valeur *zéro* et quelques symboles de fonction. En effet l'utilisateur a la liberté complète pour spécifier le type de données des valeurs de temps qui peuvent être discret ou bien continue par :

- Importer un type de données spécifique de valeurs temporelles qui satisfont la théorie du Temps, avec une sorte, dit *TimeValues* pour telles valeurs, et
- Déclarer une inclusion *subsort TimeValues < Time* et donner les équations appropriées pour interpréter le constant zéro et les opérateurs *_plus_*, *_monus_*, et *_lt_* dans *TimeValues*.

RT-Maude contient le module suivant qui définit le domaine de temps comme étant des nombres entiers :

```
fmod NAT-TIME-DOMAIN is
including LTIME . protecting NAT .
subsort Nat < Time . subsort NzNat < NzTime .
vars N N' : Nat . eq zero = 0 . eq N plus N' = N + N' .
eq N monus N' = if N > N' then sd(N, N') else 0 fi . eq N lt N' = N < N' .
endfm
```

Pour spécifier les domaines de temps continue, RT-Maude définit un sous ensemble (subsort) *NNegRat* de nombres rationnels non négatifs, et, dans le module *POSRAT-TIME-DOMAIN*, il définit le domaine de temps comme étant des nombres rationnels.

```
fmod POSITIVE-RAT is
protecting RAT .   sort NNegRat .   subsorts Zero PosRat Nat < NNegRat < Rat .
endfm
```

```
fmod POSRAT-TIME-DOMAIN is
including LTIME .   protecting POSITIVE-RAT .
subsort NNegRat < Time .   subsort PosRat < NzTime .
vars R R' : NNegRat .
eq zero = 0 .   eq R plus R' = R + R' .
eq R minus R' = if R > R' then R - R' else 0 fi .   eq R lt R' = R < R' .   endfm
```

Pour résumer et pour spécifier le domaine de temps, l'utilisateur peut :

- Soit laisser le domaine de temps non spécifié pour les instanciations plus tardifs, ou
- Importé soit de l'un des modules *NAT-TIME-DOMAIN* ou *POSRAT-TIME-DOMAIN*, ou
- définir explicitement son propre module temporisé.

Les règles Tick

La syntaxe des règles *tick* inconditionnelles est : $rl [l] : \{t\} \Rightarrow \{t'\} \text{ in time } u .$

Une règle *tick* conditionnelle est écrite avec la syntaxe : $crl [l] : \{t\} \Rightarrow \{t'\} \text{ in time } u \text{ if cond} .$

Il faut noter que chaque module temporisé import automatiquement les sortes *System* et *GlobalSystem* et l'opérateur $\{\}$ aussi bien qu'un domaine de temps réduit, nous sommes maintenant prêt à en spécifier quelques exemples.

Exemples de Modules Temporisés : Modélisation d'une horloge

Dans une série d'exemples nous donnons la spécification d'une horloge simple qui montre le temps. Le terme *clock* (*t*) dénote une horloge qui montre le temps. Le module suivant (Figure 3.8) spécifie une horloge discrète où le temps avance toujours par une unité de temps dans chaque pas "tick". Puisque le temps est discret nous utilisons les nombres naturels comme un domaine de temps et donc nous importons le module *NAT-TIME-DOMAIN* :

Le système avance le temps par 1 dans chaque pas, avec le résultat que la valeur de l'horloge augmente par 1.

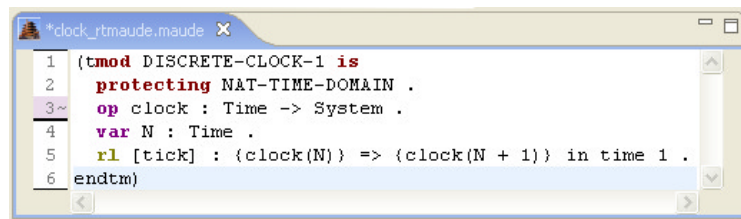


Figure 3.8. Le module temporisé DISCRETE-CLOCK-1.

Nous savons tous qu'une horloge peut s'arrêter à cause d'une batterie plate ou autres échecs. Le module prochain (Figure 3.9) modélise une horloge qui peut s'arrêter à tout moment, dans le cas où l'état est *stopped-clock(r)*, où *r* était la valeur de l'horloge quand celle ci est arrêtée. Une horloge discrète qui peut s'arrêter peut être modélisée par le module temporisé suivant :

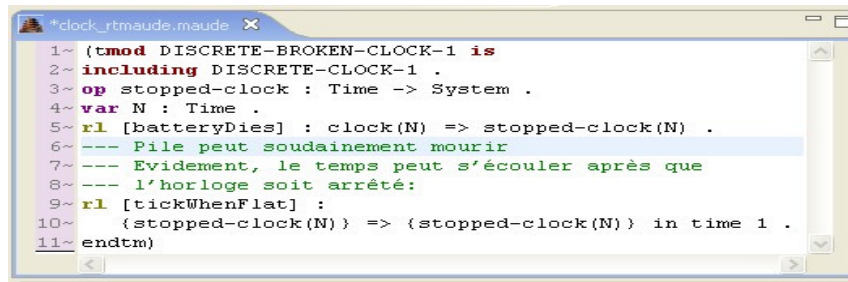


Figure 3.9. Le module temporisé DISCRETE-BROKEN-CLOCK-1.

Le temps peut s'écouler bien que l'horloge est arrêtée, car nous savons tous que le temps s'écoule sans faire changer l'état d'une horloge arrêtée.

Supposer maintenant des horloges qui comptent les heures, donc quand l'horloge atteint 24 elle devrait montrer 0. Ce cadre peut être modélisé de différentes façons. La spécification suivante (Figure 3.10) augmente juste la valeur de l'horloge modulo 24.

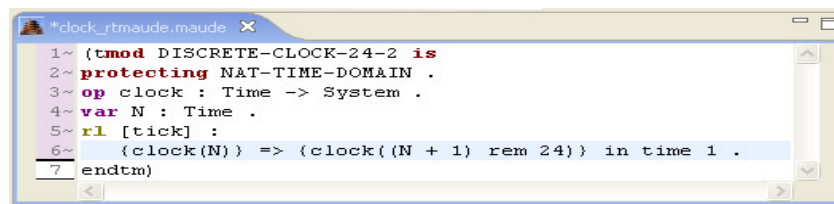


Figure 3.10. Le module temporisé DISCRETE-CLOCK-24-2.

Pour obtenir une spécification légèrement plus intéressante (légèrement différente), le module suivant (Figure 3.11) a une opération *reset* qui réinitialise une horloge avec la valeur 24 à 0 :

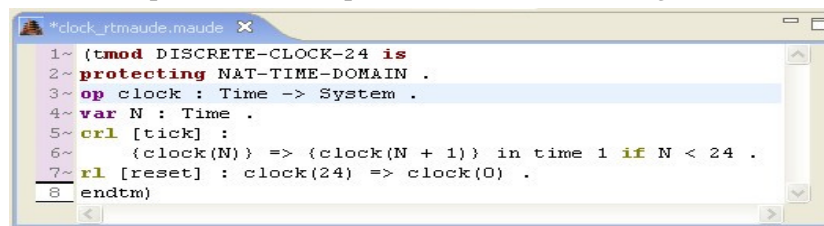


Figure 3.11. Le module temporisé DISCRETE-CLOCK-24.

L'état $\{clock(24)\}$ passe à l'état $\{clock(0)\}$ en utilisant une transition instantanée. La condition dans la règle *tick* assure que le temps ne peut pas s'écouler quand l'horloge en montre 24, donc $\{clock(24)\}$ doit être réinitialisé à $\{clock(0)\}$ avant que le temps continue à avancer.

Bien que le temps discret est approprié pour modéliser les systèmes temps réel tel que la planification des algorithmes temps réel et des protocoles de communication, les systèmes hybrides sont souvent modélisés en utilisant le domaine de temps continu. La spécification suivante (Figure 3.12) modélise l'horloge décrit précédemment en utilisant le domaine de temps continu, à savoir les nombres rationnels non négatifs :

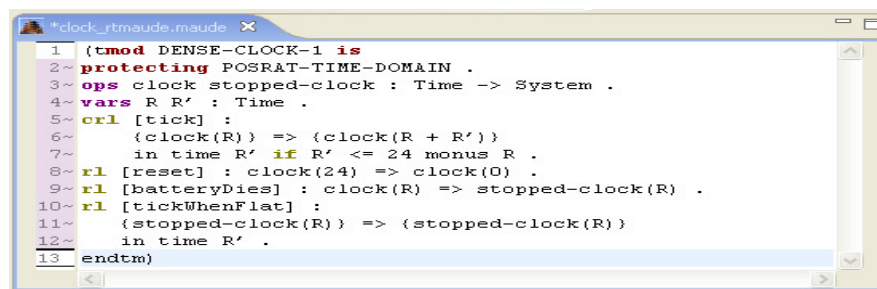


Figure 3.12. Le module temporisé DENSE-CLOCK-1.

Bien que l'exemple soit assez simple, les règles *tick* au-dessus sont typiques pour les systèmes avec le domaine de temps continu. A partir d'un état donné, le temps peut s'écouler par n'importe quelle valeur de temps jusqu'à ce que l'horloge atteigne la valeur 24. Cette augmentation non déterministe du temps est modélisée dans les règles *tick* par l'existence d'une nouvelle variable R' qui n'apparaît pas dans le côté gauche de la règle, et qui n'est pas instancié dans la condition de la règle. Cette variable R' peut en suite prendre n'importe quelle valeur entre 0 et $24-R$.

2.2.3.2. Les Modules Temporisés Orientés-Objet (Timed Object-Oriented Modules)

Les modules temporisés orientés objet, définit entre les mots clés *tomod* et *endtom*, étendent les modules orientés objet de full Maude et les modules temps pour supporter la spécification orientée objet des systèmes temps réel. Nous prenons l'exemple d'une spécification de "protocole" pour calculer le temps d'aller-retour (*Round Trip Time*) entre les nœuds d'un réseau (c.à.d. le temps que prend un message pour aller et retourner entre deux nœuds *initiateur* et *répondeur*, dans un réseau).

Dans le protocole qui suit nous supposons qu'il y a un nœud expéditeur (*initiateur*) et un nœud répondeur dans le réseau, chacun d'entre eux est modélisé par un objet. La communication est modélisée par des messages ordinaires. Le protocole est très simple : l'objet *sender* (de la class *Sender*) qui a une horloge locale, débute le protocole (dans la règle *rttRequest*) par l'envoi d'un message *rtt?(r)* à l'objet *responder* où r est la valeur actuelle de l'attribut *clock* du *sender*. Quand l'objet *responder* lit le message *rtt?(r)* (dans la règle *rttResponse*), il répond en envoyant un message *rttAck(r)* (c.à.d. avec le même temps qu'il a lu dans le message *rtt?*) en retour à *sender*. Finalement (dans la règle *treatRttresp*), l'objet *sender* reçoit le message *rttAck* avec le temps (originale), et trouve la valeur *rtt* (*round trip time*) en comparant le temps reçu au temps réel, qui est donné par son horloge locale. Le module temporisé RTT-1 (Figure 3.13) ne fixe pas le domaine de temps ou les noms des objets.

Le temps peut s'écouler par toute valeur dans la règle *tick*. Dans ce chemin, toute valeur de temps (en incluant le zéro) peut s'écouler entre les événements *rttRequest* et *rttResponse* et entre les événements *rttResponse* et *treatRttResp*. L'effet de temps écoulé sur une configuration est que l'horloge locale du sender est augmentée relativement au temps écoulé.

Le module *DISCRETE-RTT-1* (Figure 3.14) instancie les noms des objets et le domaine de temps et définit un état initial à partir duquel on peut exécuter la spécification.

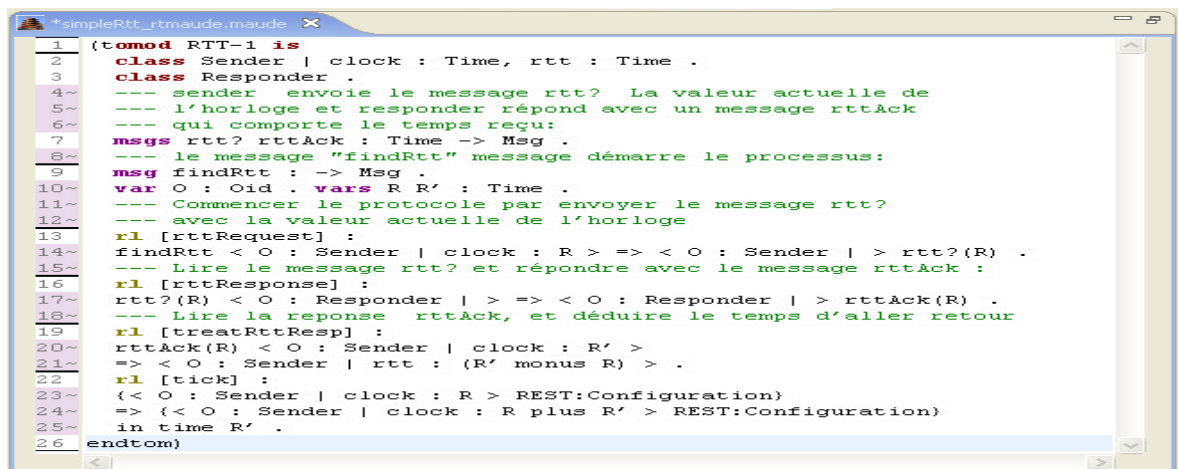


Figure 3.13. Le module temporisé orienté-objet RTT-1.

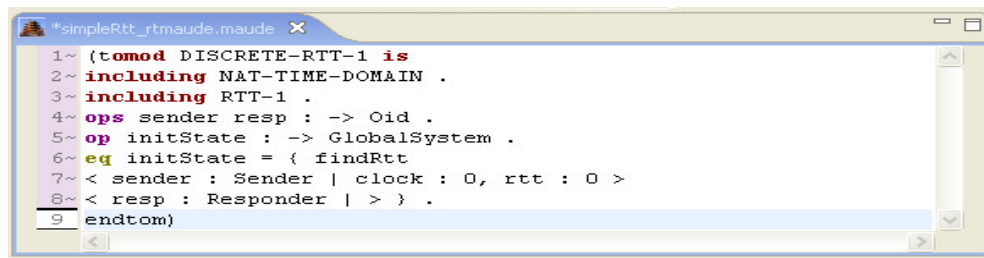


Figure 3.14. Le module temporisé orienté-objet DISCRETE-RTT-1.

2.2.4. Vérification de Modèles

Après avoir étudié les langages de modélisation orientés agent et de spécification formelle, il est tout à fait nécessaire de vérifier la conformité des modèles obtenus lors d'un raffinement sur la base des propriétés d'un modèle. De ce fait, nous présentons ici, la technique de vérification de propriétés supportée par le langage Maude qui est celle du model-checking (vérification de modèles) [Lar00, Mer00] permettant d'assurer la validité de l'aspect comportemental des modèles.

D'après Gagnon [Gag07] le model-checking consiste en la construction d'un modèle généralement fini d'un système en développement, et de s'assurer sur ce modèle si une série de propriétés y sont vérifiées. Cette technique fait une recherche exhaustive au sein de l'espace des états du système [Cla96, Che96] et a besoin de deux descriptions différentes de lui pour effectuer la vérification [Del02], qui sont :

- Un modèle du système en cours de développement qui doit être exécutable;
- Des spécifications qui représentent ce que le système doit être en mesure d'accomplir lorsqu'il sera complété. Ces spécifications seront exprimées sous la forme de "formules logiques temporelles" et seront vérifiées au sein du "modèle du système" pour tenter de savoir si le système développé les satisfait ou non.

Selon Alkhodre [Alk04] le principe général du model-checking peut être présenté comme suit. La formule de logique temporelle (par exemple LTL [Che01]) est traduite en un système de transitions (automate de Büchi [Ete00]) qui est composé (synchronisé) avec le modèle du système. L'exploration exhaustive de l'espace d'états du système de transitions global permet de répondre formellement à la question : le modèle vérifie t-il la propriété ?

En ce qui concerne les avantages de la vérification de modèles, Edmund M. Clarke dans [Cla96] les résume ainsi :

- Rapidité (des résultats peuvent être fournis en quelques minutes seulement) ;
- Lorsqu'une erreur est rencontrée, le vérificateur fournit un contre-exemple montrant comment il a atteint l'état d'erreur ;
- Un système spécifié que partiellement ne cause pas de problème ;
- Les diverses notations logiques utilisées pour définir des propriétés ont la possibilité d'exprimer assez aisément les concepts de concurrence.

Cependant, cette technique souffre du problème de l'explosion de l'espace des états du système (notamment dans les systèmes industriels, qui ont tendance à être volumineux comparativement aux applications académiques étudiées). Pour remédier à ça, trois directions complémentaires peuvent être suivies [Bab00] :

- La prise en compte de symétries tant au niveau architectural qu'au niveau des données manipulées peut offrir une réduction importante de l'espace des états,

- Les techniques dites « d'ordre partiel » qui tentent d'éviter la part d'explosion combinatoire due à la représentation du parallélisme par l'entrelacement d'actions permettent dans de nombreux cas d'éliminer le facteur exponentiel.
- De même, les techniques de vérification à la volée permettent de guider l'exploration et d'en limiter l'explosion.

La Figure 3.15 montre un bref aperçu de la technique de model-checking à son origine [Cla03].

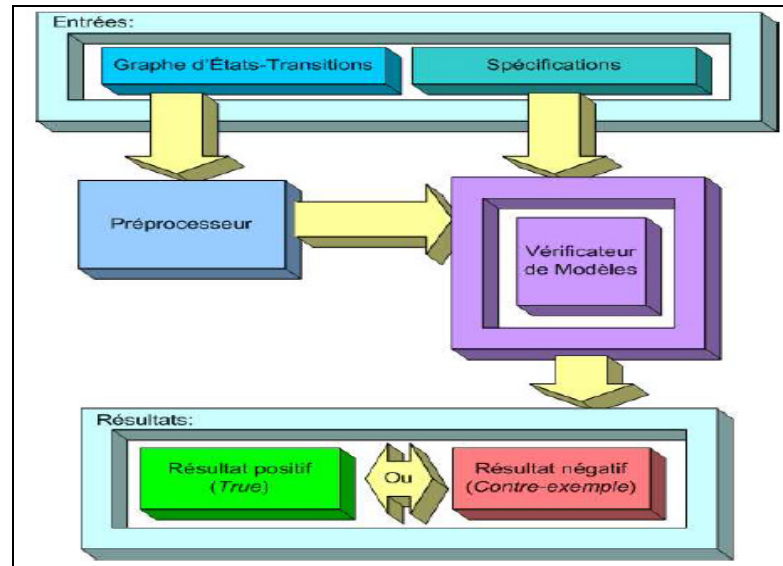


Figure 3.15. Aperçu du processus de vérification de modèles [Gag07].

Au sujet des propriétés de logique temporelle qui seront définies en vue de vérifier un système, qu'il soit logiciel ou matériel, Gagnon dans son travail [Gag07] les classe en trois catégories principales, que Wahl [Wah03] résume ainsi :

- Les propriétés de sécurité qui visent principalement à s'assurer que le système respecte certaines conditions primordiales lors de son exécution. par exemple, s'assurer que l'exclusion mutuelle est toujours respectée au sein d'un système ;
- Les propriétés de vitalité qui tentent de s'assurer du bon fonctionnement du système. Dans le cas de systèmes concurrentiels, ça peut être l'absence d'interblocage ou encore le fait que toute demande (Request) est suivie par une réponse (Grant), etc. ;
- Les propriétés d'égalité qui essayent de s'assurer que chaque tâche aura un temps d'accès au processeur équitable comparativement aux autres, notamment au sein des systèmes concurrentiels.

De plus dans [Del02, Che01] ces propriétés sont considérées de deux natures distinctes :

- Propriété désirable que l'on souhaite que le système en développement soit en mesure de satisfaire, et
- Propriété non désirable que l'on ne désire pas retrouver au sein d'un système en développement.

En résumé les propriétés temporelles sont généralement *la sûreté* de fonctionnement (quelque chose de mauvais n'arrive jamais) et *la vivacité* (quelque chose de bon est toujours possible). D'autre part, la tâche de vérification comme définie par Gallardo et ses collègues dans [Del02] est une tentative de vérifier si tous, quelques-uns ou aucun des chemins d'exécution du design courant du système en développement satisfont une propriété quelconque, qui peut être désirable

ou non *désirable*. Cette tâche est effectuée automatiquement en générant et analysant tous les états potentiels du système (dans la mesure où le nombre d'états est fini). Lorsqu'une propriété est vérifiée ou non, l'outil retourne un résultat positif ou un contre-exemple illustrant le chemin qui a mené à cet état d'erreur. Parmi les outils qui exploitent la technique de model-checking on trouve SPIN [Hol91], SMV [Mcm91], Maude [Cla05], etc.

3. Synthèse

Les travaux portant sur le passage d'UML/AUML à Maude n'étant pas très étendue [Mok07, Mok09, Mok10, Lao10, Lao13, Lao14], nous discutons de la problématique du passage d'un langage de modélisation à un langage de spécifications formelles d'une façon plus générale. Il faut noter que cette synthèse est principalement liée aux conclusions de Sadoun dans son travail de thèse [Sad14].

Parmi les langages de modélisation les plus connus et utilisés, il ya UML (et ses extensions AUML, SysML, MSC, LSC), RDF, RDFS et OWL. La notation UML est basée sur l'utilisation de diagrammes. Néanmoins, ces représentations ne fournissent pas le niveau de concision et d'expressivité qu'un langage textuel formel peut offrir [Cla06b]. Aussi les diagrammes UML sont souvent complétés par la définition de contraintes OCL qui ne peuvent être exprimées graphiquement, et que le modèle doit satisfaire. Le fait qu'UML ne possède pas une sémantique précise est un sérieux inconvénient [Cha13] qui a poussé un grand nombre de chercheurs à s'intéresser à la transformation d'un modèle UML en un langage de spécifications formelles tel que B [Ngu98, Led01, Lal01, Tru06], Object-Z ou les réseaux de Petri [Gul13] ou encore Maude [Cla06b, Mok09, Rol10, Dur12].

L'auteur de [Sad14] fait savoir aussi que la plupart des travaux qui font un passage d'UML à Maude traitent systématiquement la transformation du diagramme de classes car c'est lui qui décrit les composants du système. Ils se distinguent ensuite par la traduction d'autres diagrammes, de contraintes OCL ou par les correspondances choisies entre éléments UML et Maude. En règle générale, dans une traduction de UML à Maude, chaque type simple UML (entier, réel, booléen, string) est associé à la sorte (type) Maude prédéfinie correspondante [Cla06b, Rol10, Dur12].

Dans [Cla06b] l'approche proposée consiste en une traduction du modèle UML et des contraintes OCL [Cla06a] en une spécification équationnelle. Cependant, dans [Mok09] la traduction proposée utilise à la fois équations et règles de réécriture. Elle consiste à produire une spécification formelle Maude à partir de l'étude de différents diagrammes UML (de classes, de cas d'utilisation, de communication et d'états-transitions). La traduction produit un module orienté objet Maude pour la description de chaque classe ainsi qu'un module orienté objet pour la description de l'ensemble des messages possibles entre deux objets. Le reste des spécifications donne lieu à la génération de modules standards. Le plus souvent, les diagrammes de classes sont traduits sous forme de modules orientés objet Maude, et l'état du système par des configurations d'objets [Rom07, Riv09, Bor10] ou par des configurations d'objets et de messages [Rol10, Dur12, Cha13].

Alors que, Duràn et Roldàn [Dur12] s'intéressent à la validation de contraintes OCL qui se trouvent dans les diagrammes de classes et en proposent une traduction UML/OCL vers Maude afin de simuler des modèles UML en exécutant leurs spécifications Maude correspondantes. Cette traduction a pour objectif de permettre la vérification des scénarios qui répondent ou qui violent des contraintes données [Dur11]. D'autre part, la majorité des approches de traduction des langages d'ontologie RDF et OWL vers des langages de spécifications formelles exploitent leurs

aspects orientés objet. Par exemple [Ver05, Sen05, Hua09] proposent en général une traduction vers un module orienté objet Maude. En effet, Huang et al. [Hua09] utilisent Maude afin de construire un modèle mathématique bien défini d'un système décrit en OWL-S. La méthode de transformation proposée consiste à définir une abstraction des sous-ensembles du langage OWL-S, puis à définir pour chaque sous-ensemble, la syntaxe et la sémantique correspondantes en logique de réécriture.

Pour la vérification formelle, Gueffaz [Gue12] dans son travail de thèse, investit la vérification formelle de graphes sémantiques RDF à l'aide de techniques de model-checking. Pour cela, il propose une approche de transformation et de vérification nommée ScaleSem. Cette approche est fondée sur une transformation de graphe et a donné lieu à deux outils : RDF2SPIN [Gue11b] qui transforme des graphes RDF en langage Promela pour une vérification à l'aide du model-checker SPIN [Hol03]. et RDF2NuSMV [Gue11a] qui transforme des graphes RDF en langage NuSMV [Cim00]. L'approche ScaleSem consiste à explorer l'ensemble du graphe RDF pour en obtenir tous les triplets, puis déterminer une racine globale pour le graphe et donner une représentation du graphe résultat en modèle écrit dans le langage Promela ou le langage NuSMV. Enfin, les propriétés du modèle sont vérifiées avec l'un des deux model-checkers SPIN ou NuSMV.

Dans ce chapitre plusieurs approches pour la spécification formelle des systèmes ont été présentées mais notre choix s'est fait sur Maude car comparativement à des langages tels Object-Z ou B, il a l'avantage de permettre la définition de ses propres notations (Maude est un méta langage). Ceci lui permet d'être plus expressif et beaucoup plus facile à comprendre qu'aux notations abstraites et mathématiques d'Object-Z et autres. Il offre également la possibilité de valider les descriptions développées à l'aide de simulations. Ce qui n'est pas le cas avec d'autres langages, comme par exemple Object-Z. En effet, l'environnement Maude est très flexible par rapport aux simulations puisqu'il permet de sélectionner une configuration initiale personnalisée, ce qui a pour effet de permettre la vérification d'une partie du système sans toutefois compromettre ce qui a été accompli auparavant.

De plus, Gagnon dans [Gag07] rajoute que Maude a l'avantage d'être un langage de programmation proprement dit, comparativement aux langages de spécification formelle et de vérification. Ces derniers sont très souvent spécifiques de ce type de technique, obligeant ainsi l'utilisateur à apprendre un langage de spécification ou de vérification avant de pouvoir bénéficier des avantages de chacune de ces techniques. Comme c'est le cas du langage de spécification formelle Promela qui est le langage d'entrée du vérificateur de modèle SPIN. Heureusement, Maude comble ces deux aspects dans un seul et unique langage. Une notation Maude servira autant de spécification formelle (et à la vérification de modèles par le fait même), que de langage de programmation permettant le développement du produit. Un usager n'aura alors qu'à apprendre un seul langage, Maude, et pourra programmer son application tout en bénéficiant d'outils de vérification avancés directement dans son environnement de travail.

A la fin, du fait que le concept de module (modules fonctionnels, modules orientés objet, et modules orientés objet temporisés) est à la base des spécifications formelles Maude et RT-Maude, nous présentons dans le Tableau 3.1 ci-dessous les principales notations et opérateurs utilisés pour décrire comment un module est formaté et comment lire son schéma.

Tableau 3.1. Notations et opérateurs utilisés dans les modules RT-Maude.

	Notations/Opérateurs	Description
Importer Module	Protecting < ModuleName > .	A utiliser Seulement sans modifier
	Extending < ModuleName > .	Étendre seulement les constructeurs
	Including < ModuleName > .	Modification des comportements des TAD et des règles de déduction
Type Abstrait de Données (TAD)	Sorts .. *** sorts Action Plan .	Donnez des noms pour les types de données.
	Subsorts .. ***subsort Action < Plan .	Imposer un ordre sur les Sorts
Opérateur	op <opname> : <sort-1> ... <sort-k> → <sort> [<operatorattributes>] .	Défini sur les sortes (types).
Equation	eq [<Label>] : <Term-1> = <Term-2> .	Fournir à l'interpreteur de Maude certaines règles pour simplifier une expression
Equation Conditionnelle	ceq [<Label>] : <Term-1> = <Term-2> if <Cond-1> ∧ ... ∧ <Cond-k> .	Exécuter une réduction que si son Cond-i est réduite à true.
Règle de Réécriture	rl [<Label>] : <Term-1> => <Term-2> . crl [<Label>] : <Term-1> => <Term-2> if <Cond-1> ∧ ... ∧ <Cond-k> .	Spécifier comment le système pourrait changer au fil du temps. Cond-i est une combinaison d'expressions booléennes, de tests d'égalité,...
Objet et classe	<O: C / att1 :val1, ..., attn :valn>	O est l'identifiant d'objet du type Oid C est la classe de l'objet att1 à attn sont les attributs de l'objet val1 à valn sont leurs valeurs actuelles
Message	op SendMessage : Message -> Msg .	Les messages sont définis comme des termes de type Msg.
Variable	var A : Agent .	La déclaration d'une variable A de type agent.
Règle Temporisé (Tick)	Inconditionnelle: rl [l] : {t} => {t'} in time μ.	Modélise l'écoulement du temps dans le système. L'état global du système doit toujours avoir la forme {t}, afin de garantir que le temps avance uniformément dans toutes les parties du système.
	Conditionnelle: crl [l] : {t} => {t'} in time μ if cond .	
Module Fonctionnel	(fmod <ModuleName> is <imports> *** réutilisation, modularité < sorts> *** types de données et sous-typage <ops> *** noms / et arités d'opérations < eqs> *** comment calculer les fonctions endfm)	Description de la structure et des propriétés des états du système (et TAD).
Module Système	(mod < ModuleName > is *** partie fonctionnelle <imports> *** réutilisation, modularité < sorts> *** types de données et sous-typage <ops> *** noms / et arités d'opérations < eqs> *** comment calculer les fonctions <rules> endm)	Définit un ensemble de calculs sur les TAD spécifiés par la partie fonctionnelle. Contient également des règles de réécriture.
Module Orienté-Objet	(omod < ModuleName > is < DeclarationsAndStatements > endom) class C att1 : s1, ... , attn : sn . subclass C < C1 ... Cn .	Fournir les bases syntaxiques pour les théories de réécriture orientées objet. Déclarer les classes. Permettre la déclaration de sous-classes.
Module Temporisé	{_} : System -> GlobalSystem	Importe automatiquement les types System et GlobalSystem ainsi que l'opérateur {_} .
Module Temporisé Orienté-Objet	(tomod < ModuleName > is < DeclarationsAndStatements > endtom)	Prend en charge la spécification orientée objet des systèmes temps réel.

4. Conclusion

La formalisation de spécifications permet d'avoir à disposition une représentation interprétable et vérifiable automatiquement par une machine, de sorte à faire apparaître les ambiguïtés, omissions et contradictions liées à la prise en compte de l'ensemble des exigences spécifiées d'un même système. Dans ce contexte, le passage de spécifications non formelle ou semi-formelle à des spécifications formelles vérifiables suscite un intérêt de plus en plus grand. Dans la littérature, il est entendu qu'un passage direct de spécifications informelles à des spécifications formelles n'est pas envisageable. Il se fait en général grâce à une intervention humaine, via une représentation intermédiaire pivot qui permettrait de réduire l'écart entre les deux types de spécifications.

Notre approche de formalisation est basée sur l'exploitation du potentiel de représentation et de formalisation du langage RT-Maude. Ce dernier est un langage très puissant et très expressif. Il offre un outil pour la spécification formelle, la simulation et l'analyse formelle des systèmes temps réel. Il met en évidence la facilité et la généralité de spécification, y compris un support pour les systèmes temps réel orientés objet qui peuvent être distribués, et où le nombre d'objets et de messages peut changer dynamiquement. Real Time Maude constitue à notre avis un candidat parfait pour la translation des diagrammes "AUML" vers une notation formelle. Cette combinaison d'AUML avec RT-Maude permettra de valider formellement les modèles AUML développés.

Donc, le but de l'étude de ces langages est de pouvoir les utiliser afin de proposer un cadre formel au développement des systèmes multi-agents temps réel, c'est à dire de formaliser les principes et concepts organisationnels exprimés à partir de modèles organisationnels tels que AGR et Moise+. Nous nous appuierons d'une part sur le langage AUML dans l'étape de modélisation. Et d'autre part, nous utiliserons le langage RT-Maude pour fixer une sémantique formelle temporelle utile pour la validation. Cette approche sera détaillée dans les chapitres suivants.

Chapitre 4

Un Framework Formel pour la Spécification des SMA centré Organisation

Introduction	87
1. Formalisation du Modèle Agent-Group-Rôle	89
1.1. Aperçu du Framework Formel	89
1.2. Architecture du Framework Formel	91
2. Etude de Cas: Gestion de chaîne logistique	95
2.1. Description	95
2.2. Modèle organisationnel d'agent pour la gestion de chaîne logistique	97
2.3. Modélisation AUML de la gestion de chaîne logistique	98
2.4. Application du Processus de Translation	100
2.5. Validation de la description générée	102
3. Discussion	104
4. Conclusion	105

Introduction

L'adoption d'une approche centrée sur l'organisation pour développer un système multi-agent (SMA) repose sur la définition d'un ensemble de contraintes qu'un groupe d'agents respecte pour atteindre facilement leurs objectifs sociaux [Lem98]. Cet ensemble de contraintes est généralement décrit comme des modèles organisationnels. En ayant un modèle d'organisation sous-jacent, le système multi-agent peut assurer un certain niveau de rendement et d'efficacité [Gas01], puisque le modèle établit des moyens pour coordonner et contrôler mieux le comportement des agents [Mur14a].

Dans [Lao13], plusieurs méthodes de modélisation de système multi-agent orienté organisation ont été analysées telles que Agent-Group-Role (AGR) [Fer03a], Model of Organization for multi-agent SystEms (MOISE+), INGENIAS [Pav05], Gaia [Woo00b] et d'autres. Ils ont modifié la construction de système multi-agents en donnant naissance à de nouveaux concepts organisationnels. Cependant, ils n'offrent que des descriptions informelles ou semi-formelles pour représenter les exigences organisationnelles du système multi-agent. Pour cette raison, l'approche proposée offre une représentation conjointe des exigences organisationnelles tout en profitant des avantages des approches semi-formelles et formelles. AGR est l'un des modèles organisationnels les plus populaires et les plus utilisés. Il a été décrit de manière intuitive et informelle, à travers la définition de ses principales caractéristiques en langage naturel. Cependant, ce manque de description rigoureuse peut facilement conduire à des imprécisions et des malentendus qui pourraient entraver l'utilisation et l'analyse du modèle, et par conséquent le développement d'un système multi-agent correct [Tro11].

L'objectif principal de ce travail est la formalisation. Ainsi, une approche basée sur la logique de réécriture est décrite pour prototyper le modèle AGR pour l'organisation de systèmes multi-agents afin de pouvoir raisonner, d'une part, sur le modèle organisationnel et d'autre part sur sa sémantique générale. Cette approche formelle permet la validation et la vérification et est présentée à travers une étude de cas. Tout d'abord, une spécification formelle du modèle AGR est donnée ensuite elle est affinée pour une application spécifique qui est la gestion de chaîne logistique. L'approche de spécification est basée sur l'utilisation de la logique de réécriture [Mes92], à travers l'utilisation du langage Maude [Cla07b] conjointement avec le formalisme graphique Agent UML afin de spécifier formellement un système basé sur le modèle organisationnel Agent-Groupe-Rôle.

Notre motivation vient du fait qu'actuellement peu de travaux appliqués la logique de réécriture aux modèles organisationnels de SMA et en particulier au modèle Agent-Groupe-Rôle. Par conséquent, Maude n'a pas été largement utilisé dans la communauté d'organisation de système multi-agents, et surtout pas dans le domaine des modèles d'organisation d'agents. Ainsi, l'approche que nous proposons se distingue des autres travaux en traitant explicitement de la formalisation d'un modèle organisationnel avec le langage Maude. Néanmoins, certains travaux [Ast08, Ast09, Das08a, Van06] sont fondés sur l'utilisation de la logique de réécriture et de Maude dans le contexte des langages de programmation d'agents à caractère organisationnel. Plus de détails sur ces travaux sont donnés ci-après. Pour commencer, on trouve que les auteurs de [Ast09] considèrent que le langage de programmation offre des constructions basées sur des concepts sociaux et organisationnels. Toutefois, selon le mécanisme d'ordonnancement de ces constructions, différentes sémantiques opérationnelles peuvent être définies. Ils montrent dans leur travail

comment une telle sémantique opérationnelle peut être prototypée dans le langage Maude. Selon eux, le prototypage par le biais de la réécriture est intéressant car il leur permet à la fois de concevoir et d'expérimenter avec les définitions du langage. Pour illustrer cela, les auteurs définissent des propriétés particulières (telles que enforcement et regimentation) liées aux artefacts de coordination qu'ils vérifient ensuite avec le model-checker de Maude qui est basé sur la logique temporel linéaire. Donc, ils décrivent un framework basé sur la réécriture pour le prototypage de langage pour les SMA normatifs dans le but est la vérification.

Ensuite, un autre travail très similaire à l'utilisation de Maude dans le domaine des SMA normatifs est présenté dans [Ast08]. Les auteurs présentent un langage de programmation conçu pour faciliter la mise en œuvre d'artefacts organisationnels basé sur les normes. La sémantique opérationnelle du langage est prototypée dans Maude et les propriétés des artefacts de coordination sont vérifiées aussi avec Maude. S'ajoute à cela le travail de [Das08a] où l'auteur se concentre sur le domaine des artefacts de coordination basés sur l'organisation. Il considère qu'une proposition spécifique pour la conception et le développement de mécanismes de coordination exogènes pour les SMA est l'introduction d'un langage de programmation basé sur l'organisation [Das08b, Das09]. Ce langage de programmation proposé est conçu pour faciliter la mise en œuvre d'artefacts de coordination basés sur l'organisation en termes de normes et de sanctions. Ces artefacts, également appelés «organisations multi-agents», se réfèrent aux normes comme un moyen pour signaler quand des violations ont lieu et aux sanctions comme un moyen de répondre en cas de violations. Donc, un artefact basé sur les normes observe les actions effectuées par les agents individuels, détermine les violations causées par l'exécution des actions, et impose éventuellement des sanctions. La sémantique opérationnelle de ce langage de programmation permet de le prototyper facilement dans Maude [Cla07a]. L'implémentation Maude de l'interpréteur de ce langage de coordination basé sur l'organisation [Ast08] permet de vérifier les programmes basés sur l'organisation, et afin de raisonner sur les programmes de coordination basés sur l'organisation, une logique est conçue pour raisonner sur ces programmes [Das08b].

Enfin, dans [Van06] les auteurs présentent une implémentation de (une version simplifiée de) langage de programmation d'agent cognitif appelé 3APL en utilisant le langage de réécriture Maude. Selon eux, Maude s'est avéré approprié à la fois comme framework logique dans lequel de nombreuses autres logiques peuvent être représentées, et comme framework sémantique, à travers lequel des langages de programmation avec une sémantique opérationnelle peuvent être implémentés de manière rigoureuse. Ils explorent l'utilisation de Maude dans le contexte des langages de programmation d'agents, dans ce sens le fait que les langages tels que 3APL ont à la fois une composante logique et une composante sémantique, rend Maude très bien adapté pour leurs prototypage. De plus, ils montrent que le cycle de raisonnement du méta-niveau ou le cycle de délibération de 3APL peut être implémenté très naturellement dans Maude vu que ce dernier est réflexif. De plus, bien qu'ils aient implémenté une version simplifiée de 3APL, les auteurs soutiennent que Maude est très bien adapté pour implémenter diverses extensions de cette version. Un avantage important de Maude, outre le fait qu'il est bien adapté pour le prototypage des langages de programmation d'agents, est qu'il peut être utilisé pour la vérification car il possède un vérificateur de modèle (model-checker) basé sur la logique temporel linéaire.

En résumé, Maude prend en charge la définition de syntaxe par l'utilisateur et, par conséquent, il offre des prototypes d'analyseurs gratuitement. Un autre avantage de l'utilisation de Maude est qu'il fournit un support d'outils pour des possibilités intéressantes telles que la simulation de modèle, l'analyse d'accessibilité et la vérification de modèle (model checking) [Cla07b]. De plus, du fait que les spécifications de la logique de réécriture sont exécutables, le framework formel proposé peut être utilisé comme un prototype, ce qui permet de le simuler afin de pouvoir fournir un premier retour (feedback) sur l'acceptation de la solution. Maude [Tro11] offre différentes possibilités pour réaliser la simulation, y compris l'exécution étape par étape, plusieurs stratégies d'exécution, etc. Bien que ce travail ne se concentre pas sur la vérification de modèle, le fait que Maude offre des moyens de vérification est une motivation importante pour cet effort de formalisation du modèle organisationnel AGR dans Maude.

Ce chapitre est structuré comme suit: La section 1 décrit comment le modèle d'organisation AGR peut être spécifié formellement dans Maude lors de l'utilisation du processus de translation. La section 2 détaille l'application du framework proposé dans la section précédente pour la modélisation et la formalisation d'une étude de cas portant sur un système de gestion de chaîne logistique. La section 3 donne un bref aperçu des principaux travaux connexes. Et Enfin, certaines conclusions et activités de recherche à suivre sont décrites à la section 4.

1. Formalisation du Modèle Agent-Group-Rôle

1.1. Aperçu du Framework Formel

Bien qu'il existe de nombreux modèles intéressants pour les systèmes multi-agents, ils sont généralement définis de manière informelle. Par conséquent, les concepteurs de SMA ont été incapables d'exploiter pleinement leurs points communs et de les spécialiser ou de les réutiliser pour des problèmes spécifiques. Afin d'utiliser ces modèles et faciliter leur réutilisation, une approche formelle est proposée basée sur des concepts organisationnels. La description formelle est le résultat de la composition des formalismes Maude et Agent UML. La sémantique de ces multi-formalismes est définie par une logique de réécriture. Cette sémantique opérationnelle permet la validation et la vérification des spécifications. L'approche proposée est présentée à travers la spécification de l'étude de cas gestion de chaîne logistique (en anglais Supply Chain Management) qui a été utilisée pour concevoir des systèmes multi-agents organisationnels basés sur le modèle Agent-Groupe-Rôle. Selon [Hil04] et d'autres travaux prouvent que la spécification formelle peut être utilisée pour décrire des concepts de modèles qui peuvent être affinés pour répondre à des besoins particuliers du système.

Le framework proposé adopte une approche basé sur la logique de réécriture pour décrire les contraintes organisationnelles liées au modèle AGR qui a été présenté au chapitre 2, à travers la définition d'un ensemble de modules Maude (voir Figure 4.1). L'un des avantages d'une telle formalisation est qu'elle soit systématique. Afin de décrire le modèle AGR en utilisant Maude, la représentation formelle du méta-modèle introduite dans [Lao14] sera suivie.

Étant donné qu'il existe deux dimensions organisationnelles dans Agent-Group-Role selon l'étude présentée dans [Cou09]. Premièrement, les notions de groupes et de rôles

composent la dimension structurelle. De plus, la définition des interactions entre agents forme la dimension dialogique. La translation avec le langage basé sur la logique de réécriture formalisera les concepts d'AGR en fonction de leur dimension correspondante, de la même façon que dans [Lao14].

Ainsi, pour décrire le modèle AGR en termes de la logique de réécriture, il faut d'abord définir la syntaxe du modèle, puis décrire la sémantique correspondante en termes du langage Maude. Ce dernier est choisi comme langage de spécification formel vu qu'il est bien adapté pour le prototypage du modèle organisationnel AGR et qu'il est fourni avec un vérificateur de modèle basé sur la logique temporelle linéaire. En outre, le mécanisme d'importation de modules dans Maude aide à construire des applications modulaires à partir de petits modules, ce qui facilite le débogage, la maintenance ou l'extension. Dans ce qui suit, les constructeurs syntaxiques de base qui sont nécessaires pour comprendre comment la logique de réécriture est utilisée pour formaliser le modèle AGR sont brièvement présentés.

Par conséquent, le modèle AGR pour l'organisation de système multi-agent est prototypé en plusieurs modules: les neuf premiers (module 1, jusqu'au module 9), définissent la syntaxe du modèle, tandis que les derniers (module 10; 12 et 13), implémentent la sémantique. Les lignes figurant dans *la définition 1* ci-dessous sont extraites des modules syntaxiques (section 2.2) pour décrire la dimension structurelle. Ils représentent la déclaration et les types dont dépend le modèle AGR:

Definition 1: Description de la structure des états du système

```
sorts State NameState Action Condition .
sorts GroupList Group AgentRole Message .
class Agent | CurrentState : State, PlayRole : AgentRole, GrList : GroupList,
      AcqList : AcquaintanceList, MBox : MailBox .
op SendMessage : Message -> Msg .
op ReceiveMessage : Message -> Msg .
op TargetState : State Condition -> State .
```

D'autre part, la partie sémantique est implémentée par des règles de réécriture liées à la dimension dialogique d'AGR qui aident à décrire comment l'état de l'organisation peut changer avec le temps. Pour des raisons de limitations d'espace, on ne présente qu'une règle de réécriture conditionnelle dans *la définition 2*.

Definition 2: Spécifier comment le système peut changer

```
crl [Send-Cli] : Event(A, S, Cond)
  < A : Client | CurrentState : S, AcqList : ACL, Order : P, Deadline : D, Cost : Pr >
=>
  < A : Client | CurrentState : TargetState(S,Cond), AcqList : TailA(ACL), Order : P,
    Deadline : D, Cost : Pr >
  SendMessage(A : ActionToAccomplish(S, Cond) : HeadA(ACL)) Event(HeadA(ACL),
CorrespondingState(S), CorrespondingCondition(ActionToAccomplish(S, Cond)))
if (IsSendingAction(ActionToAccomplish(S, Cond)) == true) .
```

Plus de détails sur les différents modules syntaxiques et sémantiques définis à la Figure 4.1 sont donnés ci-dessous.

1.2. Architecture du Framework Formel

Le framework proposé (Figure 4.1) est composé de plusieurs modules Maude: modules fonctionnels (fmod), modules orientés objet (omod) et modules orientés objet temporisés (tomod).

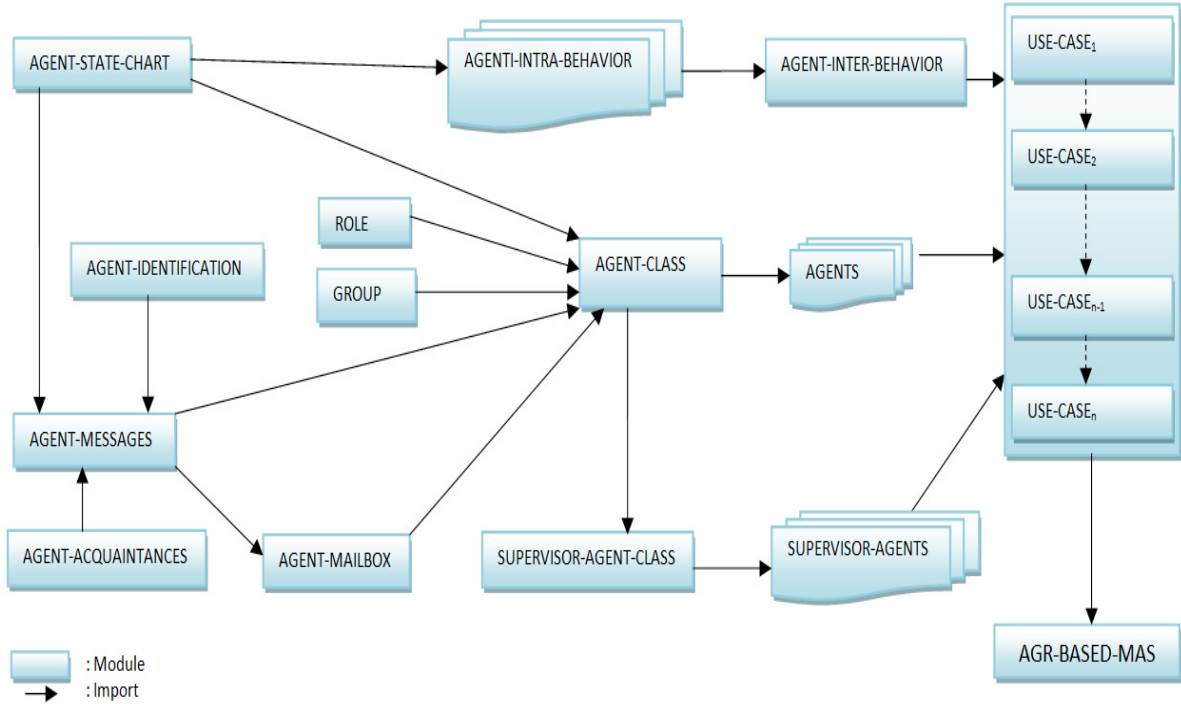


Figure 4.1. Les Modules du Framework Formel.

Les principaux modules du framework sont décrits ci-dessous.

Commençant par le module fonctionnel 1 qui définit les différents états de l'agent et comment passer d'un état à un autre par des actions. En effet, les agents se comportent en fonction d'actions dans l'environnement ou d'actions internes. Chaque fois que des actions se produisent, les agents passent d'un état à un autre.

Module 1: AGENT-STATE-CHART

```
(fmod AGENT-STATE-CHART is
  sorts State NameState .    sort Action .    sort Condition .
  op AgentState : NameState -> State .    op IsInternalAction : Action -> Bool .
  op TargetState : State Condition -> State .    op ActionToAccomplish : State Condition ->
  Action .
  ...
endfm)
```

Dans le module orienté objet 2, le concept de base de "Groupe" est donné pour savoir les groupes auxquels les agents appartiennent.

Module 2: GROUP

(omod GROUP is

including STRING .

sorts GroupList Group .

subsort Group < GroupList .

subsort String < GroupList .

op _:_ : Group GroupList -> GroupList . **op** EmptyGroupList : -> GroupList .

var Gr : Group . **var** GrL : GroupList .

eq Gr : EmptyGroupList = Gr .

endom)

Le module fonctionnel 3 est utilisé pour définir le concept de base du modèle Agent-Groupe-Rôle, qui est "Rôle" définissant ainsi les rôles que les agents peuvent jouer dans les groupes.

Module 3: ROLE

(fmod ROLE is

sort AgentRole .

endfm)

Le mécanisme utilisé pour l'identification des agents est illustré dans le module orienté objet 4.

Module 4: AGENT-IDENTIFICATION

(omod AGENT-IDENTIFICATION is

sort Identifier .

subsort Oid < Identifier .

endom)

Pour la description de la structure des messages proposés et les opérations de communication (envoi / réception) utilisées par les agents pour échanger des messages, le Module orienté objet 5 est défini, qui importe les Modules 1, 4 et 6 respectivement.

Module 5: AGENT-MESSAGES

(omod AGENT-MESSAGES is

protecting AGENT-STATE-CHART .

protecting AGENT-IDENTIFICATION .

protecting AGENT-ACQUAINTANCES .

sort Message . **sort** Content .

subsort Action < Content . **subsort** Acquaintance < Identifier .

op SendMessage : Message -> Msg .

op ReceiveMessage : Message -> Msg .

op _:_ : Identifier Content Identifier -> Message . **op** GetIdent : Message -> Identifier .

vars A A1 : Identifier . **var** Cont : Content .

eq GetIdent (A : Cont : A1) = A .

endom)

Pour faciliter la communication, les agents sont généralement dotés d'une boîte aux lettres contenant les messages reçus des autres agents ainsi que la liste de leurs connaissances. Dans ce cas, le module fonctionnel 6 et le module 7 qui importe le module 5 sont proposés pour gérer respectivement les listes de connaissances des agents et leurs boîtes aux lettres. Par liste de connaissances nous voulons dire les agents avec lesquels un agent est amené à communiquer.

Module 6: AGENT-ACQUAINTANCES

```
(fmod AGENT-ACQUAINTANCES is
  including STRING .
  sorts AcquaintanceList Acquaintance .
  subsort Acquaintance < AcquaintanceList .
  subsort String < Acquaintance .
  op HeadA : AcquaintanceList -> Acquaintance .
  op ErrorAcquaintance : -> Acquaintance .
  op TailA : AcquaintanceList -> AcquaintanceList .
  op _:_ : Acquaintance AcquaintanceList -> AcquaintanceList .
  ...
  var Ac : Acquaintance .    vars AcL : AcquaintanceList .
  eq Ac : EmptyAcquaintanceList = Ac .
  eq IsEmptyAcquaintanceList(Ac : AcL) = false .
  eq IsEmptyAcquaintanceList(EmptyAcquaintanceList) = true .
  ...
endfm)
```

Module 7: AGENT-MAIBOX

```
(fmod AGENT-MAILBOX is
  protecting AGENT-MESSAGES .
  sort MailBox .
  subsort Message < MailBox .
  op FrontMBox : MailBox -> Message .    op RemoveMBox : MailBox -> MailBox .
  op _ : Message MailBox -> MailBox .    op InsertInMBox : Message MailBox -> MailBox .
  ...
  var M : Message .    var MB : MailBox .
  eq RemoveMBox(EmptyMailBox) = ErrorMailBox .
  eq FrontMBox(M MB) = M .
  eq M EmptyMailBox = M .
  eq InsertInMBox(M, MB) = if MB == EmptyMailBox then M else M MB fi .
  ...
endfm)
```

Dans le module orienté objet 8 appelé AGENT-CLASS, la structure de la classe de base des agents est définie. Cette classe (ligne [1]) a comme attributs: CurrentState, PlayRole, GrList, AcqList et MBox, pour contenir dans cet ordre: son état actuel, le rôle joué par l'agent, sa liste de groupes, ses connaissances et sa boîte aux lettres. Ce module importe les modules 1, 2, 3, 5 et 7.

Module 8: AGENT-CLASS

```

(omod AGENT-CLASS is
  protecting AGENT-STATE-CHART .
  protecting GROUP .           protecting ROLE .
  protecting AGENT-MESSAGES .  protecting AGENT-MAILBOX .
  class Agent | CurrentState : State, PlayRole : AgentRole, GrList : GroupList,
                    AcqList : AcquaintanceList, MBox : MailBox .           ---[1]
endom)

```

Pour décrire le superviseur de chaque groupe d'agents, la classe SuperAgent est définie avec l'attribut Responsibility (ligne [1]) dans le module orienté objet 9 qui importe le module 8, en tant que sous-classe de la classe Agent (ligne [2]).

Module 9: SUPERVISOR-AGENT-CLASS

```

(omod SUPERVISOR-AGENT-CLASS is
  extending AGENT-CLASS .
  sort Manager .    subsort String < Manager .
  class SuperAgent | Responsibility : Manager .           ---[1]
  subclass SuperAgent < Agent .                         ---[2]
endom)

```

Les modules fonctionnels 10 nommés AGENTI-INTRA-BEHAVIOR importent le Module 1 et sont utilisés pour décrire les comportements des agents du système à travers la définition des différents états, conditions et actions de chaque agent. Cependant, Afin de respecter les interactions entre ces agents, comme illustré dans les diagrammes de protocole AUML, les connexions entre eux sont effectuées via le module fonctionnel 11. Ce dernier importe les différents modules 10.

Module 10: AGENTI-INTRA-BEHAVIOR

```

(fmod AGENTI-INTRA-BEHAVIOR is
  protecting AGENT-STATE-CHART .
  ***** User Part
  ... endfm)

```

Module 11: AGENT-INTER-BEHAVIORS

```

(fmod AGENT-INTER-BEHAVIORS is
  protecting AGENTI-INTRA-BEHAVIOR .
  ...
  op CorrespondingState : State -> State .  op CorrespondingCondition : Action -> Condition .
  ***** User Part
  ...
endfm)

```

A Chaque cas d'utilisation défini dans les différents diagrammes de protocole est associé un module temporel orienté objet 12, qui porte le même nom que le cas d'utilisation correspondant. Ces modules incluent différentes règles de réécriture montrant les scénarios d'interaction entre les agents. Ces règles peuvent être instantanées ou temporelles (Tick), conditionnelles ou inconditionnelles.

Module 12: USE-CASE_i

```

(tomod USE-CASEi is
  including AGENT-CLASS . including SUPERVISOR-AGENT-CLASS.
  including AGENT-INTER-BEHAVIORS.
  rl [1] : Configuration1 => Configuration2.
  ...
  rl [m] : Configuration2m-1 => Configuration2m.
endtom)

```

Une fois générée, tous les modules USE-CASE_i sont importés dans le module orienté objet temporisé 13 qui décrit toutes les exigences organisationnelles du système.

Module 13: AGR-BASED-MAS

```

(tomod AGR-BASED-MAS is
  including USE-CASE1.
  ...
  including USE-CASEm.
endtom)

```

La forme générale des règles temporelles (Tick) utilisées pour assurer la progression du temps est donnée dans la définition 3, où le message Timer est utilisé pour modifier l'horloge de l'Agent Temps Réel définie par l'attribut Clock (ligne [1]).

Definition 3: Spécification de la progression du temps dans le système.

```

cr1 [tick] :{  Timer(Timeout)                --[1]
               < A : Agent | PlayRole: Initiator, Clock : T >  REST:Configuration } --[1]
               => { Timer(Timeout monus 1)  < A : Agent | Clock : T plus 1 >  REST:Configuration }
               in time 1 if (Timeout > zero).

```

2. Etude de Cas: Gestion de chaîne logistique

2.1. Description

Le développement de logiciels est un processus complexe, avec le risque de faire des erreurs dans chacune des différentes étapes. L'utilisation de la notation formelle dès la phase d'analyse réduira grandement ce risque.

Dans le contexte économique actuel, de nombreuses recherches sont actuellement menées dans le domaine des systèmes multi-agents concernant la modélisation et l'optimisation des chaînes logistiques [Naw10]. D'après [Hug02a], la Gestion de Chaîne Logistique modélise la production dans les entreprises depuis l'envoi de la commande jusqu'à la livraison. Cette application implique de recevoir une commande ainsi que de générer un plan de production de cette commande, de négocier le prix et le délai, de modifier le plan si certaines contraintes sont violées, de fabriquer le produit et enfin de le livrer. La gestion de chaîne logistique considère trois types d'acteurs (voir Figure 4.2):

- 1) Les clients qui placent, modifient et suppriment des commandes,
- 2) L'entreprise avec les relations clients et la production,
- 3) Les fournisseurs de matières premières qui peuvent être d'autres sociétés.

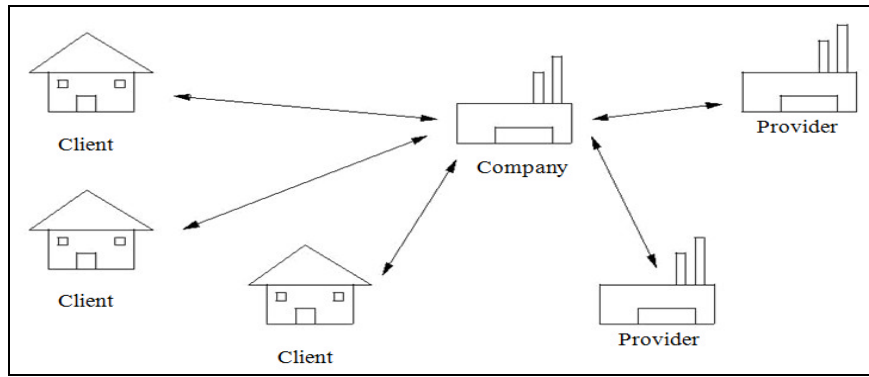


Figure 4.2. La gestion de chaîne logistique [Hug02a].

Dans [Naw10], les auteurs ont indiqué que dans la littérature, divers modèles de systèmes multi-agents existent pour la gestion de chaîne logistique et la modélisation des éléments de chaîne logistique, en particulier pour les entreprises et leurs différents départements. Un aperçu plus large des différentes solutions peut être trouvé dans [Moy06, She99]. Des exemples de systèmes multi-agents pour la gestion de chaîne logistique sont présentés. Ils sont ensuite décrits pour considérer le problème étudié, l'approche appliquée, le nombre d'agents et les rôles joués par eux. À partir de cet aperçu, on peut remarquer que les agents sont appliqués à la gestion de chaîne logistique, à sa construction et à sa coordination. Ils sont utilisés pour augmenter la flexibilité, pour un meilleur choix de partenaires commerciaux et pour simuler la coopération globale qui émerge de la concurrence locale.

Afin de valider l'approche proposée, une organisation simple de gestion chaîne logistique construite selon le modèle organisationnel Agent-Groupe-Rôle pour le développement de SMA a été instanciée, composée de deux groupes d'agents. Le premier groupe est composé du **client** qui passe, modifie et supprime les commandes; ainsi que les agents order-Acquisition; et logistics. Le deuxième groupe contient quant lui les agents Scheduler; Dispatcher; Resource; Transporter; et provider. De plus, pour chaque groupe, un agent superviseur est défini comme représenté sur la Figure 4.3, son rôle est de coordonner les interactions de groupe d'agents.

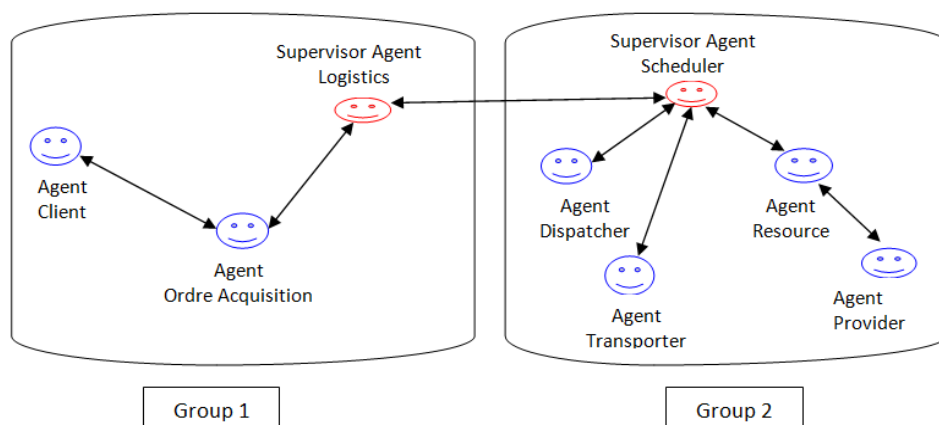


Figure 4.3. Structure du système multi-agents de la gestion de chaîne logistique.

2.2. Modèle organisationnel d'agent pour la gestion de chaîne logistique

En utilisant les concepts et éléments du modèle Agent-Groupe-Rôle, une description du système multi-agents pour la modélisation et la formalisation de la gestion de chaîne logistique a été effectuée. La décomposition du SMA pour la gestion de chaîne logistique basée sur les concepts d'AGR est représentée par plusieurs agents regroupés en deux groupes, comme illustré à la Figure 4.4:

Le Groupe 1 est composé des agents suivants:

- L'Agent Client représente le client final, commandant des produits fabriqués par l'entreprise, modifiant ou supprimant des commandes. Il coopère avec l'agent order-acquisition.
- L'Agent Order-acquisition reçoit les commandes de l'agent client et négocie le prix et le délai avec l'agent Logistics. Il est responsable de la connexion entre les clients et l'entreprise. Il informe les clients dès que les livraisons sont reportées ou livrées.
- L'Agent Logistics est le Superviseur de ce groupe pour gérer les autres agents, négocie le délai et le prix avec l'agent order-acquisition. Dès que les commandes sont acceptées, il demande à l'agent Scheduler de générer un plan pour cette commande. Ce plan est envoyé aux agents Transporter, Resource et Dispatcher. Il demande un nouveau plan dès que certaines modifications apparaissent soit si le client souhaite modifier la commande, ou bien des retards surviennent sur la production. Tous les agents faisant parties de l'entreprise, sont subordonnés à l'agent logistics qui détermine les stratégies et modifie les fonctions d'objectif des agents.

De plus, le Groupe 2 contient cinq agents, à savoir:

- L'Agent Scheduler est le Superviseur de ce deuxième groupe, il génère un plan en fonction de l'ordre, du délai et de l'allocation des employés et des ressources.
- L'Agent Transporter est responsable de la livraison des matières premières pour la production et la livraison du produit final au client.
- L'Agent Resource gère l'allocation des matériaux afin de réaliser la commande. Si le matériel n'est pas disponible, il passe une commande à l'agent Provider.
- L'Agent Dispatcher gère les agents employés. Il alloue du travail aux agents employés, et informe si certains problèmes surviennent provoquant un retard dans la livraison.
- L'Agent Provider fournit quant à lui les matières premières si l'entreprise n'a pas assez de matériaux pour la commande. En fait, il peut s'agir d'un agent order-acquisition d'une autre société.

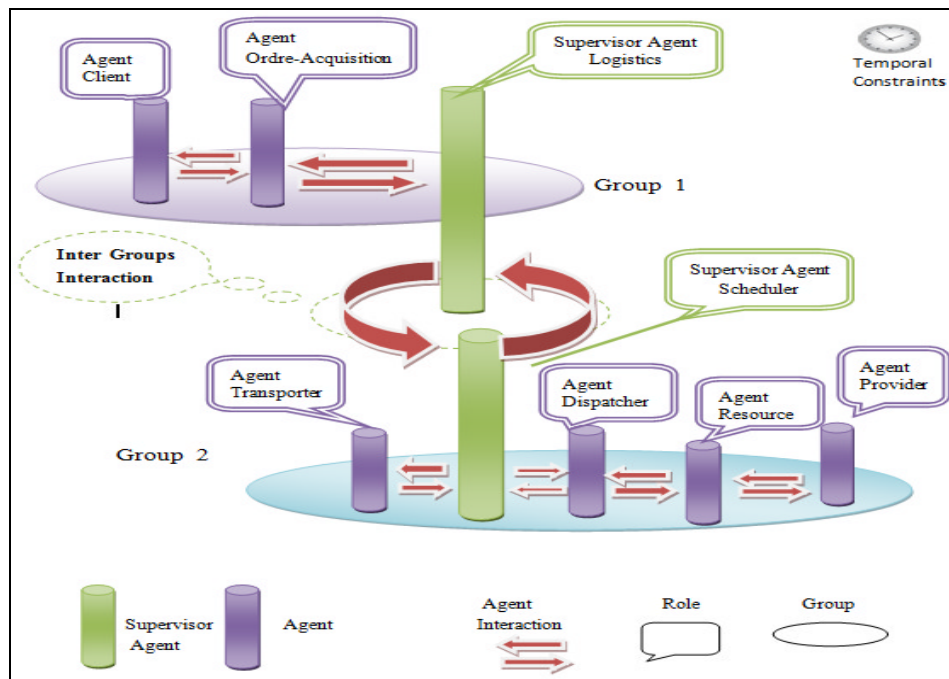


Figure 4.4. Décomposition agents selon AGR de la gestion de chaîne logistique.

2.3. Modélisation AUML de la gestion de chaîne logistique

2.3.1. Diagramme de Classe conceptuel de la gestion de chaîne logistique

Les diagrammes de classe [Hug02a] correspondent à l'architecture du système. Ces diagrammes semblent intéressants pour représenter les différents rôles d'agents et les relations entre eux. UML et par conséquent Agent UML permettent de représenter plusieurs niveaux d'abstraction lors de la conception des diagrammes de classe. En effet, il n'est parfois pas intéressant d'avoir une vue précise du système avec toutes les dépendances et tous les attributs. Les diagrammes de classe de haut niveau permettent de saisir le système dans son intégralité. Un diagramme de classe pour le niveau conceptuel de l'exemple de gestion de chaîne logistique en fonction du modèle organisationnel AGR est illustré à la Figure 4.5. Un package est un mécanisme standard dans Agent UML pour regrouper des éléments, dans la Figure 4.5, un package globale nommé Organization contient deux autres packages nommés Group1, et Group2 afin de regrouper tous les agents du système de gestion de chaîne logistique.

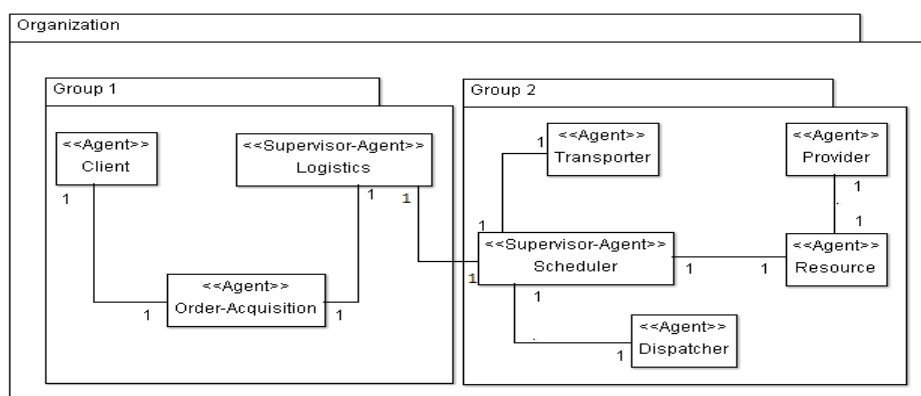


Figure 4.5. Diagramme de Classe conceptuel AUML de la gestion de chaîne logistique.

2.3.2. Diagramme de Cas d'utilisation de la gestion de chaîne logistique

Les diagrammes de cas d'utilisation représentent les cas d'utilisation, les acteurs, et les relations entre les deux. Le principal avantage de ce diagramme est de se concentrer sur le quoi et non sur le comment, c'est-à-dire sur le comportement du système et non sur la façon dont le système est implémenté. Comme illustré sur la Figure 4.6, les huit agents de la gestion de la chaîne logistique décrits ci-dessus interagissent avec les trois cas d'utilisation de l'organisation d'agents: Place-command, Modify-command, et Delete-command. Le cas d'utilisation Place-Command est lié au cas d'utilisation Modify-Command par la relation "Include" et au cas d'utilisation Delete-Command par la relation "Extend".

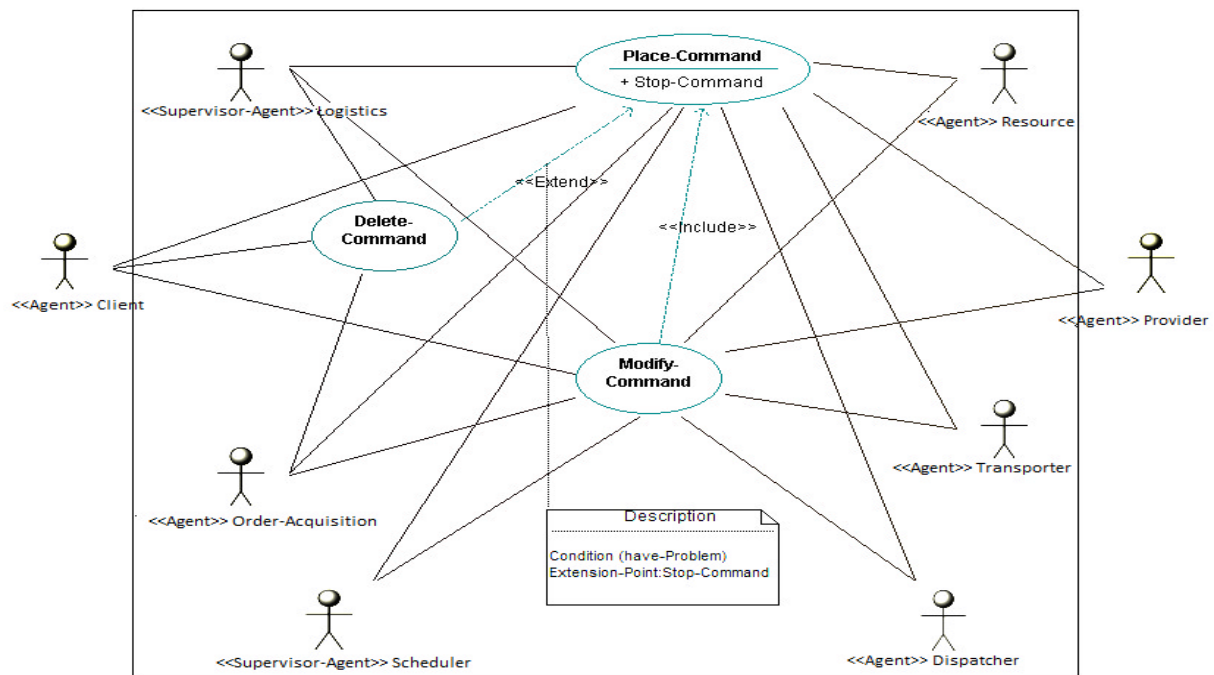


Figure 4.6. Diagrammes de cas d'utilisation de la gestion de chaîne logistique.

2.3.3. Diagramme de Protocole de la gestion de chaîne logistique

Les diagrammes de protocole [Hug02a] appelés diagrammes de séquence dans UML décrivent le flux de messages entre les agents. En effet, les messages ne vont pas d'un agent à un autre mais d'un rôle à un autre. L'avantage de cette approche est de réduire la taille du diagramme. Plusieurs travaux existent sur les diagrammes de protocole [Ode00, Bau00] parmi d'autres. Plusieurs interactions existent dans l'exemple de la gestion de chaîne logistique, donnant de nombreux diagrammes de protocole. Voici les listes de scénarios qui donnent des protocoles d'interaction:

- Commander un produit où tous les agents du SMA interviennent.
- Modifier une commande où tous les agents du SMA interviennent.
- Annuler une commande où interviennent les agents client, order-acquisition, et logistics.

Ici, la commande est centrée sur un produit. Ce protocole d'interaction est vraiment central dans le SMA où tous les agents sont utilisés. En effet, l'agent client passe une commande, l'agent order-acquisition reçoit la commande et négocie le prix et le délai avec les agents client et logistics. Ce dernier calcule le prix et le délai en fonction des informations provenant de

Module 14: SUPERVISOR-AGENTS

```
(tomod SUPERVISOR-AGENTS is
  extending SUPERVISOR-AGENT-CLASS .
  ***** User Part
    subclass Logistics < SuperAgent .
    subclass Scheduler < SuperAgent .
    class Logistics .      class Scheduler .
endtom)
```

Module 15: AGENTS

```
(omod AGENTS is
  extending AGENT-CLASS .
  ***** User Part
    including STRING .      including NAT .
    subclass Client < Agent .  subclass Order-Acquisition < Agent .
    subclass Transporter < Agent .  subclass Dispatcher < Agent .
    subclass Resource < Agent .  subclass Provider < Agent .

    class Client | Order : String, Deadline : Nat, Cost : Nat .
    class Provider | PriceConstraint : Nat, Cost : Nat .
    class Transporter | DelayConstraint : Nat, Deadline : Nat .
    class Dispatcher | DelayConstraint : Nat, Deadline : Nat .
    class Order-Acquisition .  class Resource .
endom)
```

Module 16: PLACE-COMMAND

```
(tomod PLACE-COMMAND is
  including NAT-TIME-DOMAIN .  including AGENTS .
  including SUPERVISOR-AGENTS .  including AGENT-INTER-BEHAVIORS .
  ***** User Part
    subsort String < Identifier .  ops Event GetEvent : Identifier State Condition -> Msg .
    op Execute : Action -> Msg .  ops IsAcceptedDelay IsAcceptedPrice : -> Msg .
    ops IsRefusedDelay IsRefusedPrice : -> Msg .
    op UpDateAcqList : MailBox -> AcquaintanceList .
    op SchedulePlan : -> Msg . op ModifyOrder : -> Msg .
    ...
    vars A A1 A2 A3 A4 A5 A6 A7 : Identifier .  vars S S1 S2 S3 S4 S5 : State .
    vars MB : MailBox .  vars ACL : AcquaintanceList .
    vars Cond Cond1 : Condition .  var Act : Action . var M : Message .
    eq UpDateAcqList(MB) = if MB == EmptyMailBox then EmptyAcquaintanceList
      else GetIdent(FrontMBox(MB)) : UpDateAcqList(RemoveMBox(MB))fi .
    -----| Sending (Agent Client) |-----
    crl [Send-Cli] : Event(A, S, Cond)
      < A : Client | CurrentState : S, AcqList : ACL, Order : P, Deadline : D, Cost : Pr >
      => < A : Client | CurrentState : TargetState(S,Cond), AcqList : TailA(ACL),
```

```

        Order : P, Deadline : D, Cost : Pr >
    SendMessage(A : ActionToAccomplish(S, Cond) : HeadA(ACL)) Event(HeadA(ACL),
        CorrespondingState(S), CorrespondingCondition(ActionToAccomplish(S, Cond)))
        if (IsSendingAction(ActionToAccomplish(S, Cond)) == true) .
-----| Receiving (Agent Client) |-----
cr1 [Receive-Cli1] : Event(A, S, Cond)
        SendMessage(A1 : Cont : A)
        < A : Client | CurrentState : S, AcqList : ACL >
        => < A : Client | CurrentState : TargetState(S, Cond), AcqList : ACL >
        if (IsReceivingAction(ActionToAccomplish(S, Cond)) == true) and (Cond ==
        ReceiveAccept) .
cr1 [Receive-Cli2] : Event(A, S, Cond)
        SendMessage(A1 : Cont : A)
        < A : Client | CurrentState : S, AcqList : ACL >
        => < A : Client | CurrentState : TargetState(S, Cond), AcqList : ACL >
        ModifyOrder
        if (IsReceivingAction(ActionToAccomplish(S, Cond)) == true) and (Cond ==
        ReceivePropose) .
        ... endtom)

```

Module 17: ROLE

```

(fmod ROLE is
    sort AgentRole .
    ***** User Part
    ops Client Order-Acquisition Logistics Scheduler Transporter Dispatcher Resource
        Provider : -> AgentRole .
endfm)

```

Module 18: AGR-BASED-MAS

```

(tomod AGR-BASED-MAS is
    including PLACE-COMMAND .
    including MODIFY-COMMAND .
    including DELETE-COMMAND .
endtom)

```

2.5. Validation de la description générée

Une fois que l'exemple de la gestion de chaîne logistique soit spécifié dans Maude en appliquant le processus de translation entre les diagrammes AUMML présentés dans la section 2.3 et le framework proposé, le résultat est une spécification basé sur la logique de réécriture des différents cas d'utilisation du système. Maude à l'avantage d'offrir un support d'outils pour des possibilités intéressantes telles que la simulation de modèle, l'analyse d'accessibilité et la vérification de modèle [Cla07b]. De plus, du fait que les descriptions de la logique de réécriture produites sont exécutables, elles peuvent être utilisées comme un prototype, ce qui

nous permet de la simuler. Selon [Tro11], Maude permet aussi de réaliser la simulation de diverses manières, y compris l'exécution pas-à-pas, plusieurs stratégies d'exécution, etc. Le résultat du processus de simulation est la configuration finale des objets obtenu après les étapes de réécriture, qui n'est qu'un modèle. Ainsi, la spécification formelle est utilisée conjointement avec l'outil Maude afin de générer des traces de simulation. Donc, la simulation est un outil très utile capable de fournir un premier retour sur l'acceptation d'une solution.

Pour valider l'exemple de la gestion de chaîne logistique, une configuration initiale est considérée où l'agent Client commence par passer un order (ou une commande) avec certaines contraintes sur le *délai* et le *prix* du produit. Ces contraintes vont être acceptées du fait qu'elles répondent aux exigences des autres agents qui composent le système de la gestion de chaîne logistique. La Figure 4.8 illustre le module orienté objet temporisé SUPPLY-CHAIN-MANAGEMENT, qui importe le module de base AGR-BASED-MAS du framework formel pour spécifier une configuration initiale du système. Il décrit les agents de l'exemple dans leurs états initiaux. Selon la façon dont les règles de Maude sont définies, le résultat de la réécriture illimitée d'une telle configuration est illustré par la Figure 4.9. Le résultat de cette configuration montre que tous les agents atteignent leurs états de succès ce qui explique que les contraintes imposées par l'agent client ont été acceptées par tous les agents du système.

```
(tomod SUPPLY-CHAIN-MANAGEMENT is
  extending AGR-BASED-MAS .

  op Initstate : -> GlobalSystem .

eq Initstate = { Event("client", AgentState(StartC), IsInitialized)
  < "client" : Client | PlayRole : Client, CurrentState : AgentState(StartC), GrList : "Group-1", AcqList : "Order-Acquisition",
    MBox : EmptyMailBox, Order : "Machine", Deadline : 15, Cost : 20 >

  < "Order-Acquisition" : Order-Acquisition | PlayRole : Order-Acquisition, CurrentState : AgentState(StartO), GrList : "Group-1",
    MBox : EmptyMailBox, AcqList : "logistics" >

  < "logistics" : Logistics | PlayRole : Logistics, Responsibility : "Manager1", CurrentState : AgentState(StartL), GrList : "Group-1",
    MBox : EmptyMailBox, AcqList : "scheduler" >

  < "scheduler" : Scheduler | PlayRole : Scheduler, Responsibility : "Manager2", CurrentState : AgentState(StartS), GrList : "Group-2",
    MBox : EmptyMailBox, AcqList : ("transporter" : ("dispatcher" : "resource")) >

  < "transporter" : Transporter | PlayRole : Transporter, CurrentState : AgentState(StartT), GrList : "Group-2", MBox : EmptyMailBox,
    AcqList : "scheduler", DelayConstraint : 10, Deadline : 15 >

  < "dispatcher" : Dispatcher | PlayRole : Dispatcher, CurrentState : AgentState(StartD), GrList : "Group-2", MBox : EmptyMailBox,
    AcqList : "scheduler", DelayConstraint : 10, Deadline : 15 >

  < "resource" : Resource | PlayRole : Resource, CurrentState : AgentState(StartR), GrList : "Group-2", MBox : EmptyMailBox,
    AcqList : "provider" >

  < "provider" : Provider | PlayRole : Provider, CurrentState : AgentState(StartP), GrList : "Group-2", AcqList : "resource",
    MBox : EmptyMailBox, PriceConstraint : 10, Cost : 20 >

  TimerOfNeg(50) TimerOfSched(30) Compter(0) } .

endtom)      (trew Initstate with no time limit .)
```

Figure 4.8. La Configuration initiale de l'exemple de gestion de chaîne logistique.

```

Elapsed time: 00:00:00.270

Result '[ClockedSystem]' :

{< "Order-Acquisition" : Order-Acquisition | AcqList : ("logistics" :
  "client"), CurrentState : AgentState(SuccessO), GrList : "Group-1", MBox : (("logistics" : SendAccept : "Order-Acquisition") "client" : Purchase :
  "Order-Acquisition"), PlayRole : Order-Acquisition >

< "client" : Client | AcqList : EmptyAcquaintanceList, Cost : 20, CurrentState : AgentState(SuccessC), Deadline : 15, GrList : "Group-1",
  MBox : EmptyMailBox, Order : "Machine", PlayRole : Client >

< "dispatcher" : Dispatcher | AcqList : EmptyAcquaintanceList, CurrentState : AgentState(SuccessD), Deadline : 15, DelayConstraint : 10,
  GrList : "Group-2", MBox : EmptyMailBox, PlayRole : Dispatcher >

< "logistics" : Logistics | AcqList : ("scheduler" : "Order-Acquisition"), CurrentState : AgentState(SuccessL), GrList : "Group-1",
  MBox : (("scheduler" : SendAccept : "logistics") "Order-Acquisition" : IsPossible : "logistics"), PlayRole : Logistics,
  Responsibility : "Manager1" >

< "provider" : Provider | AcqList : EmptyAcquaintanceList, Cost : 20, CurrentState : AgentState(SuccessP), GrList : "Group-2",
  MBox : EmptyMailBox, PlayRole : Provider, PriceConstraint : 10 >

< "resource" : Resource | AcqList : ("provider" : "scheduler"), CurrentState :
  AgentState(SuccessR), GrList : "Group-2", MBox : (("provider" :
  DecisionProcess : "resource") "scheduler" : RequestAbility : "resource"), PlayRole : Resource >

< "scheduler" : Scheduler | AcqList :
  EmptyAcquaintanceList, CurrentState : AgentState(SuccessS), GrList :
  "Group-2", MBox : "logistics" : RequestPlan : "scheduler", PlayRole :
  Scheduler, Responsibility : "Manager2" >

< "transporter" : Transporter | AcqList : EmptyAcquaintanceList, CurrentState : AgentState(SuccessT), Deadline : 15, DelayConstraint : 10,
  GrList : "Group-2", MBox : EmptyMailBox, PlayRole : Transporter > in time 80

```

Figure 4.9. Résultat d'exécution de la Configuration initiale de l'exemple de gestion de chaîne logistique.

3. Discussion

Dans les systèmes multi-agents centrés organisation, il existe la définition d'un ensemble de contraintes généralement décrites par des modèles organisationnels, tels que Agent-Group-Role [Fer03a], Model of Organization for multi-agent SystEms (Moisse) [Hüb02], et d'autres avec leurs infrastructures associées. L'adoption d'un modèle organisationnel sous-jacent peut fournir l'adaptabilité au système à l'exécution [Mur14b].

Le travail présenté dans ce chapitre est une intégration des résultats présentés dans [Lao13, Lao14] avec des extensions supplémentaires liées à la description du modèle Agent-Groupe-Rôle. En particulier, une approche similaire à celle présentée dans [Lao14] est suivie afin d'avoir une formalisation basée sur la logique de réécriture du modèle AGR offrant ainsi une sémantique formelle au système multi-agents. Par conséquent, un langage basé sur la logique réécriture doit être introduit en tant que langage de spécification pour décrire le processus de translation du modèle organisationnel.

L'idée principale est d'étudier la possibilité de traduire (transformer) le modèle organisationnel AGR vers une spécification formelle en langage Maude. Une proposition laissée ouverte dans [Lao14]. Au début le travail a consisté à comprendre les principaux aspects du modèle organisationnel AGR, puis à le décrire en termes de logique de réécriture.

Les résultats présentés dans la section 3 montrent que cette formalisation est faisable, au moins pour ce modèle.

De plus, dans [Ast08, Ast09, Das08a], les auteurs décrivent l'utilisation de la logique de réécriture et de Maude dans le contexte des langages de programmation d'agents avec des fondements organisationnels. Ces travaux diffèrent de l'approche proposée ici par le fait qu'ils implémentent un langage de programmation d'agent qui possède déjà une sémantique formelle indépendante de Maude. Par conséquent, ces techniques de mise en œuvre sont basées sur l'idée que les règles de transition peuvent être traduites en règles de réécriture.

Un autre exemple de l'utilisation de Maude dans le contexte des langages de programmation d'agent est présenté dans [Van06]. Dans ce travail, les auteurs remarquent que les langages de programmation d'agents tels que 3APL ont à la fois une composante logique et une composante sémantique, ce qui rend Maude très bien adapté pour leurs prototypage. De plus, bien qu'ils aient implémenté une version simplifiée de 3APL, les auteurs soutiennent que Maude est un bon candidat pour implémenter diverses extensions de cette version implémentée. Et ils ajoutent qu'un avantage important de Maude, outre le fait qu'il est bien adapté pour le prototypage de 3APL, et qu'il peut être utilisé pour la vérification car il possède un vérificateur de modèle (model checker) basé sur la logique temporel linéaire.

4. Conclusion

Plusieurs méthodes décrivant les exigences organisationnelles de système multi-agent sont proposées. Cependant, ils ne proposent que des descriptions informelles ou semi-formelles. Dans ce chapitre, une approche permettant la spécification et la validation de système multi-agent centré sur l'organisation est présentée. Plus précisément, un modèle organisationnel particulier de SMA est spécifié. Cette formalisation montre que cette approche peut être appliquée à des modèles organisationnels qui n'ont pas été analysés formellement au début. L'approche proposée permet de capturer les aspects fonctionnels, statiques et dynamiques du système multi-agent basé sur le modèle d'organisation Agent-Groupe-Rôle à l'aide de diagrammes AUML, puis de traduire cette description graphique dans le langage Maude.

Ainsi, une approche basée sur la logique de réécriture est appliquée pour spécifier formellement le modèle AGR. Les avantages d'une telle approche sont, premièrement, que le modèle AGR soit présenté avec des termes formels et non ambigus et deuxièmement que la spécification décompose le modèle en des concepts formels qui peuvent être réutilisés dans des applications spécifiques. Ainsi, l'utilisation de notations formelles pour spécifier le modèle AGR permet de produire des descriptions organisationnelles précises. Cela offre également un meilleur support à leurs processus de vérification et de validation. D'autre part, la phase de validation permet de simuler des spécifications avec des scénarios d'interaction prédéfinis tels que l'étude de cas gestion de chaîne logistique. La spécification d'autres modèles organisationnels multi-agents est également planifiée en suivant le processus décrit dans ce chapitre. Cela constituera une bibliothèque de modèles multi-agents génériques et réutilisables.

Ce travail est la première étape dans la réalisation de la formalisation organisationnelle à travers un langage basé sur la logique de réécriture. D'autres travaux suivront en analysant les concepts organisationnels temporels de système multi-agent temps réel (SMATR). En

effet, la possibilité d'inclure une dimension temporelle dans le modèle organisationnel est d'une importance majeure pour traiter les problèmes des contraintes temporelles que les agents d'un SMATR peuvent être amenés à respecter et elle doit être étudiée pour spécifier formellement le modèle concret de tel système à travers l'extension Real Time Maude du langage Maude.

Chapitre 5

Vers un Modèle Organisationnel pour la Spécification des SMATR

Introduction	108
1. Modèle Organisationnel pour les SMATR	109
1.1. Méta-Modèle Organisationnel	110
1.2. Définition des rôles d'organisation d'agent avec les diagrammes de cas d'utilisation temporel AUML centré organisation	111
1.3. Modélisation statiques d'organisation d'agents avec les diagrammes de classe temporel AUML centré organisation	112
1.4. Modélisation des comportements dynamique d'organisation d'agent	113
2. Spécification formelle du modèle organisationnel de SMATR	113
2.1. Processus de Translation	114
3. Discussion	117
4. Conclusion	118

Introduction

Argente, Julian et Botti [Arg06] ont établi que dans l'analyse et la conception de SMA, les méthodologies ont évolué d'une vue initiale du système qui est centrée sur les aspects individuels des agents à une vue actuelle du système en tant qu'organisation dans laquelle les agents forment des groupes et des hiérarchies et suivent des règles de comportement spécifiques. Par conséquent, ils distinguent deux approches méthodologiques différentes: les méthodologies centrées sur l'agent et celles centrées sur l'organisation. Les méthodologies centrées sur l'agent ne sont pas assez matures; ils réduisent le potentiel de déploiement d'une communauté d'agents à grande échelle développée par des personnes différentes [Obo11]. Nous nous intéressons à cette limitation en utilisant l'abstraction organisationnelle pour modéliser les SMA en général et les systèmes multi-agents temps réel (SMATR) en particulier.

Une tendance récente dans le développement de systèmes distribués temps réel est l'utilisation de SMATR. Contrairement aux SMA conventionnels, les SMATR reflètent les caractéristiques intrinsèques des systèmes temps réel, plus précisément les contraintes temporelles. Pendant de nombreuses années, les concepteurs de SMA ont des méthodologies de développement et des langages de modélisation qui ne reflète pas les différentes contraintes temporelles que ces systèmes peuvent avoir. En effet, il n'est pas facile de conceptualiser les concepts de SMATR en utilisant des approches classiques d'ingénierie logicielle d'agent. Par conséquent, un problème de recherche critique pour la communauté des agents temps réel a été la définition d'un modèle organisationnel approprié pour l'analyse et la conception de leurs propriétés principales.

De plus, même les méthodologies proposées pour le développement de SMATR comme «RT-Message» [Jul04], «BDI-ASDP étendu pour le temps réel» [Mel05] et «la Méthode de développement de Lichen Zhang» [Zha06], sont inadéquates. Elles ont certainement apporté des réponses importantes dans le processus de développement de SMATR. Cependant, l'aspect méthodologique n'est pas encore maîtrisé. En effet, aucune de ces méthodologies ne dispose d'un modèle organisationnel prenant en compte la formalisation des concepts organisationnels du futur système, comme nous l'avons illustré dans le chapitre 2.

À notre avis, la modélisation des aspects organisationnels de SMATR est un moyen important pour les activités d'analyse et de conception. En outre, la conception de SMA nécessite l'implication de techniques de génie logiciel. Parmi ces techniques: UML² est probablement le langage le plus connu et le plus utilisé pour la modélisation orientée objet. Les développeurs SMA ont récemment les mêmes fonctionnalités en particulier le langage Agent UML (AUML) [Ode00]. Cependant, il n'y a actuellement aucun travail qui applique AUML pour les spécifications organisationnelles de SMATR et à la fois pour les applications temps réel.

Le présent travail est une continuité de ce qui a été présenté au chapitre précédent. Il se trouve dans le contexte du même projet, dont l'objectif est de développer un modèle organisationnel générique pour une spécification orientée organisation des aspects des SMATR. Dans ce chapitre, nous proposons les principaux concepts organisationnels et temporels de notre méta-modèle ainsi que les extensions réalisées dans les diagrammes AUML qui étendent un travail précédent [Lao10], à partir des étapes suivantes:

- Extraction des rôles d'agent en fonction des concepts de cas d'utilisation,

² Unified Modelling Language: <http://www.uml.org/index.htm>

- Représentation des structures organisationnelles statiques de SMATR en se basant sur la sémantique et la syntaxe du diagramme de classe étendue AUML temporel [Lao10],
- Extension des diagrammes de séquences temporelles et des diagrammes d'états transition temporels [Lao10] pour représenter les comportements interactifs entre les rôles d'agents et les comportements de raisonnement dans les rôles d'agents eux-mêmes.

Par conséquent, l'approche proposée ici, considère conjointement les aspects fonctionnels, statiques et dynamiques de SMATR. Son but est de combiner les avantages d'AUML et les concepts organisationnels extraits de l'étude faite au chapitre 2 sur cinq méthodologies d'organisation d'agents dans une seule technique dès la phase d'analyse, et ceux avant de décrire la structure de base du framework de spécification formelle RT-Maude associé aux différentes représentations AUML.

Ce chapitre qui constitue une autre contribution est organisé comme suit: **la section 1** propose la conception du méta-modèle organisationnel par affinement et extension de modèles existants adapté aux SMATR en décrivant leurs entités et relations respectives. Ensuite, **la section 2** présente de manière approfondie les extensions faites aux diagrammes AUML. Après cela, **la section 3** donne la structure de base du framework de spécification formelle à utiliser. Et terminer par la conclusion dans **la section 4**.

1. Modèle Organisationnel pour les SMATR

En se basant sur les caractères organisationnels inhérents au SMA, nous proposons une idée de modélisation de SMATR centré sur les organisations à l'aide de représentations AUML étendues, qui comprennent: la définition des rôles d'organisation, la représentation de leurs structures statiques et leurs comportements dynamiques. Il convient de noter que les trois types de modèles sont liés entre eux et doivent être référencés les uns aux autres. Au cours du processus de modélisation de SMA, comment établir un modèle organisationnel et comment l'étendre de manière appropriée en fonction des caractéristiques de SMATR est-ce notre problème? Notre approche considère la représentation des concepts sociaux conjointement avec les contraintes temporelles des agents comme la base de la modélisation de SMATR et présente une première idée d'un modèle organisationnel pour spécifier les systèmes multi-agents temps réel.

Un modèle organisationnel est proposé sur la base des principaux concepts de: Temporal Organization; Role; Group; Social Norms; Goal; Task; Agent Organization Use Case; Temporal Agent Organization Use Case; Agent Organization; et Real Time Agent organization ... etc. Ces concepts sont décrits avec les diagrammes AUML, qui sont étendus afin d'inclure toutes les entités organisationnelles nécessaires pour décrire les dimensions fonctionnelles, structurelles et dynamiques d'une organisation. Ils sont utilisés pour représenter: (i) comment les rôles d'agent sont définis et connectés; (ii) quelle fonctionnalité ils offrent dans l'organisation en utilisant la notion de cas d'utilisation; et (iii) quelles restrictions sont nécessaires pour contrôler le comportement de l'entité au sein de l'organisation de SMATR.

Le modèle organisationnel proposé décrit les entités de SMATR (agents, organisations, rôles, contraintes temporelles, groupe, tâche, objectif ... etc) et leur relation (relations sociales, fonctionnalité requise ou offerte). En outre, nous essayons d'enrichir les diagrammes AUML avec certains concepts organisationnels (listés ci-dessus) basés sur la notion de stéréotype AUML et sur les entités du méta-modèle d'organisation (Figure 5.1).

Dans les sous-sections suivantes, la contribution principale est, en fait, des extensions de la notation proposée dans [Lao10]. Nous utilisons le suffixe "O" pour indiquer "organisation".

1.1. Méta-Modèle Organisationnel

Le méta-modèle d'organisation que nous proposons vise à spécifier des organisations de systèmes multi-agents temps réel. Nous adoptons certaines des idées du travail présentées dans [Del10].

Ce méta-modèle est utilisé pour l'analyse et la conception de SMATR, au moyen de la description de leurs composants (organisations, rôles, agents); fonctionnalités (objectifs et tâches); et interactions, à travers les concepts de : Temporal Organization, Task, Goal, Group, Agent-O, Real Time Agent-O, and External Agent-O, Temporal Restriction, etc. Ces concepts sont utilisés pour représenter: (i) comment les entités sont groupées et connectées entre elles; (ii) quelle fonctionnalité ils offrent dans l'organisation; et (iii) quelles restrictions sont nécessaires pour contrôler le comportement des entités. Ce méta-modèle d'organisation décrit les entités du système et comment elles sont liées les unes aux autres (leurs relations sociales).

Comme le montre la Figure 5.1, le méta-modèle organisationnel proposé contient plusieurs concepts, chacun d'eux est discuté ci-dessous. L'entité "Organisation" qui est composée de: But; Tâche; et les entités du groupe; et associé aux notions d'organisation temporelle et de protocole externe. Elle agit comme un groupe d'agents jouant plusieurs rôles qui décrivent un ensemble de fonctionnalités et des objectifs / tâches. Comme elle inclut également les normes sociales, plus particulièrement les restrictions temporelles, auxquelles peuvent accéder des membres spécifiques de l'organisation (Real-Time Agent-O) qui contrôlent leur comportement. De plus, cette entité organisationnelle est étendue avec le concept d'organisation temporelle défini comme une entité sociale composée de groupe d'Agents-O dont il existe au moins un Real-Time Agent-O (un agent avec des restrictions temporelles dans certaines de ses responsabilités ou tâches [Jul04]), qui accomplissent plusieurs tâches ou fonctions distinctes et sont structurées de manière à atteindre l'objectif principal de l'organisation de SMATR.

Un rôle définit une position au sein de l'organisation dont le comportement est censé atteindre un objectif particulier ou un ensemble d'objectifs. La classe agent est un élément atomique qui permet de définir les collections Agent-Organisation (Agent-O; Real-Time Agent-O; and External Agent-O), qui peuvent être détaillées par la suite, en définissant tous ses compartiments (Figure 5.4). Les agents sont affectés à jouer des rôles c'est-à-dire à effectuer le comportement attendu de ces rôles. Les associations d'héritage et d'agrégation sont les relations les plus courantes et les plus définies.

De plus, les protocoles d'interaction et les messages échangés sont explicitement pris en charge. Les protocoles d'interaction définissent les interactions entre les rôles ou entre l'organisation et les acteurs externes. Un protocole peut être de deux types, externe ou interne. Les protocoles externes spécifient les interactions entre l'organisation et les acteurs externes (c'est-à-dire, les humains ou d'autres applications logicielles), tandis que les protocoles internes spécifient les interactions entre les agents jouant des rôles spécifiques dans l'organisation. Les messages peuvent être utilisés pour définir des protocoles d'interaction. Ils sont généralement utilisés pour les communications.

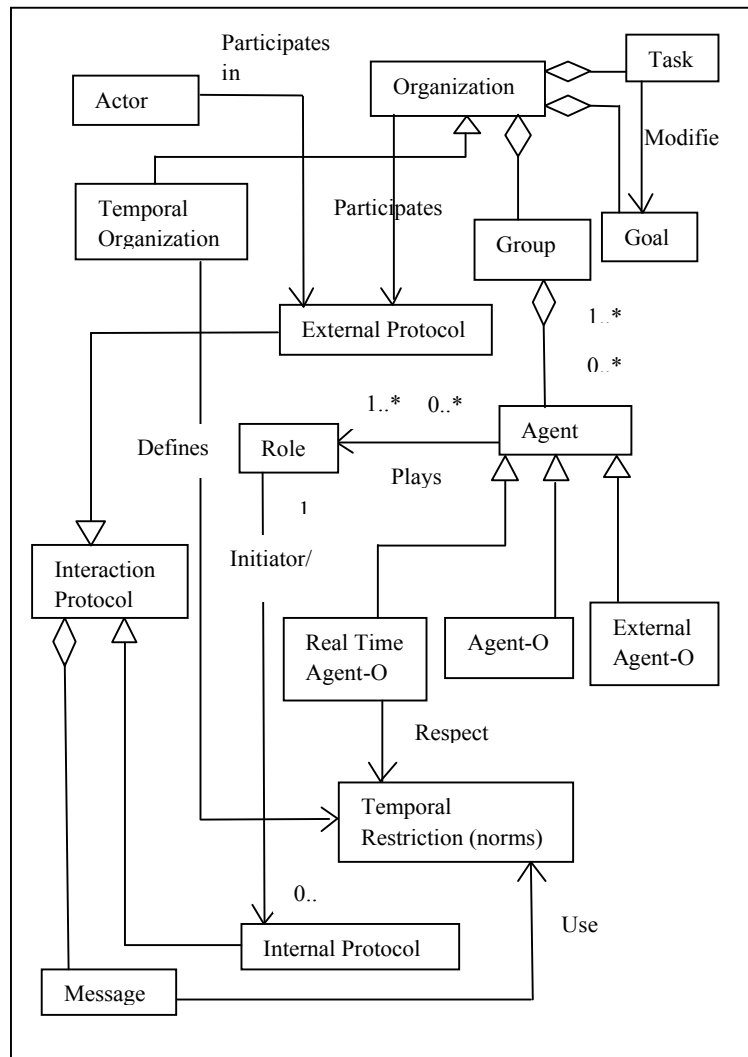


Figure 5.1. Le méta-modèle organisationnel.

1.2. Definition des Rôles d'organisation d'agent avec les diagrammes de cas d'utilisation temporel AUML centré organisation

Nous proposons que dans la phase initiale des exigences, un modèle organisationnel soit défini, détaillant les principaux acteurs, rôles, objectifs et dépendances. Dans ce modèle, plusieurs éléments organisationnels sont pris en compte (tels qu'une structure simple, des rôles d'agent et des objectifs).

Par conséquent, la plupart des méthodologies d'organisation d'agents analysent le SMA d'un point de vue organisationnel. Cependant, les contraintes temporelles que les agents temps réel doivent respecter ne sont pas prises en compte et il n'y a pas de proposition pour les intégrer dans l'organisation.

Dans [Che02], l'auteur décrit les rôles comme la synthèse des responsabilités, des fonctions et des comportements pertinents dans une organisation. Un certain rôle est responsable de certains objectifs. Au cours du processus de modélisation de l'organisation, les rôles sont les unités de base et la méthode qui prend les rôles comme éléments de modélisation de base; adopte des diagrammes

de cas d'utilisation pour décrire les fonctions du système (un objectif peut être considéré comme une abstraction d'un ensemble d'exigences fonctionnelles).

Étant donné que, l'analyse basée sur les cas d'utilisation est prouvée pour spécifier les exigences de SMA [Hei00, Pap00, Bau05], cela nous a motivés à appliquer ces mêmes techniques mais avec quelques modifications sur la modélisation organisationnelle de SMATR. En utilisant le mécanisme d'extension de «stéréotypage», les diagrammes de cas d'utilisation seront enrichis par les «cinq stéréotypes» suivants:

- Les stéréotypes «Agent-O Use Case» (Figure 5.2.a) et «Temporal Agent-O Use Case» (Figure 5.2.b) sont utilisés pour désigner respectivement, un cas d'utilisation représentant une fonctionnalité effectuée par des agents et un cas d'utilisation qui interagit avec des agents temps réel se trouvant dans certaines organisations.
- Les stéréotypes «Agent-O» (Figure 5.3.a), «External Agent-O» (Figure 5.3.b) et «Real Time Agent-O» (Figure 5.3.c) décrivent dans cet ordre: les agents au sein d'une organisation, les agents externes à l'organisation et les agents temps réel qui sont des agents-O ayant certaines contraintes temporelles à respecter en plus.

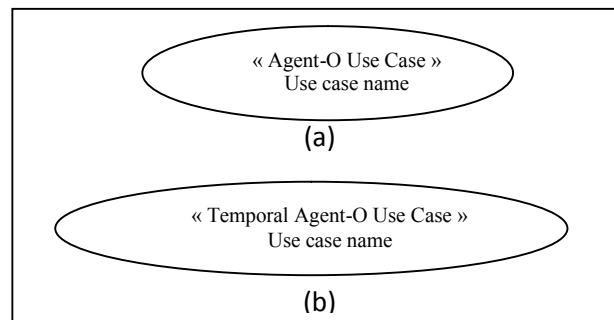


Figure 5.2. Cas d'utilisation organisationnel stéréotypés.

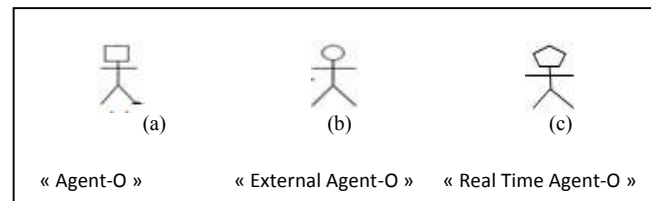


Figure 5.3. Notation d'organisation d'Agent.

1.3. Modélisation statique d'organisation d'agents avec les diagrammes de classe temporel AUML centré organisation

La modélisation des aspects organisationnels statiques pour un agent temps réel doit refléter les relations statiques entre les rôles, en tenant compte des restrictions temporelles de l'agent temps réel dans les organisations. Le but des diagrammes de classe AUML est de représenter les caractères statiques du SMA. Avec l'extension de ce type de diagrammes avec des entités organisationnelles basées sur le concept de stéréotype AUML, nous pouvons obtenir des diagrammes de classe temporel AUML centrés organisation. A tous les compartiments décrivent dans les diagrammes de classe temporel AUML proposé par [Lao10], de nouveaux compartiments appelés: Temporal Organization; Groups; Social norms; Goal; and Task; sont ajoutés (Figure 5.4).

« Real Time Agent-O »
role1, role2, ..., rolen
Temporal constraint1 (soft/hard).....
TemporalOrganization1..TemporalOrganizationn
Group1 ... Groupn
Social norms ...
Goals1 ... Goaln
Task1 ... Taskn
statechart1...statechartn
attribut1.....attributn
operation1...operationn
protocol1: role...protocoln: role

Figure 5.4. Diagramme de classe temporel AUMML centré organisation.

1.4. Modélisation des comportements dynamique d'organisation d'agent

Le modèle proposé décrit le comportement souhaité de l'organisation d'agent et sa structure générale (rôles, interactions et contraintes temporelles) au moyen des diagrammes d'état transition et de protocoles AUMML qui décrivent respectivement les comportements individuels et collectifs des agents.

Bien que ce framework organisationnel représente une approche importante pour la modélisation de SMATR, il s'agit d'une extension récente qui combine des concepts organisationnels (extraits des approches analysées au chapitre 2) avec les diagrammes étendus AUMML présentés dans [Lao10]. Dans ce framework, les avantages des deux approches (entités organisationnelles et représentations AUMML étendues avec la notation temporelle) sont unifiées, mais il manque encore des outils appropriés pour l'analyse et la conception.

2. Spécification formelle du modèle organisationnel de SMATR

Comme souligné ci-dessus, les méthodologies proposées pour le développement de SMA et celles proposées pour le développement de SMATR sont inadéquates. Ils ont certainement apporté des réponses importantes dans le processus de développement de SMATR. Cependant, l'aspect méthodologique n'est pas encore maîtrisé. En effet, aucune de ces méthodologies ne prend en compte la formalisation des exigences organisationnelles pour le futur système. La qualité de l'analyse du modèle revêt une extrême importance pour le reste des phases du processus de développement. Leur spécification formelle et leur validation permettent d'éviter de nombreux problèmes pouvant affecter la qualité du développement.

Formaliser les aspects organisationnels de SMATR est à notre avis un moyen important pour les activités d'analyse et de conception. De plus, la conception de SMATR nécessite l'implication de langages formels. Parmi ces langages: RT-Mauve [Olv07] est probablement le plus connu et le plus largement utilisé pour la spécification formelle orientée objet. Il n'y a actuellement aucun travail qui applique RT-Mauve à la spécification organisationnelle formelle de SMATR et à la fois aux applications temps réel.

Le présent travail se place dans le contexte de notre projet, dont l'objectif est de développer un framework formel organisationnel générique pour la spécification des aspects organisationnels de SMATR. Comme première étape de notre projet, nous avons présenté dans un travail précédent [Lao13], des extensions faites sur les diagrammes AUMML pour décrire les exigences

organisationnelles des systèmes multi-agents temps réel. Cependant, les modèles AUML [Ode00] souffrent d'un manque de sémantique formelle. Ils peuvent contenir des incohérences difficiles à détecter manuellement. Les méthodes formelles représentent une solution intéressante à ce problème. Les spécifications formelles auront pour effet d'éliminer les ambiguïtés dans l'interprétation des modèles. Pour cela, la combinaison d'AUML et de RT-Maude validera formellement le modèle organisationnel développé.

Ce travail présente une approche systématique supportant la translation des concepts organisationnels de SMATR représentée par l'extension des diagrammes AUML [Lao13] en spécification formelle RT-Maude. Ce dernier est un langage multi-paradigmes qui combine la programmation fonctionnelle et la programmation orientée objet. En outre, RT-Maude est très puissant en termes de spécification, de validation et de vérification des systèmes concurrents, ce qui en fait un bon candidat pour la spécification et la validation de SMATR.

Le but de cette approche est de traduire des représentations AUML étendus dans un framework formel RT-Maude pour intégrer la validation formelle de la cohérence du modèle organisationnel, dès la phase d'analyse.

2.1. Processus de Translation

Dans cette section, nous présentons la démarche qui nous permet d'obtenir une spécification formelle RT-Maude.

Le processus de translation proposé vise à spécifier formellement les diagrammes AUML étendus décrits ci-dessus (section 1) en langage RT-Maude afin de décrire les exigences organisationnelles de SMATR. Ce processus est divisé en trois étapes principales (Figure 5.5): (i) description des exigences organisationnelles de SMATR à l'aide des diagrammes AUML, (ii) validation inter-diagrammes et (iii) génération de spécification formelle RT-Maude.

La première étape est la phase d'analyse habituelle du processus de développement logiciel. La deuxième étape vise à valider la cohérence entre les modèles conçus et la dernière étape concerne la génération systématique de code source RT-Maude à partir des diagrammes AUML considérés.

Le framework formel proposé (Figure 5.6) est composé de plusieurs modules RT-Maude: modules fonctionnels, modules orientés objet et modules orientés objet temporisés. Nous présentons les principaux modules du framework formel proposé. Nous utilisons le suffixe "O" pour indiquer "organisation".

Le module GOAL décrit les objectifs de l'organisation des agents, et le module TASK définit les tâches qu'un agent-O peut effectuer. Ces deux modules avec le module GROUP (qui est utilisé pour définir les groupes Agent-O) sont importés dans le module ORGANIZATION afin de définir les organisations liées aux objectifs de l'agent.

Les modules AGENTi-O-BEHAVIOR qui importent le module ORGANIZATION sont utilisés pour illustrer le comportement individuel des Agents-O, et afin d'implémenter les interactions entre les différents Agents-O, des connexions entre eux sont effectuées via le module LINKED-BEHAVIOR, qui réutilise les modules AGENTi-O-BEHAVIOR.

Le mécanisme d'identification des Agents-O est défini par le module IDENTIFICATION, et la description de la structure des messages échangés entre les différents Agents-O est effectuée via le module MESSAGE qui importe les modules IDENTIFICATION et TASK.

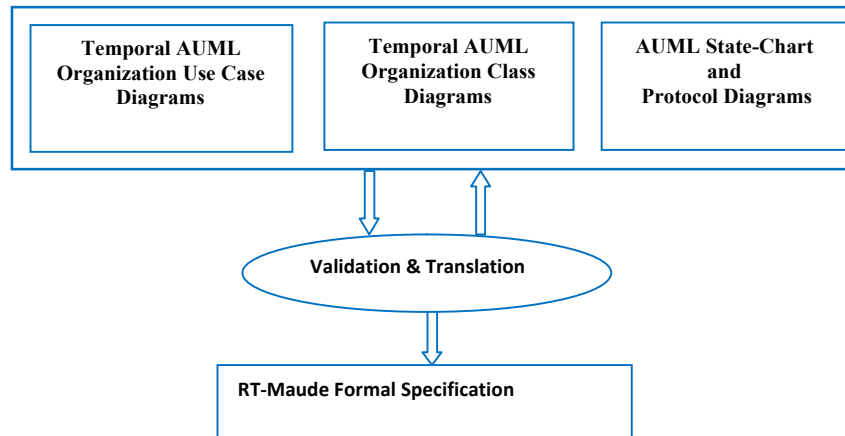


Figure 5.5. Aperçu du processus de translation.

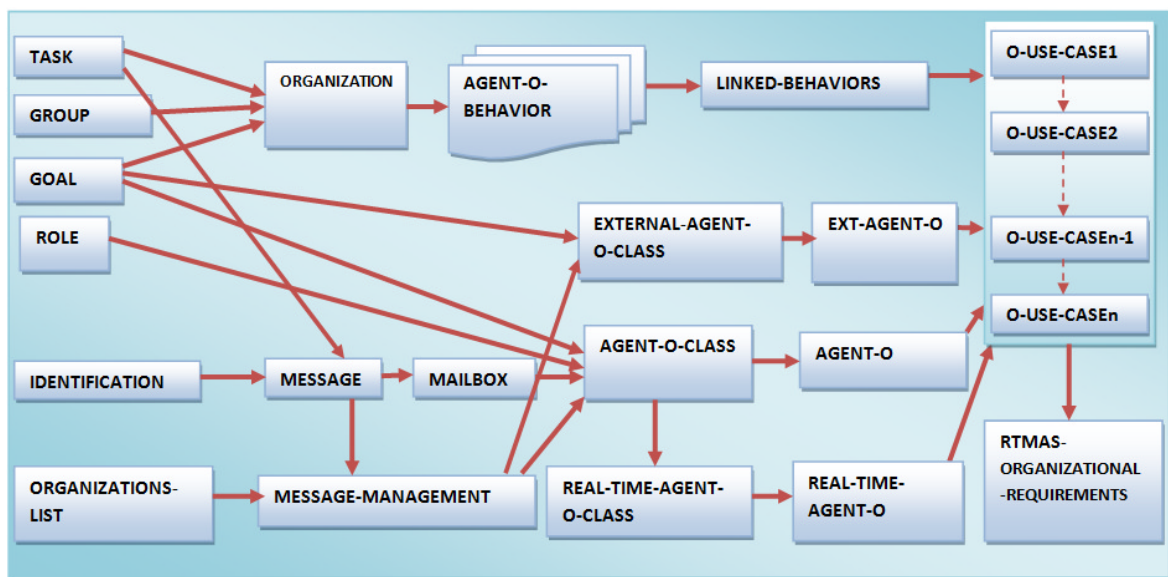


Figure 5.6. Les Modules Maude généré.

Pour faciliter la communication des agents, ils sont généralement dotés d'une boîte aux lettres contenant les messages reçus des autres agents-O et d'une liste de leurs organisations. Pour cela, nous définissons les modules fonctionnels MAILBOX et ORGANISATIONS-LIST pour gérer respectivement les Mailboxes et les organisations des listes d'agents.

Les rôles associés aux Agents-O sont définis dans le module ROLE, et pour décrire les opérations d'envoi/réception, nous définissons le module MESSAGE-MANAGEMENT qui importe les modules ORGANIZATIONS-LIST et MESSAGE.

Le module orienté objet EXTERNAL-AGENT-O-CLASS (Figure 5.7) est utilisé pour définir la classe de base des agents externes de l'organisation, avec les attributs CurrentGoal et OrgList (ligne [1]) représentant l'objectif actuel de l'agent et sa liste d'organisations. Ce module importe les modules GOAL et MESSAGE-MANAGEMENT.

```
(omod EXTERNAL-AGENT-O-CLASS is
  protecting GOAL . protecting MESSAGE-MANAGEMENT .
  class ExtAgentO | CurrentGoal: Goal, OrgList : OrganizationList .---[1]
  endom)
```

Figure 5.7. Module orienté objet EXTERNAL-AGENT-O-CLASS

Dans le module orienté objet AGENT-O-CLASS (Figure 5.8), nous définissons la structure de base de la classe des agents interne à l'organisation. Cette classe (ligne [1]) a comme attributs: PlayRole, CurrentGoal, MBox et OrgList pour contenir dans cet ordre: le rôle joué par l'agent, son objectif actuel, sa boîte aux lettres et sa liste d'organisations. Ce module importe tous les modules: GOAL, ROLE, MAILBOX et MESSAGE-MANAGEMENT.

```
(omod AGENT-O-CLASS is
  protecting GOAL. protecting ROLE.
  protecting MAILBOX . protecting MESSAGE-MANAGEMENT .
class AgentO | PlayRole : Role, CurrentGoal : Goal, MBox : MailBox,
               OrgList: OrganizationList .    --[1]
endom)
```

Figure 5.8. Module orienté objet AGENT-O-CLASS.

Pour décrire les Agents Temps Réel de l'organisation, nous avons défini la classe RealTimeAgentO avec l'attribut Clock (ligne [1]) dans le module orienté objet temporisé REAL-TIME-AGENT-O-CLASS (Figure 5.9) en tant que sous-classe de la classe Agent -O (ligne [2]).

```
(tomod REAL-TIME-AGENT-O-CLASS is
  extending AGENT-O-CLASS .
class RealTimeAgentO | Clock : Time .    ---[1]
subclass RealTimeAgentO < AgentO .    ---[2]
endtom)
```

Figure 5.9. Module orienté objet temporisé REAL-TIME-AGENT-O-CLASS.

A chaque cas d'utilisation de l'organisation est associé un module orienté objet O-USE-CASEi (Figure 5.10), qui porte le même nom que le cas d'utilisation de l'organisation correspondant. Dans chaque module O-USE-CASEi sont définies les règles de réécriture décrivant les différents scénarios d'interaction entre les agents-O définis dans les différents diagrammes de protocole AUML,. Notez que ces règles peuvent être des règles instantanées ou des règles tick, conditionnelles ou inconditionnelles.

```
(tomod O-USE-CASEi is
  including EXTERNAL-AGENTS-O . including AGENTS-O .
  including REAL-TIME-AGENTS-O. including LINKED-BEHAVIORS.
rl [1] : Configuration1 => Configuration2. ...
rl [m] : Configuration 2m-1 =><Configuration2m.
endtom)
```

Figure 5.10. Module orienté objet temporisé O-USE-CASEi.

Une fois générés, tous les modules O-USE-CASEi sont importés dans le module orienté objet temporisé RTMAS-ORGANIZATIONAL-REQUIREMENTS (Figure 5.11) qui décrit tous les concepts organisationnels du système.

```
(tomod RTMAS-ORGANIZATIONAL-REQUIREMENTS is
  including O-USE-CASE1. ... including O-USE-CASEm.
endtom)
```

Figure 5.11. Module orienté objet temporisé RTMAS-ORGANIZATIONAL-REQUIREMENTS.

La règle utilisée pour assurer la progression du Temps dans le système est donnée à la Figure 5.12 on l'appelle Tick Rule. Dans cette Figure nous avons défini le message Timer pour changer l'horloge de l'agent temps réel de l'organisation définie par l'attribut Clock (ligne [1]).

```

crl [tick] : {Timer(TimeOut)
< A : RealTimeAgentO | CurrentGoal : G, Clock : T, PlayRole:
Initiator> --[1] REST:Configuration}
=> {Timer(TimeOut minus 1)
< A : RealTimeAgentO | CurrentGoal: G, Clock : T plus 1>
REST:Configuration }
in time 1 if (TimeOut > zero).

```

Figure 5.12. Tick Rule.

3. Discussion

L'utilisation des notations AUML pour spécifier les concepts organisationnels de SMATR permet de produire un modèle organisationnel générique. Dans cette contribution, nous avons présenté les lignes directrices d'un modèle organisationnel générique supportant la description graphique des concepts organisationnels de SMATR.

L'approche proposée permet de représenter l'aspect fonctionnel (diagramme de cas d'utilisation temporel AUML centré organisation), les aspects statiques (diagramme de classe temporel AUML centré organisation), et les aspects dynamiques (diagrammes de protocole AUML et diagrammes d'état transition AUML) de SMATR selon la perspective organisationnelle. Les caractéristiques temporelles des systèmes temps réel peuvent être modélisées à l'aide d'AUML en utilisant les stéréotypes, les valeurs marquées et les contraintes en général. Nous croyons que l'approche proposée est particulièrement appropriée à la modélisation SMATR.

Nous avons proposé un méta-modèle d'organisation afin d'inclure principalement les concepts Temporal Organization; Agent-O; Real Time Agent-O; and External Agent-O avec les concepts habituels de rôle; Objectif; Tâche; et les normes sociales (en particulier les restrictions temporelles) qui ont été extraites des approches organisationnelles étudiées précédemment (AGR [Fer03a], Moise+ [Hub02a], INGENIAS [Gom05], Gaia [Woo00b] et RT-Message [Jul04]); pour être intégrés dans un cadre à la modélisation des organisations de SMATR. Ainsi, ce méta-modèle décrit les composants de l'organisation, les relations et les connexions.

Parmi les techniques possibles utilisées dans la littérature pour valider et vérifier les modèles informels et/ou semi-formels, figure celle qui consiste à traduire ces trois vues en descriptions formelles RT-Maude pour produire des descriptions précises et mieux soutenir leur processus vérification et validation.

Nous proposons que dans la phase initiale des exigences, un modèle organisationnel soit défini, détaillant les principaux acteurs, rôles, objectifs et dépendances. Dans ce modèle, plusieurs éléments organisationnels sont considérés (comme une structure simple, des rôles d'agent et des objectifs) et pendant le processus de translation, nous avons développé un framework formel basé sur le langage RT-Maude, où plusieurs modules sont générés, comme illustrés dans la Figure 5.6, pour intégrer la validation formelle de la cohérence du modèle organisationnel, dès la phase d'analyse pour produire des descriptions précises et aussi offrir un meilleur support à leurs processus de vérification et de validation.

En effet, notre idée consiste essentiellement à utiliser conjointement les diagrammes étendus AUML présentés dans la section 1, et le framework formel RT-Maude (section 2) ensemble pour soutenir la spécification formelle des exigences organisationnelles de SMATR. Dans ce framework organisationnel formel, les forces des deux approches sont unifiées.

Cependant, nous tenons à souligner que les résultats rapportés dans ce chapitre sont préliminaires, il reste encore beaucoup à faire. Ainsi, dans le prochain chapitre, l'expansion de ce modèle organisationnel pour englober les différents concepts, le long des dimensions identifiées qui se trouvent dans le modèle d'organisation Moise+ est développé. Nous décrivons également d'une part la validation de ces spécifications RT-Maude obtenus à partir des diagrammes étendus AUMML considérés, en utilisant le modèle organisationnel Moise+ et en appliquant les résultats d'étude de cas concrète. D'autre part, le développement de technique d'analyse formelle supplémentaire, à savoir l'analyse temporelle (en utilisant le model-checking) est illustrée pour produire des descriptions précises et aussi mieux soutenir leurs processus de vérification et de validation, en intégrant les possibilités offertes par Real-Time Maude pour vérifier certaines propriétés de la spécification des exigences organisationnelles du système multi-agent temps réel.

4. Conclusions

La formalisation des concepts organisationnels représente une activité importante au cours du processus de développement des systèmes multi-agents temps réel. Elle produit une description rigoureuse et offre une base solide pour les activités de vérification et de validation. Plusieurs méthodologies décrivant les concepts organisationnels de SMA sont proposées.

Cependant, ils n'offrent que des descriptions informelles ou semi-formelles. Dans cette contribution, nous avons proposé une approche générique permettant d'une part de capturer l'aspect fonctionnel (diagramme de cas d'utilisation temporel AUMML centré organisation), statique (diagramme de classe temporel AUMML centré organisation) et dynamique (diagrammes de protocole AUMML et diagrammes d'état transition AUMML) de SMATR basé sur la perspective organisationnelle, et d'autre part, traduire les descriptions AUMML vers une description formelle du langage RT-Maude. Ce dernier se caractérise par la puissance de la description et intègre plusieurs outils de vérification et de validation.

Bien que le framework organisationnel formel obtenu représente une approche importante pour la modélisation de SMATR, il s'agit d'une extension récente qui combine les diagrammes étendus AUMML [Lao13], avec une spécification RT-Maude. Dans ce framework, les avantages des deux approches sont unifiés.

Comme continuation à ce travail, nous prévoyons dans le chapitre suivant le développement de technique d'analyse formelle supplémentaire, à savoir l'analyse temporelle en utilisant la vérification de modèle par le Modèle checker fourni par le langage RT-Maude pour vérifier certaines propriétés de la spécification des exigences organisationnelles du système multi-agent temps réel.

Chapitre 6

Formalisation du Modèle Organisationnel RT-Moise+

Introduction	120
1. Méta-modèle Organisationnel RT-Moise+	121
2. Spécification Formelle de RT-Moise+	123
2.1. Architecture du Framework proposé	123
3. Cas d'Etude: Système de Gestion de Conférence (SGC)	129
3.1. Description	129
3.2. Décomposition selon RT-Moise+ du Système de Gestion de Conférence	130
3.3. Modélisation AUML du Système de Gestion de Conférence	133
3.4. Application du Processus de Translation	140
3.5. Validation de la Description Générée	143
3.6. Vérification de la Description Générée	145
4. Conclusion	148

Introduction

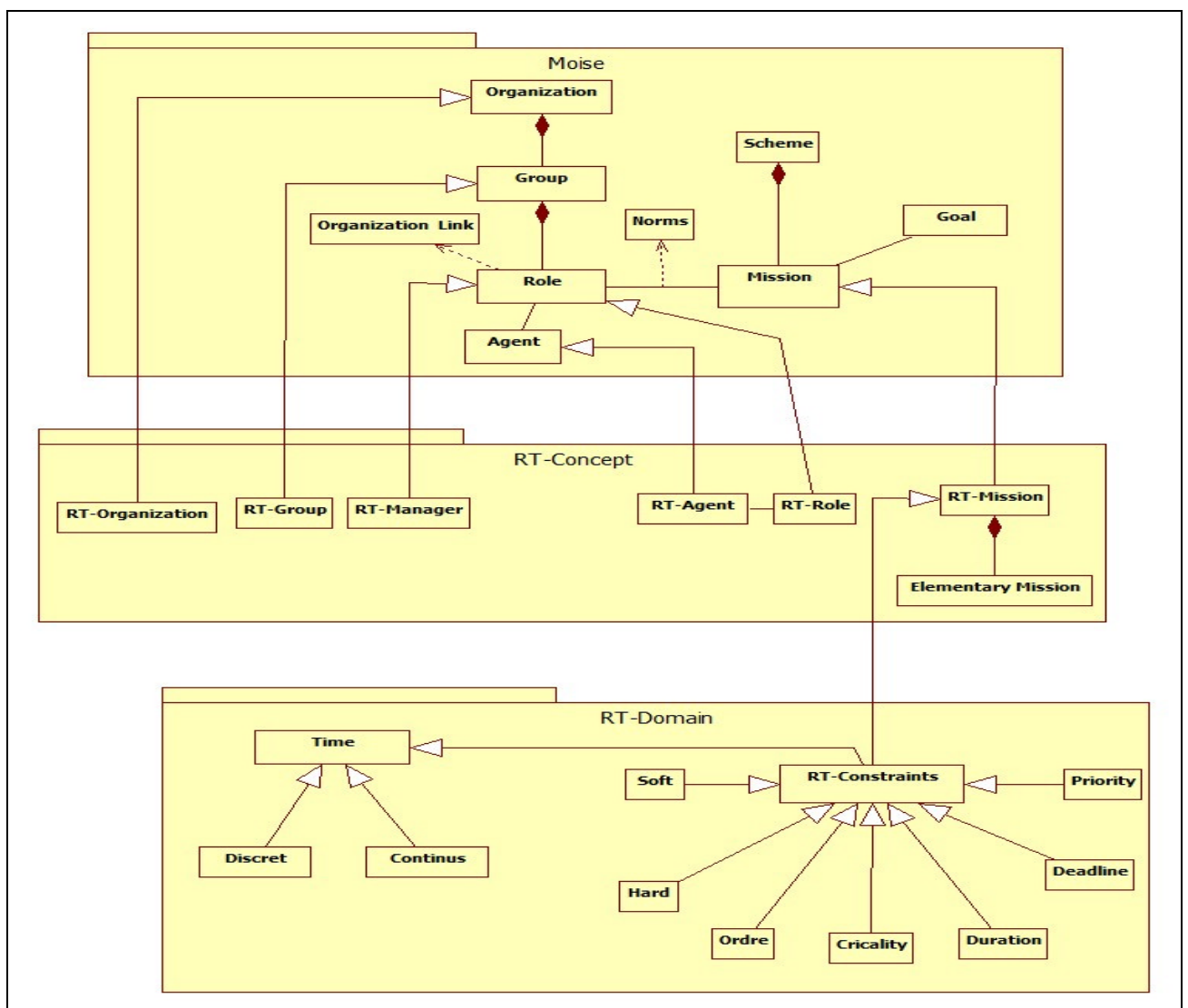
Les travaux sur les interactions et la coopération dans les systèmes multi-agents ont abouti à la proposition de nombreux modèles utilisant des concepts sociaux appelés modèles organisationnels. On retrouvera ainsi des concepts tels que les rôles, groupes, et communautés qui rappellent les caractéristiques sociales des organisations humaines.

La diversité des interactions, si elle permet la résolution de problèmes complexes, amène également des situations conflictuelles (conflit d'accès à une ressource, etc) et des situations où des actions redondantes sont entreprises par les agents. Il devient alors nécessaire de coordonner les interactions existantes entre les agents, afin de structurer leurs activités [Foi98]. Ce problème est abordé à travers la notion d'organisation qui est un autre aspect fondamental du paradigme multi-agents. Cette notion se préoccupe de définir, gérer et modifier les relations entre les individus. Cette description se fait en termes de définition de rôles, de comportements, de relations d'autorité, d'assignation de tâches, etc [Gas92]. Elle permet de contrôler l'ensemble des agents pour qu'ils résolvent ensemble un problème, ceci en restreignant les comportements individuels pour limiter les conflits potentiels et pour gérer le problème de cohérence globale dans le système.

Dans [Gut99] les auteurs définissent l'organisation comme un framework d'activité et d'interaction, mis en place par la définition de groupes, rôles et de leurs relations. Ils considèrent que l'organisation peut être décrite uniquement par sa structure, c'est à dire par la manière dont sont définis et liés les groupes et rôles, sans se concerner du comportement effectif des agents. Dans notre travail nous allons essayer de concentrer notre effort sur l'aspect fonctionnel des agents au sein de l'organisation sans entrer dans les détails des architectures d'agents et des modèles de comportements.

L'approche que nous proposons pour la description formelle des systèmes multi-agents temps réel centré organisation est principalement basée sur le modèle organisationnel Moise+ (Model of Organization for multi-agent SystEms) [Hub02a]. Ce dernier a été étendu pour intégrer la dimension temporelle composante importante des SMATR. Le nouveau méta-modèle obtenu est noté RT-Moise+ pour Real Time Moise+. Au début nous avons utilisé le formalisme de modélisation graphique Agent UML vu qu'il offre plusieurs modèles conceptuels pour : décrire les besoins fonctionnels des agents ; spécifier les protocoles d'interaction ; représenter le comportement interne et collectif des agents ; etc. Dans notre contexte, on a utilisé les diagrammes de classe, de cas d'utilisation, de communication, d'état transition, de protocoles, et de temps. Cependant, AUML souffre d'un manque de sémantique formelle. Les modèles AUML développés pourront donc contenir des incohérences ou des inconsistances difficiles à détecter manuellement. La combinaison d'AUML avec un langage de spécification formelle basé sur la logique de réécriture comme RT-Maude permettra à notre avis de valider formellement les modèles AUML développés. Donc, nous allons spécifier formellement les modèles AUML décrivant les concepts organisationnels des SMATR centré sur le modèle RT-Moise+. Cette démarche permet d'éliminer certaines erreurs de façon précoce avant de passer aux phases de conception et d'implémentation.

Ce chapitre est organisé comme suit : nous commençons tout d'abord par présenter le méta-modèle permettant l'extension du modèle organisationnel Moise+ pour intégrer la dimension temporelles des systèmes multi-agent temps réel on la nommé RT-Moise+ pour



Le premier englobe les concepts du modèle Moise+, à savoir, l'Organisation, le Group, le Rôle, l'Agent et les Missions. Alors que le deuxième package RT-Domain représente notre spécification des concepts temps réel. Naturellement, le temps est le concept de base des systèmes temps réel. Le temps peut être discret ou continu. Les contraintes temps réel sont des contraintes spécifiées par des valeurs temporelles. En conséquence, ces contraintes peuvent être des deadline (c'est-à-dire une valeur qu'on ne peut pas dépasser) ou des périodes spécifiées par une date de début et une date finale. A notre avis, les périodes peuvent être utilisées pour spécifier la validité des connaissances ou des missions. Nous pouvons distinguer des contraintes soft et des contraintes hard. En plus, il est important de distinguer les contraintes critiques et les contraintes non-critiques.

Afin de relier les deux Packages, nous avons introduit de nouveaux concepts liés aux organisations temps réel. Le concept de base est peut être le concept RT-Rôle (Rôle temps réel). Ce rôle est spécifié par l'exécution de missions temps réel. Les missions temps réel sont des missions avec des contraintes temps réel. Naturellement, une mission temps réel est composée d'une ou plusieurs missions élémentaires (chaque mission élémentaire avec durée d'exécution). Les missions temps réel doivent être exécutées par des agents temps réel. En conséquence, les rôles temps réels doivent être adoptés seulement par des agents temps réel. L'existence d'un rôle temps réel au sein d'un group rend ce dernier un group temps réel. Par rapport aux groupes ordinaires, les groupes temps réel disposent d'un rôle particulier qui gère les contraintes temps réel. La Figure 6.2 résume bien les concepts organisationnels de RT-Moise+.

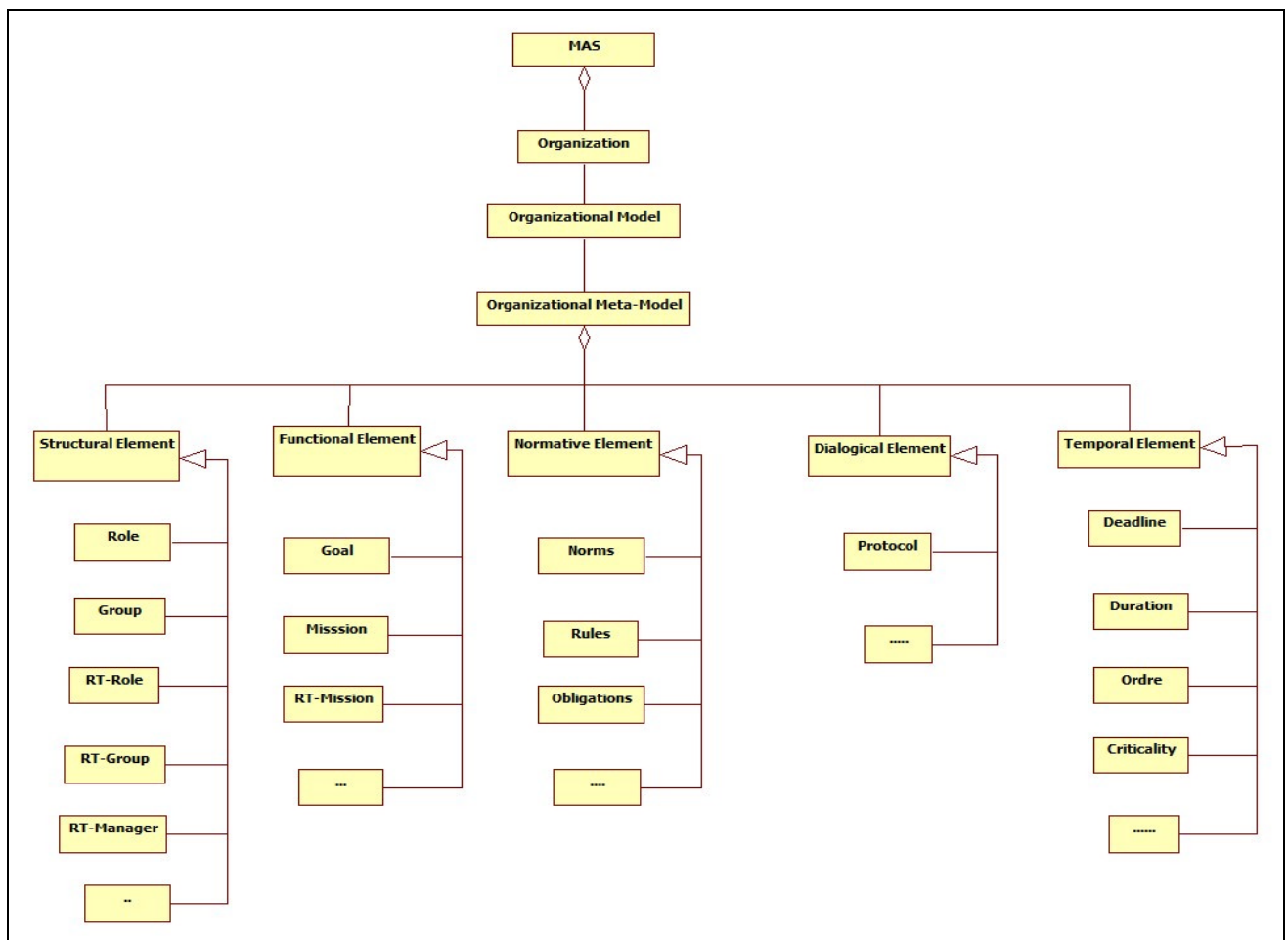


Figure 6.2. Ontologie des concepts organisationnels de RT-Moise+.

Dans la section suivante, nous allons décrire notre approche basée sur la description formelle du modèle RT-Moise+ en utilisant le langage RT-Maude.

2. Spécification Formelle de RT-Moise+

Dans cette section un framework formel est développé pour spécifier formellement le modèle RT-Moise+. Son architecture ainsi que la fonction de chacun de ces composants sont données ci-dessous.

2.1. Architecture du Framework Proposé

L'architecture du framework est donnée dans la Figure 6.3 ci-dessous. Comme vous pouvez le constater, elle est composée d'un ensemble de modules qui sont de divers types, fonctionnel (fmod); orienté objet (omod) ou bien orienté objets temporisés (tomod).

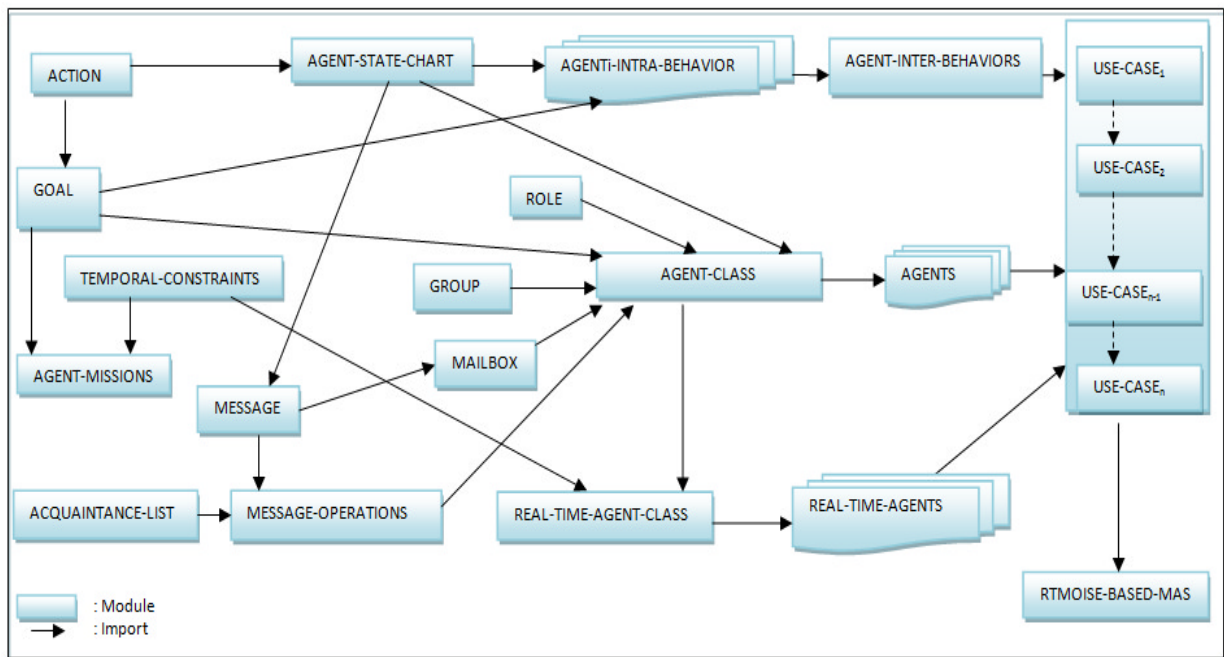


Figure 6.3. Architecture du Framework proposé.

On commence par le module fonctionnel **ROLE** (Figure 6.4) qui contient le type *AgentRole* qui est utilisé pour définir l'ensemble des rôles que peut jouer les différents agents.

```
(fmod ROLE is
  sort AgentRole .
  ***** User Part
endfm)
```

Figure 6.4. Le module fonctionnel ROLE.

Pour définir le type des agents, nous avons proposé le module orienté objet **Group** (Figure 6.5) qui contient le type *GroupeType* qui est utilisé pour définir l'ensemble des Types des différents agents. C'est-à-dire est-ce que l'agent appartient à un groupe interne au système ou bien à un groupe externe de celui-ci.

```

(omod GROUP is
  including STRING .
  sorts GroupeType .
  ****[1]

  ***** User Part

endom)

```

Figure 6.5. Le module orienté objet GROUP.

Dans le module fonctionnel **ACTION** (Figure 6.6), en plus des types *Action* et *ActionList*, nous définissons trois autres fonctions: *IsInternalAction*, *IsReceivingAction*, et *IsSendingAction*. La première fonction détermine si une action est interne, alors que les deux dernières fonctions déterminent respectivement si l'action est une action de réception ou bien elle est destinée à un agent de la liste d'accointances des agents.

```

(omod ACTION is
  including STRING .
  sorts ActionList Action .
  subsort Action < ActionList .
  subsort String < ActionList .

  op _:_ : Action ActionList -> ActionList .
  op EmptyActionList : -> ActionList .
  op IsInternalAction : Action -> Bool .
  op IsReceivingAction : Action -> Bool .
  op IsSendingAction : Action -> Bool .
  var Act : Action . var ActL : ActionList .
  eq Act : EmptyActionList = Act .

endom)

```

Figure 6.6. Le module orienté objet Action.

Pour définir le but ou l'objectif de chaque agent, nous avons défini le module Orienté Objet **GOAL** (Figure 6.7) qui contient les types *Goal*, *GoalList*, ainsi que des opérations utilisées pour la construction et la manipulation d'un Goal comme:

- *NextGoal* (ligne [2]) qui détermine le Goal prochain en tenant compte du goal actuel de l'agent et de l'action à accomplir.

```

(omod GOAL is
  including STRING .
  including ACTION .
  sorts GoalList Goal .
  subsort Goal < GoalList .
  subsort String < GoalList .

  op NextGoal : Goal Action -> Goal .
  op _:_ : Goal GoalList -> GoalList .
  op EmptyGoalList : -> GoalList .
  op ErrorGoal : -> Goal .
  op ErrorGoalList : -> GoalList .
  op EmptyGoalList : -> GoalList .
  var Go : Goal . var GoL : GoalList .
  eq Go : EmptyGoalList = Go .

endom)

```

Figure 6.7. Le module orienté objet GOAL.

Le module fonctionnel **AGENT-STATE-CHART** (Figure 6.8) contient les déclarations des différents types nécessaires pour la description d'un état du diagramme d'état transition (ligne [1]) d'un agent, en plus de la définition du type *Condition* (ligne [2]) et des opérations utilisées pour la construction et la manipulation d'un état (lignes 3, 4 et 5) telles que:

- *TargetState* (ligne [4]) qui détermine l'état prochain de l'agent.
- *ActionToAccomplish* (ligne [5]) qui détermine l'action à accomplir en tenant compte de l'état de l'agent et de la condition vérifiée.

```

(fmod AGENT-STATE-CHART is
  including ACTION .
  sorts State NameState .
  sort Condition .

  op AgentState : NameState -> State .
  op TargetState : State Condition -> State .
  op ActionToAccomplish : State Condition -> Action .

endfm)

```

Figure 6.8. Le module fonctionnel AGENT-STATE-CHART.

Maintenant pour décrire l'ensemble des contraintes temporelles du SMATR que ses agents temps réel doivent respectés et prendre en considération dans leurs comportement, nous avons défini le module **TEMPORAL-CONSTRAINTS** (Figure 6.9) qui contient un ensemble de type de donnée, d'opération et d'équation pour les manipuler.

```

(fmod TEMPORAL-CONSTRAINTS is
  including NAT-TIME-DOMAIN .
  sorts RTConstraint RTConstraintList .
  subsort RTConstraint < RTConstraintList .
  subsort Time < RTConstraint .
  op HeadR : RTConstraintList -> RTConstraint .
  op TailR : RTConstraintList -> RTConstraintList .
  op _:_ : RTConstraint RTConstraintList -> RTConstraintList .
  op ErrorRTConstraint : -> RTConstraint .
  op ErrorRTConstraintList : -> RTConstraintList .
  op EmptyRTConstraintList : -> RTConstraintList .
  op IsEmptyRTConstraintList : RTConstraintList -> Bool .
  var RTCon : RTConstraint . vars RTConL : RTConstraintList .
  eq RTCon : EmptyRTConstraintList = RTCon .
  eq IsEmptyRTConstraintList(EmptyRTConstraintList) = true .
  eq IsEmptyRTConstraintList(RTCon : RTConL) = false .
  eq HeadR(EmptyRTConstraintList) = ErrorRTConstraint .
  eq HeadR(RTCon : RTConL) = RTCon .
  eq HeadR(RTCon) = RTCon .
  eq TailR(EmptyRTConstraintList) = ErrorRTConstraintList .
  eq TailR(RTCon : RTConL) = RTConL .
  eq TailR(RTCon) = EmptyRTConstraintList .

endfm )

```

Figure 6.9. Le module fonctionnel TEMPORAT-CONSTRAINTS.

Les différents Goal ainsi que l'ensemble des contraintes temporelles décrits ci-dessus seront inclus dans le module orienté objet **AGENT-MISSIONS** (Figure 6.10) en important les modules **GOAL** et **TEMPORAT-CONSTRAINTS** définissant ainsi les *missions* de chaque agent du système qui implémentent le concept de mission du modèle RT-Moise+.

```

(omod AGENT-MISSIONS is
  including GOAL .
  including TEMPORAL-CONSTRAINTS .
  sort Plan .
  sort Mission .
  sort RTMission .

  op _:_ : GoalList ActionList -> Plan .
  op _:_:_ : Plan Goal ActionList -> Mission .
  op _:_:_:_ : Plan Goal ActionList RTConstraint -> RTMission .

endom)

```

Figure 6.10. Le module orienté objet AGENT-MISSIONS.

Concernant les messages que les agents peuvent s'échanger entre eux, nous avons proposé un module orienté objet **MESSAGE** (Figure 6.11) qui importe le module **AGENT-STATE-CHART**. Ce module définit les trois types: *Identifier* (ligne [1]), *Message* (ligne [2]), et *Content* (ligne [3]) ainsi que des opérations et des équations pour les manipulées. Un message doit contenir les noms de l'émetteur, du récepteur, et un contenu.

```
(omod MESSAGE is
  protecting AGENT-STATE-CHART .
  sort Identifier .                ***[1]
  sort Message .                  ***[2]
  sort Content .                  ***[3]
  subsort Oid < Identifier .
  subsort Action < Content .

  op _:_:_ : Identifier Content Identifier -> Message .
  op GetIdent : Message -> Identifier .
  vars A A1 : Identifier . var Cont : Content .
  eq GetIdent (A : Cont : A1) = A .
endom)
```

Figure 6.11. Le module orienté objet MESSAGE.

Le module fonctionnel **MAILBOX** (Figure 6.12) permet de stocker les messages que reçoit chaque agent des autres agents de l'environnement, pour cela, on lui a importé le module **MESSAGE**. Ce module contient un type *MailBox*, et un ensemble d'opération pour la manipulation des messages tels que :

- *FrontMBox* qui donne le premier élément de la liste des messages reçu.
- *RemoveMBox* qui supprimer le premier message de la liste des messages.

Ainsi que des équations pour implémenter ces opérations. Tout cela va permettre de bien gérer la communication entre les agents.

```
(fmod MAILBOX is
  protecting MESSAGE .
  sort MailBox .
  subsort Message < MailBox .
  op ErrorMessage : -> Message .
  op ErrorMailBox : -> MailBox .
  op EmptyMailBox : -> MailBox .
  op FrontMBox : MailBox -> Message .          ***[1]
  op RemoveMBox : MailBox -> MailBox .          ***[2]
  op _ : Message MailBox -> MailBox .          ***[3]
  op InsertInMBox : Message MailBox -> MailBox .

  var M : Message . var MB : MailBox .
  eq RemoveMBox(EmptyMailBox) = ErrorMessage .
  eq RemoveMBox(M EmptyMailBox) = EmptyMailBox .
  eq RemoveMBox(M) = EmptyMailBox .
  eq RemoveMBox(M MB) = MB .
  eq FrontMBox(M MB) = M .
  eq FrontMBox(M) = M .
  eq FrontMBox(EmptyMailBox) = ErrorMessage .
  eq M EmptyMailBox = M .
  eq InsertInMBox(M, MB) = if MB == EmptyMailBox then M else M MB fi .
endfm)
```

Figure 6.12. Le module orienté objet MAILBOX.

Un agent doit savoir la liste des agents avec laquelle il est amené à communiquer. Cette liste est appelée « liste de connaissance » qui sera définie à travers le module **ACQUAINTANCE-LIST** (Figure 6.13). Ce dernier contient un ensemble d'opérations et d'équations facilitant la manipulation de cette liste.

```

(fmod ACQUAINTANCE-LIST is
  including STRING .
  sorts AcquaintanceList Acquaintance .
  subsort Acquaintance < AcquaintanceList .
  subsort String < Acquaintance .
  op HeadA : AcquaintanceList -> Acquaintance .          ***[1]
  op TailA : AcquaintanceList -> AcquaintanceList .       ***[2]
  op _ : Acquaintance AcquaintanceList -> AcquaintanceList .
  op ErrorAcquaintance : -> Acquaintance .
  op ErrorAcquaintanceList : -> AcquaintanceList .
  op EmptyAcquaintanceList : -> AcquaintanceList .
  op IsEmptyAcquaintanceList : AcquaintanceList -> Bool .
  var Ac : Acquaintance . vars AcL : AcquaintanceList .
  eq Ac : EmptyAcquaintanceList = Ac .
  eq IsEmptyAcquaintanceList (EmptyAcquaintanceList) = true .
  eq IsEmptyAcquaintanceList (Ac : AcL) = false .
  eq HeadA (EmptyAcquaintanceList) = ErrorAcquaintance .
  eq HeadA (Ac : AcL) = Ac .
  eq HeadA (Ac) = Ac .
  eq TailA (EmptyAcquaintanceList) = ErrorAcquaintanceList .
  eq TailA (Ac : AcL) = AcL .
  eq TailA (Ac) = EmptyAcquaintanceList .
endfm)

```

Figure 6.13. Le module fonctionnel ACQUAINTANCE-LIST.

Pour la définition des opérations d'envoi et de réception de message: *SendMessage* et *RecieveMessage* (lignes 1 et 2]) sont défini dans le module orienté objet **MESSAGE-OPERATIONS** (Figure 6.14) qui importe les deux modules **MESSAGE** et **ACQUAINTANCE-LIST**.

```

(omod MESSAGE-OPERATIONS is
  protecting ACQUAINTANCE-LIST .
  protecting MESSAGE .
  subsort Acquaintance < Identifier .

  op SendMessage : Message -> Msg .          ***[1]
  op ReceiveMessage : Message -> Msg .       ***[2]

endom)

```

Figure 6.14. Le module orienté objet MESSAGE-OPERATIONS.

La description de l'aspect statique des agents c'est-à-dire leur classe de base est définie par le module orienté objet **AGENT-CLASS** (Figure 6.15). Ce module importe les modules: **AGENT-STATECHART**, **ROLE**, **GOAL**, **MAILBOX**, **MESSAGE-OPERATIONS**, et **GROUP**. Il contient la classe **Agent** (ligne [1]) avec les attributs: *PlayRole* (le rôle de l'agent), *CurrentState* (l'état courant de l'agent), *AgentGoal* (le but de l'agent), *GrType* (le type de l'agent), *AcqList* (sa liste d'accointance), et *MBox* (boite aux lettres).

```

(omod AGENT-CLASS is
  protecting AGENT-STATE-CHART .
  protecting ROLE .
  protecting GOAL .
  protecting MAILBOX .
  protecting MESSAGE-OPERATIONS .
  protecting GROUP .

  class Agent | CurrentState : State, PlayRole : AgentRole, AgentGoal : Goal, GrType : GroupeType,
               AcqList : AcquaintanceList, MBox : MailBox .          ***[1]

endom)

```

Figure 6.15. Le module orienté objet AGENT-CLASS.

Les agents temps réel qui sont une catégorie des agents mais ayant des contraintes temporelles à respecter, sont définis par la classe *RTAgent* avec l'attribut *RTConstList* (ligne [1]) dans le module orienté objet temporisé **REAL-TIME-AGENT-CLASS** (Figure 6.16). Cette classe *RTAgent* est une sous classe de la classe *Agent* (ligne [2]) et contient l'ensemble des attributs : *PlayRole*, *AgentGoal*, *GrType*, *CurrentState*, *AcqList*, *MBox*, pour décrire dans l'ordre, le rôle actuel de l'agent, son Goal, son Type, son état courant, sa liste d'accointances, sa boîte aux lettres, et ses contraintes temporelles.

```
(omod REAL-TIME-AGENT-CLASS is
  extending AGENT-CLASS .
  protecting TEMPORAL-CONSTRAINTS .
  including NAT-TIME-DOMAIN .

  class RTAgent | RTConstList : RTConstraintList .      ***[1]
  subclass RTAgent < Agent .                             ***[2]
endom)
```

Figure 6.16. Le module orienté Objet REAL-TIME-AGENT.

Les deux modules **AGENTS** (Figure 6.17), et **REAL-TIME-AGENTS** (Figure 6.18) qui étendent les modules **AGENT-CLASS**, et **REAL-TIME-AGENT-CLASS** respectivement, sont définis afin de permettre à l'utilisateur d'en spécifier l'ensemble des agents classiques et des agents temps réel dont il a besoin dans le système.

```
(omod AGENTS is
  extending AGENT-CLASS .
  ***** User Part
endom)
```

Figure 6.17. Le module orienté Objet AGENTS.

```
(omod REAL-TIME-AGENTS is
  extending REAL-TIME-AGENT-CLASS .
  ***** User Part
endom)
```

Figure 6.18. Le module orienté Objet REAL-TIME-AGENTS.

Le module fonctionnel **AGENT-INTRA-BEHAVIOR** (Figure 6.19) est défini pour décrire l'ensemble des interactions de chaque agent comme spécifier dans le diagramme d'état transition correspondant. Ce module importe les modules **AGENTSTATE-CHART** et **GOAL**. L'utilisateur dans ce module va définir *les états*, *les conditions*, *les goals* et *les actions* de l'agent, pour ensuite déterminer :

- *le prochain état* de l'agent selon son état source et la condition qui doit être vérifiée,
- *le prochain goal* selon le goal actuel et l'action à accomplir,
- *l'action à exécuter* d'après l'état actuel et la condition, et à la fin
- *le type* des différentes actions.

```
(fmod AGENT-INTRA-BEHAVIOR is
  protecting AGENT-STATE-CHART .
  protecting GOAL .

  ***** User Part
endfm)
```

Figure 6.19. Le module fonctionnel AGENT-INTRA-BEHAVIOR.

Pour le respect des interactions entre les agents comme décrit dans le diagramme de protocole, on a défini le module fonctionnel **AGENTS-INTER-BEHAVIORS** (Figure 6.20). Ce module importe les modules **AGENT-INTRA-BEHAVIOR** des agents du système. Il contient les deux opérations *correspondingState* (ligne [1]), et *correspondingCondition* (ligne [2]) pour faire la correspondance entre les différents états des agents et aussi entre les actions de l'expéditeur et du destinataire.

```
(fmod AGENT-INTER-BEHAVIORS is
  protecting AGENT-INTRA-BEHAVIOR .

  op CorrespondingState : State -> State .          ***[1]
  op CorrespondingCondition : Action -> Condition . ***[2]

  ***** User Part

endfm)
```

Figure 6.20. Le module fonctionnel AGENT-INTER-BEHAVIOR.

Concernant la description des fonctionnalités du système, on a défini le module temporisé orienté-objet **SYSTEM_USE_CASES** (Figure 6.21) qui contient l'ensemble des règles de réécriture décrivant les différents cas d'utilisation proposés dans les diagrammes AUML correspondant. Il importe les modules **AGENTS**, **REAL-TIME-AGENTS** et **AGENT-INTER-BEHAVIORS**.

```
(tomod SYSTEM_USE_CASES is
  including AGENTS .
  including REAL-TIME-AGENTS .
  including AGENT-INTER-BEHAVIORS .

  ***** User Part

endtom)
```

Figure 6.21. Le module temporisé orienté objet SYSTEM_USE_CASES.

Le module temporisé orienté-objet **RTMOISE-BASED-MAS** (Figure 6.22) importe le module **SYSTEM_USE_CASES** (Figure 6.21) pour décrire tout le système multi-agent temps réel qui est basé sur le modèle organisationnel RT-Moise+.

```
(tomod RTMOISE-BASED-MAS is
  including SYSTEM_USE_CASES .

endtom)
```

Figure 6.22. Le module temporisé orienté objet RTMOISE-BASED-MAS.

3. Cas d'Etude: Système de Gestion de Conférence (SGC)

3.1. Description

On a choisi de valider notre approche avec l'exemple Système de Gestion de Conférence (SGC). Dans cette étude de cas on a les agents suivants:

- l'agent temps réel PCChair qui est un agent interne au système. Il fait les tâches suivantes:
 - Initialise la conférence par l'envoi du message CFP aux participants.
 - Récupère les abstracts à partir de l'agent DataBase pour les partitionner et les envoyer aux agents PCMember.

- Il évalue les reviews provenant des agents Reviewers.
- Et à la fin, informe les agents Authors de ses décisions.
- L'agent PCMember est de type interne:
 - Reçoit les abstracts provenant de PCChair
 - Affecte chacun des abstracts reçus à trois Reviewers.
- L'agent temps réel DataBase est interne au système. Il gère la base de donnée par :
 - La collecte des papiers soumis et les versions finales des papiers acceptés.
 - Bloque les papiers venant hors délais c'est-à-dire après *les dates de soumission et d'envoi de version finale*.
 - Envoi les abstracts à l'agent PCChair et les papiers aux agents Reviewers.
 - Et à la fin, envoie les versions finales des agents Authors qui ont terminé l'enregistrement à la conférence vers l'agent Printer.
- L'agent temps réel Reviewer qui est externe au système:
 - Reçoit les abstracts provenant de l'agent PCMember.
 - Récupère les papiers à partir de l'agent DataBase pour les évaluer et établir un rapport (review) pour chaque papier.
 - Envoie les reviews à l'agent PCChair avant la fin de la *date d'examinassion* qui précède la *date de notification*.
- L'agent temps réel Author est un agent externe. Cet agent :
 - Reçoit le message CFP provenant de l'agent PCChair.
 - Envoie son papier et la version finale en cas d'acceptation de sa soumission initiale.
 - Fait l'enregistrement à la conférence,
 - et bien sûr à chaque étape, il faut qu'il respecte *les contraintes temporelles* imposées dans le CFP c'est-à-dire il ne faut pas dépasser *les dates de soumission, de camera ready et d'enregistrement*. Sinon il sera sanctionné.
- L'agent temps réel Printer est externe, il:
 - Reçoit les versions finales des papiers acceptés provenant de l'agent DataBase.
 - Imprime le *proceeding* et l'envoi à l'agent PCChair avant *la date de la conférence*.

3.2. Décomposition selon RT-Moise+ du Système de Gestion de Conférence

3.2.1. Spécification Structurale (SS)

Nous avons fait la SS (Figure 6.23) du système de gestion de conférence selon le modèle RT-Moise+ proposé. Cette structure se compose de deux groupes :

- Un groupe interne qui contient : PCChair, PCMember, DataBase.
- Un groupe externe qui contient ; Author, Researcher, Printer et Reviewer.

Chaque agent joue un rôle, mais pour les entités PC (Program Committee), Organisation, Participants, SGC, elles sont des rôles abstraits ou y à aucun agent qui peut les jouer.

Concernant les communications entre les agents, elles peuvent être inter ou intra groupe, et le type de ces communications est soit d'acointance, de communication, d'autorité ou de compatibilité comme indiqué dans [Hub02b].

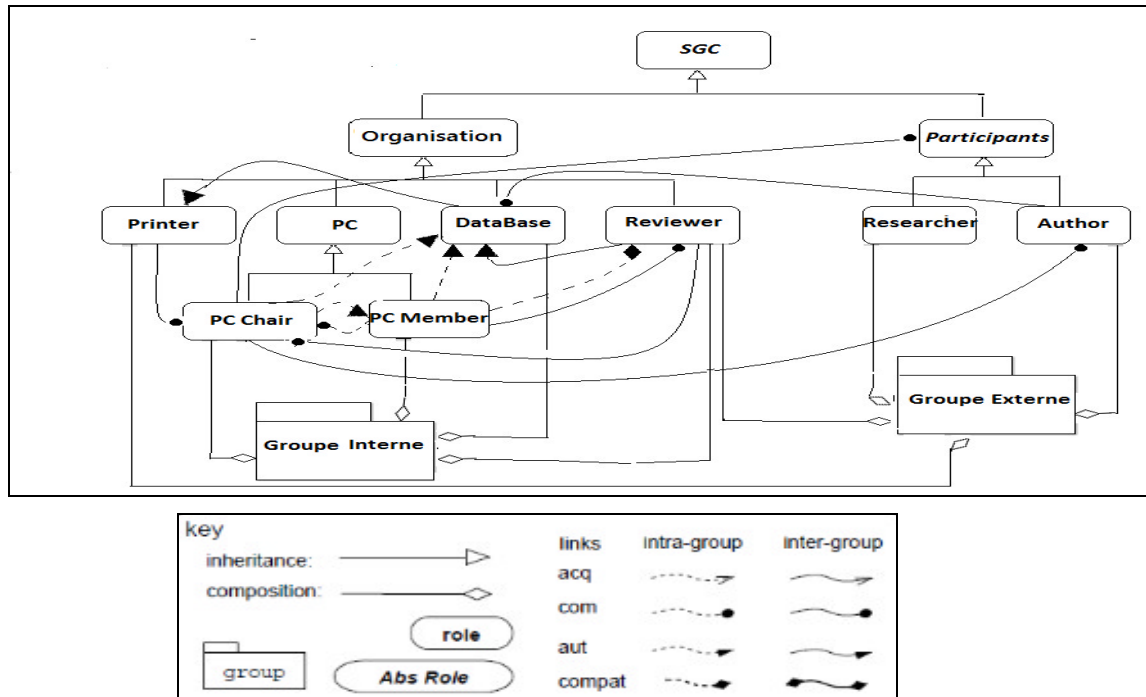


Figure 6.23. Structure du SGC selon RT-Moise+.

3.2.2. Spécification Fonctionnelle (SF)

La SF du SGC est illustrée par le schéma social présenté à la Figure 6.24 où on a un goal principal (g_0) à atteindre par l'exécution d'une série d'actions qui sont associées à d'autres goals/sub-goals (g_1 à g_{20}). Chaque action est liée à une mission particulière par la définition de plan. Plus de détails sur ce schéma sont donné ci-dessous.

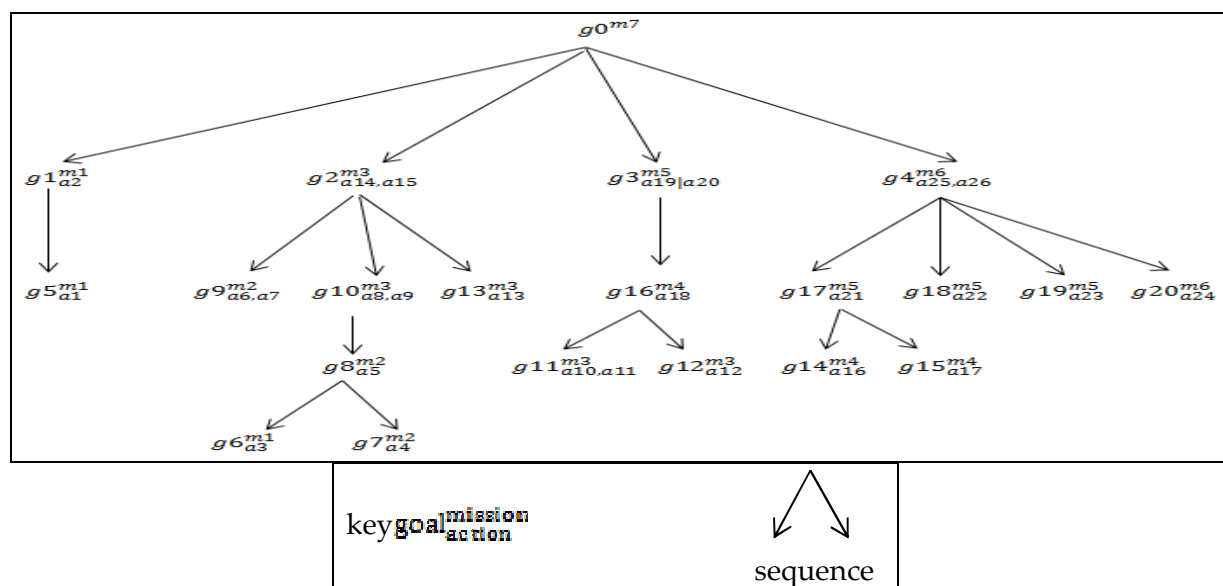


Figure 6.24. Exemple de Schéma Social du SGC selon RT-Moise+.

▪ Description du schéma Social de SGC

Cette description est établit à travers la définition de l'ensemble des missions, plans, buts, actions et contraintes associés à chaque agents du SGC. En effet, chaque agent possède un ensemble de missions à exécuter pour atteindre ses buts où chaque but (goal) est lié à un plan. Ce dernier se définit par rapport au goal précédent (en cas de besoin) et l'ensemble des actions à accomplir, Comme indiqué ci-après:

○ Agent temps réel PCChair

Mission Temps Real	Plan
M1=<g5,P5,{a1}> M2=<g7, P7, {a4}> M2=<g9,P9,{a6,a7}> M4=<g14, P14, {a16}> M4=<g15, P15, {a17}> M4=<g16, P16, {a18}> M5=<g3, P3, {a19, a20}, {DN}>	P3 (g3) = [g16], a19 a20. P5 (g5) = a1. P7 (g7) = a4. P9 (g9) = [g8], a6, a7. P14 (g14) = a16 P15 (g15) = a17. P16 (g16) = [g14],[g15],a18.
But	Action
g3: Inform Authors. g5: Create Conference. g7: Get Abs. g9: Partition Papers. g14: Collect Reviews. g15: Make Decision. g16: Select Papers.	a1: Send CFP to Participants. a4: Request Abs from DB. a6: Define Paper Domain. a7: Send Paper's Abs to PCM. a16: Get Reviews. a17: Study Reviews. a18: Final Decision. a19: Send Accept. a20: Send Refuse.
Contrainte Temporelle	
DN : Date of Notification.	

○ Agent temps réel Author

Mission Temps Real	Plan
M1=<g1, P1,{a2}, { Ds }> M5=<g17, P17, {a21}, { DCR}> M5=<g19, P19, {a23}, { DE }>	P1 (g1) = [g5], a2. P17 (g17) = a21. P19 (g19) = a23.
But	Action
g1: Participate to Conference. g17: Camera Ready Version. g19: Authors Registration.	a2: Send Paper to DB. a21: Send Final paper to DB. a23: Registered to Conference.
Contrainte Temporelle	
Ds : Date of Soumission. DCR : Date of Camera Ready. DE : Date of Registration.	

○ Agent PCMember

Mission	Plan
M3=<g10,P10,{a8,a9}>	P10 (g10) = a8,a9.
But	Action
g10: Assign Reviewer.	a8: Collect Abs from PCC. a9: Send Abs to Reviewer.

○ Agent temps réel Data-Base

Mission Temps Real	Plan
M1=<g6, P6, {a3}, { D_S }> M2=<g8, P8, {a5}> M3=<g13, P13, {a13}> M5=<g18, P18, {a22}, { D_{CR} }> M6=<g20, P20, {a24}, { D_E }>	P6 (g6) = a3. P8 (g8) = [g6], [g7], a5. P13 (g13) = [g11], [g12], a13. P18 (g18) = a22. P20 (g20) = a24.
But	Action
g6: Collect papers. g8: Distribute Abs of papers. g13: Distribute assigns papers. g18: Collect Final paper. g20: Distribute Final paper.	a3: Storage Paper. a5: Send Abs to PCC. a13: Send assigns Papers to Reviewer. a22: Storage Final paper. a24: Send Final Papers of registered authors.
Contrainte Temporelle	
D_S : Date of Soumission. D_{CR} : Date of Camera Ready. D_E : Date of Registration.	

○ Agent temps réel Printer

Mission Temps Real	Plan
M6=<g4,P4,{a25,a26}, { D _c }>	P4 (g4) = [g17], [g18],[g19],[g20],a25,a26.
But	Action
g4: Print Proceeding.	a25: Print Final Papers. a26: Send Proceeding to PCC.
Contrainte Temporelle	
D _c : Date of Conférence.	

○ Agent temps réel Reviewer

Mission Temps Real	Plan
M3=<g2,P2,{a14,a15}, { D_{EE} }> M3=<g11, P11, {a10, a11}>. M3=<g12, P12, {a12}>.	P2 (g2) = [g9], [g10], [g13], a14, a15. P12 (g12) = a12 P11 (g11) = a10, a11.
But	Action
g2: Review Paper. g11: Get assigns Abs. g12: Get assigns Papers.	a10: Collect Abs from PCM. a11: Send Accept. a12: Request assigns Papers from DB. a14: Study assigns Papers. a15: Send Reviews to PCC.
Contrainte Temporelle	
D_{EE} : Date of End Examination.	

3.3. Modélisation AUML du Système de Gestion de Conférence

3.3.1. Diagramme de classe du Système de Gestion de Conférence

Le diagramme de classe ci-dessous (Figure 6.25) est utilisé pour représenter la structure statique du système multi-agent temps réel correspondant au SGC à travers la description des différentes classes des agents.

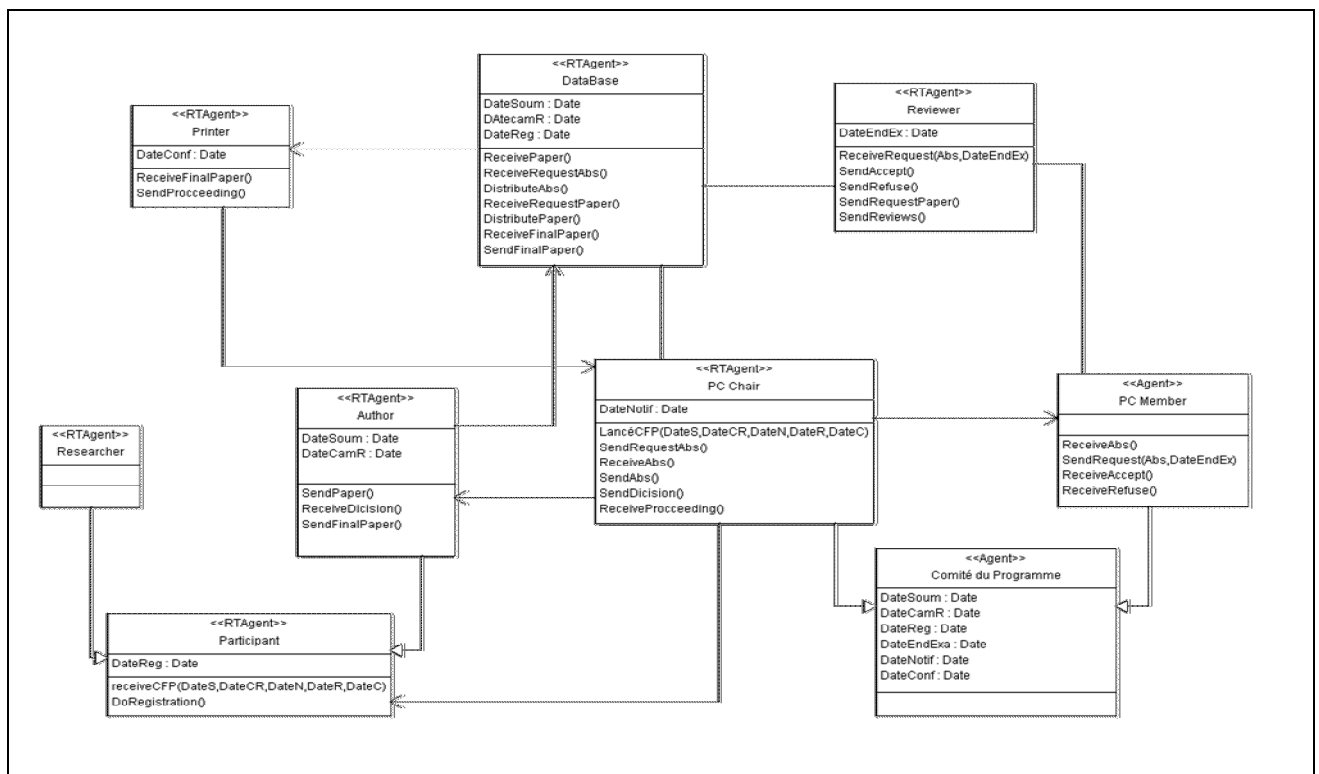


Figure 6.25. Diagramme de Classe du SGC.

3.3.2. Diagramme de cas d'utilisation du Système de Gestion de Conférence

Le diagramme de cas d'utilisation suivant (Figure 6.26) décrit les différentes fonctionnalités des agents PCChair, DataBase, PCMember, Author, Reviewer, et Printer.

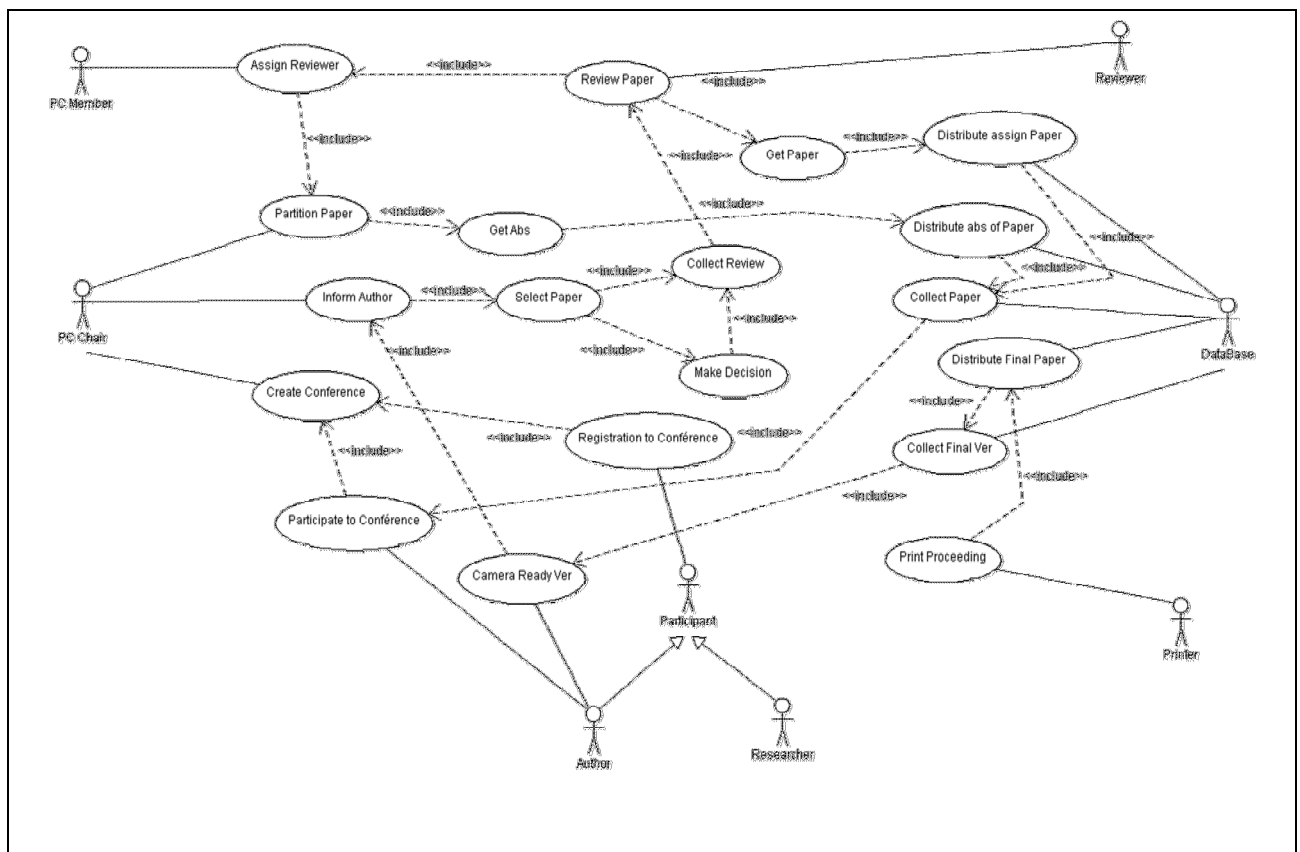


Figure 6.26. Diagramme de Cas d'utilisation du SGC.

3.3.3. Diagramme de Communication du Système de Gestion de Conférence

Les communications des agents du SMATR sont décrites par le diagramme de communication de la Figure 6.27. Ce diagramme décrit la séquence des messages envoyés et reçu entre les agents pour la création d'une conférence.

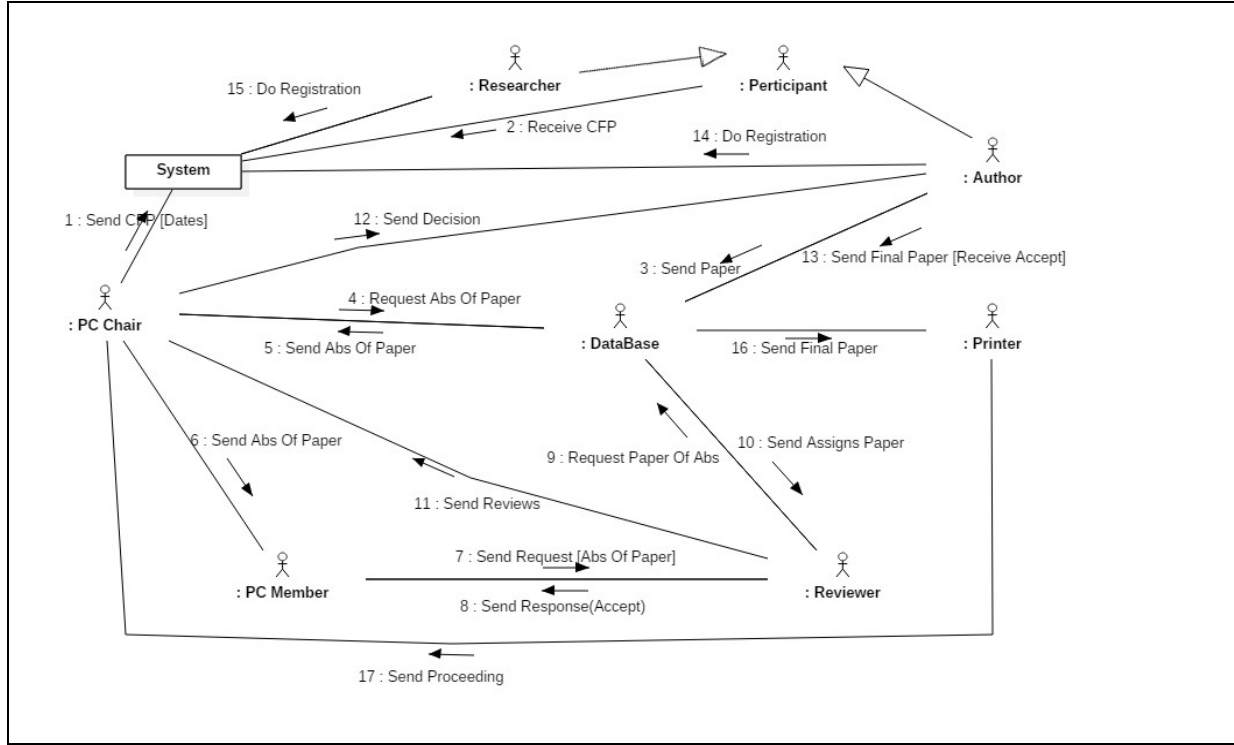


Figure 6.27. Diagramme de Communication du SGC.

3.3.4. Diagrammes d'état transition du Système de Gestion de Conférence

Les diagrammes d'états-transitions suivants (Figures 6.28, 6.29, 6.30, 6.31, 6.32, 6.33) décrivent respectivement le comportement interne des agents PCChair, DataBase, PCMember, Author, Reviewer, et Printer.

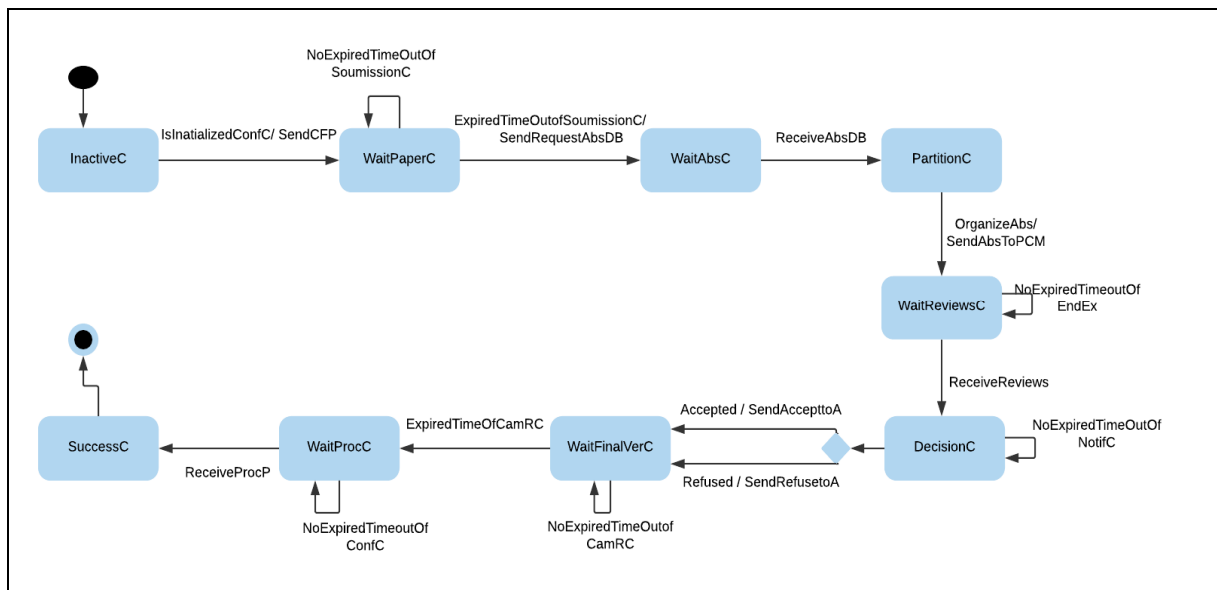


Figure 6.28. Diagramme d'état transition de l'agent PCChair.

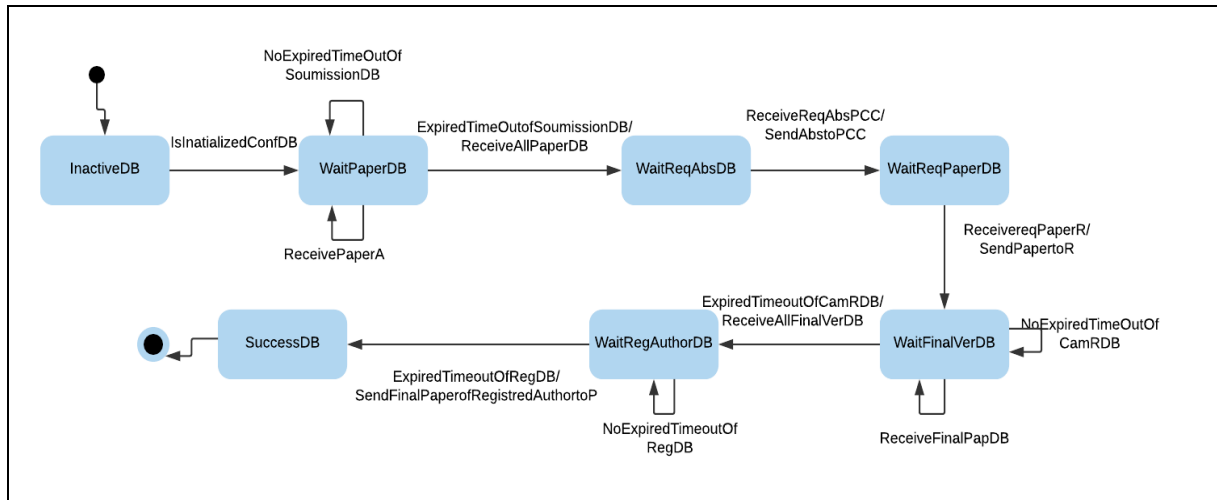


Figure 6.29. Diagramme d'état transition de l'agent Data-Base.

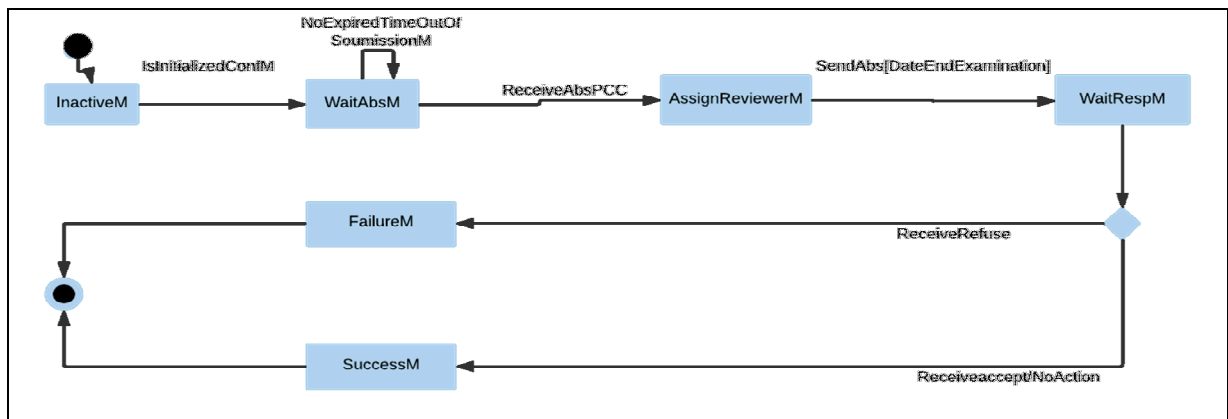


Figure 6.30. Diagramme d'état transition de l'agent PCMember.

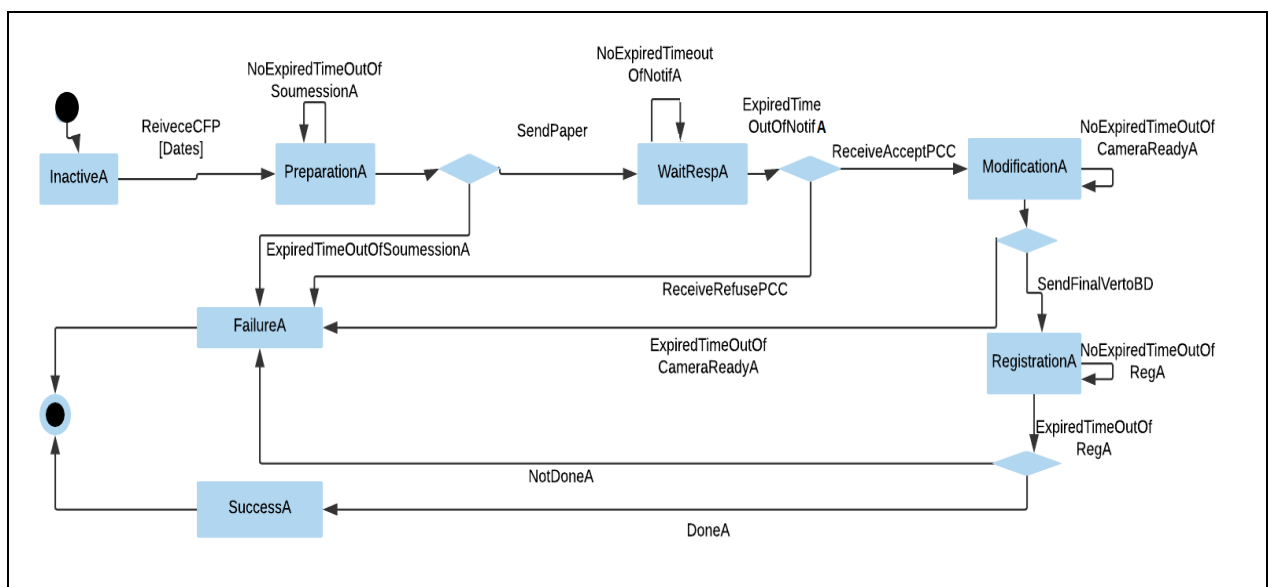


Figure 6.31. Diagramme d'état transition de l'agent Author.

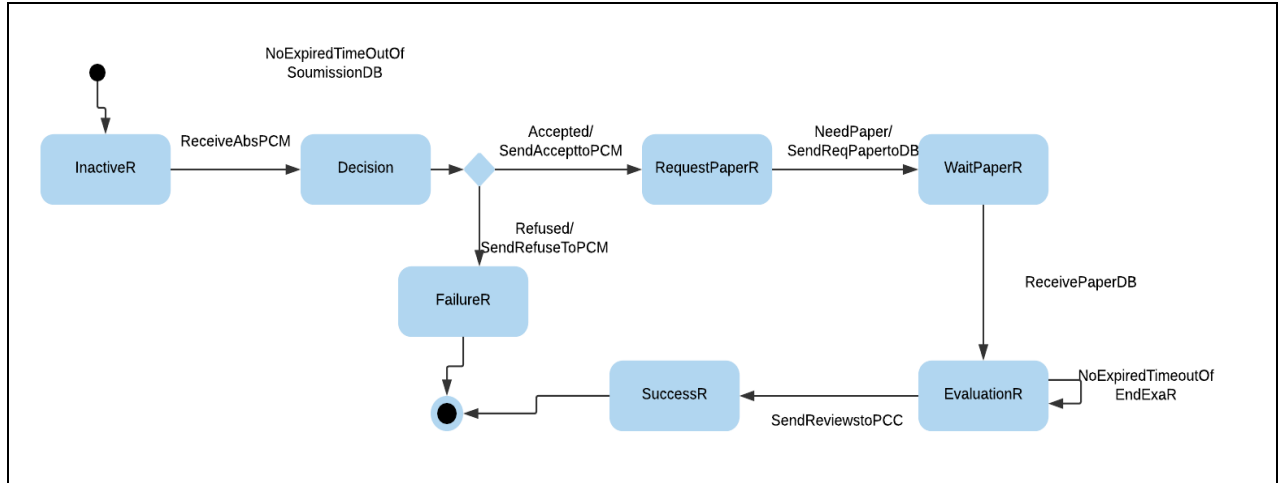


Figure 6.32. Diagramme d'état transition de l'agent Reviewer.

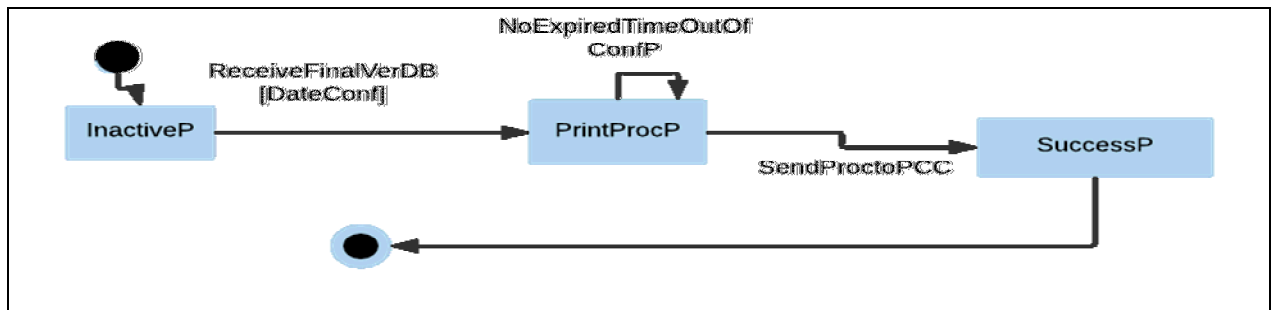


Figure 6.33. Diagramme d'état transition de l'agent Printer.

3.3.5. Diagramme de Protocole du Système de Gestion de Conférence

Nous avons six agents qui vont communiquer pour créer une conférence (Figure 6.34). Au début le PCChair lance la conférence par un CFP qui l'envoie aux différents participants. Après les agents Authors concernés par ce CFP vont préparer des papiers et les envoyer à l'agent DataBase, ce dernier peut les recevoir mais avant l'expiration de *la date prévue de soumission*.

Lorsque la *date de soumission* sera expirée, le PCChair demande les abstracts des papiers soumis auprès du DataBase qui les lui envoie. Ensuite, le PCChair organise ces abstracts et les transmet aux agents PCMembers. Ces derniers vont chercher des Reviewers pour les examiner.

Chaque Reviewer pourra à son tour accepter ou refuser les demandes d'examen reçues, mais s'il acceptera, il va devoir demander les papiers concernés depuis le DataBase, qui va automatiquement les lui transmettre. Une fois les avoir examinés le Reviewer va faire son rapport (Reviews) qui l'envoie au PCChair afin que ce dernier décide d'accepter ou rejeter les papiers concernés et envoie la décision aux auteurs correspondant.

A la fin, chaque agent Author qui reçoit l'acceptation de son papier va préparer la version finale de son travail et fait l'enregistrement avant l'expiration *des dates* définies. Une fois le temps d'enregistrement soit expiré le DataBase va transmettre les papiers des Authors à l'agent Printer pour préparer le *proceeding* des papiers de la conférence qu'il envoie au PCChair avant *la date prévue de la conférence*.

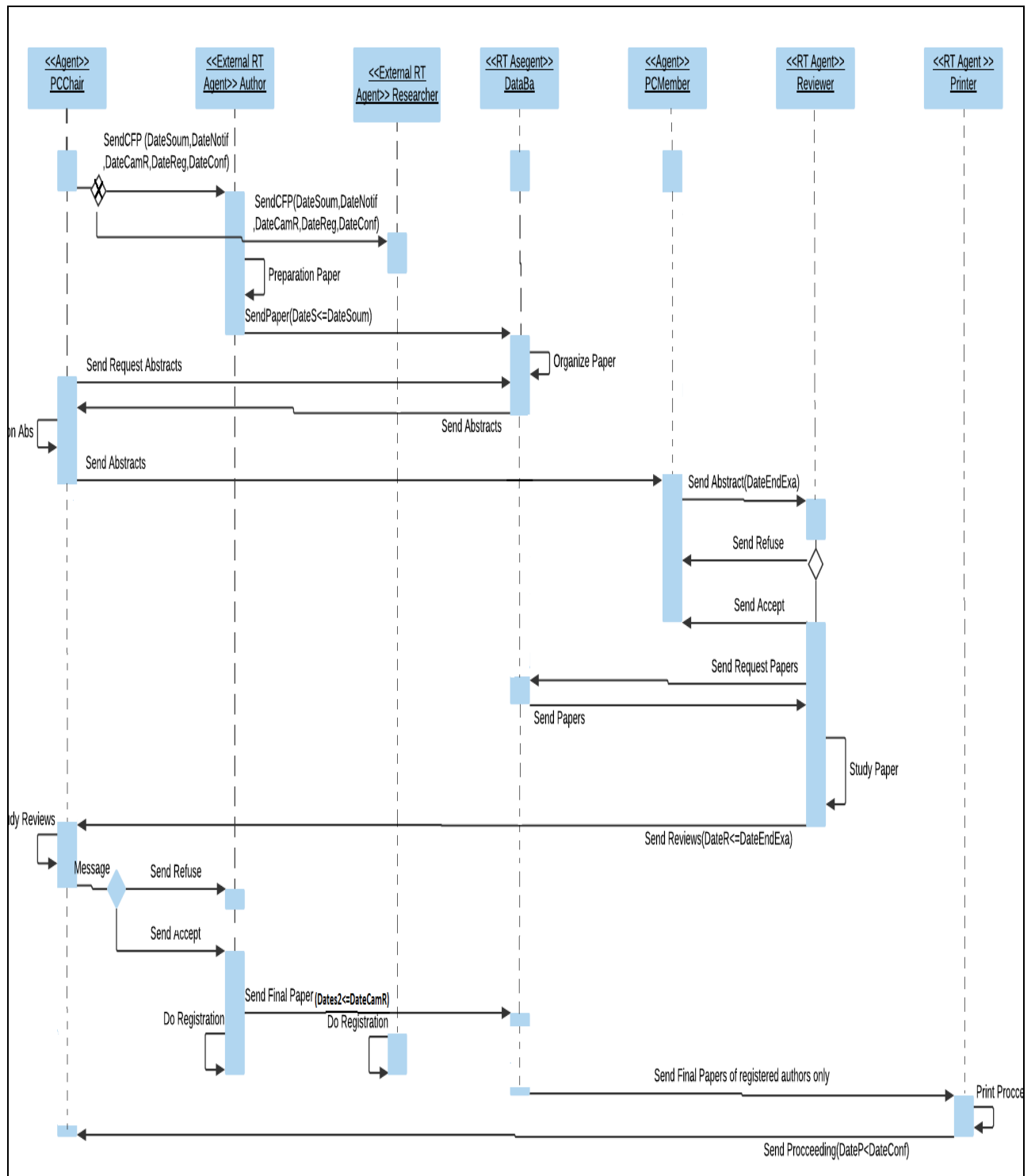


Figure 6.34. Diagramme de protocole du SGC.

3.3.6. Diagrammes de Temps du Système de Gestion de Conférence

Ici, nous donnons les différents diagrammes temporels (Figure 3.35 à Figure 6.40) décrivant l'enchaînement des activités de chaque agent selon la ligne du temps. Ces représentations nous permettent de bien voir l'ensemble des *contraintes temporelles* au quel chaque agent est amené à respecter.

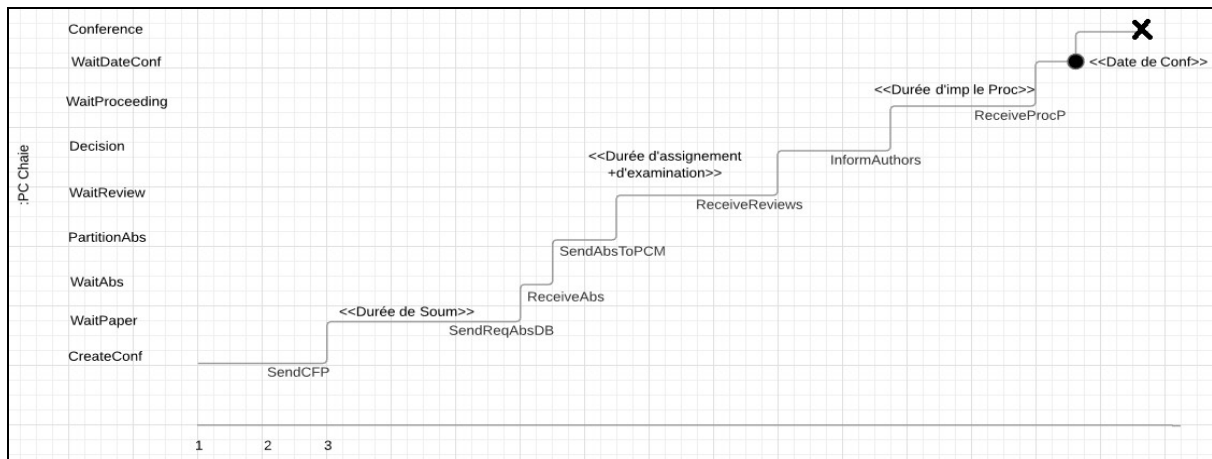


Figure 6.35. Diagramme de temps de l'agent PCChair.

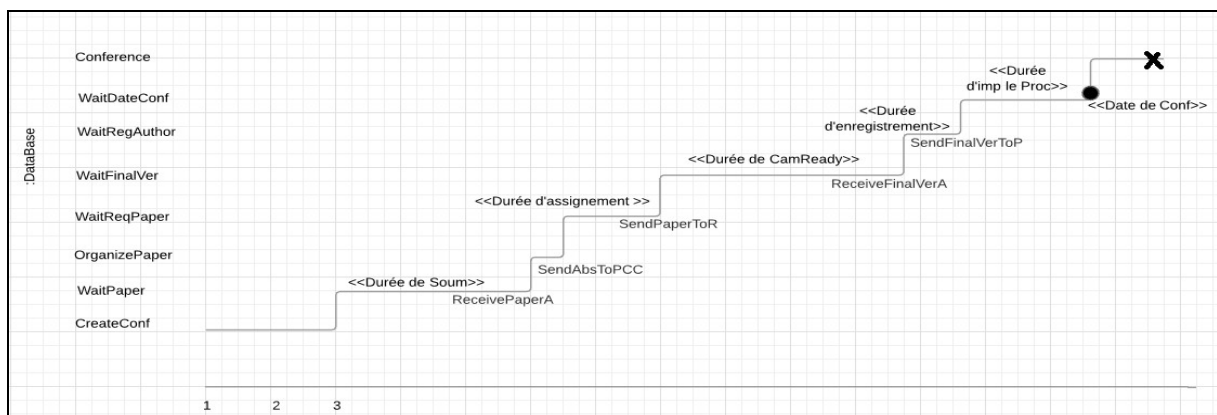


Figure 6.36. Diagramme de temps de l'agent DataBase.

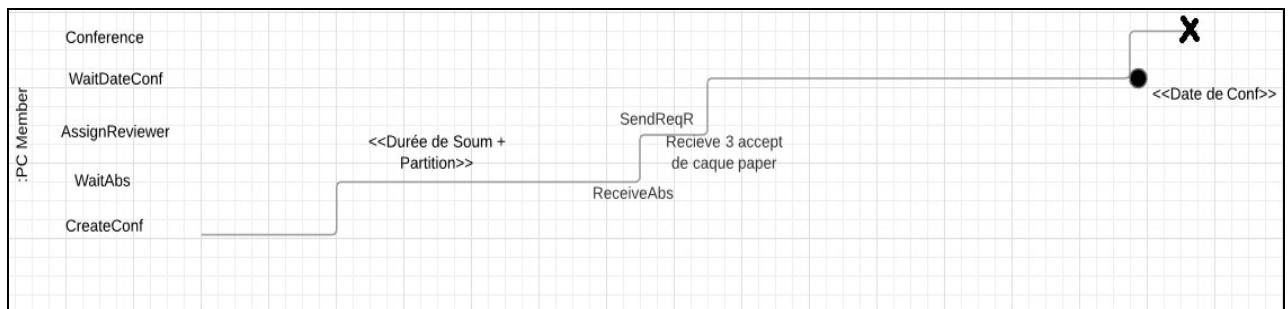


Figure 6.37. Diagramme de temps de l'agent PCMember.

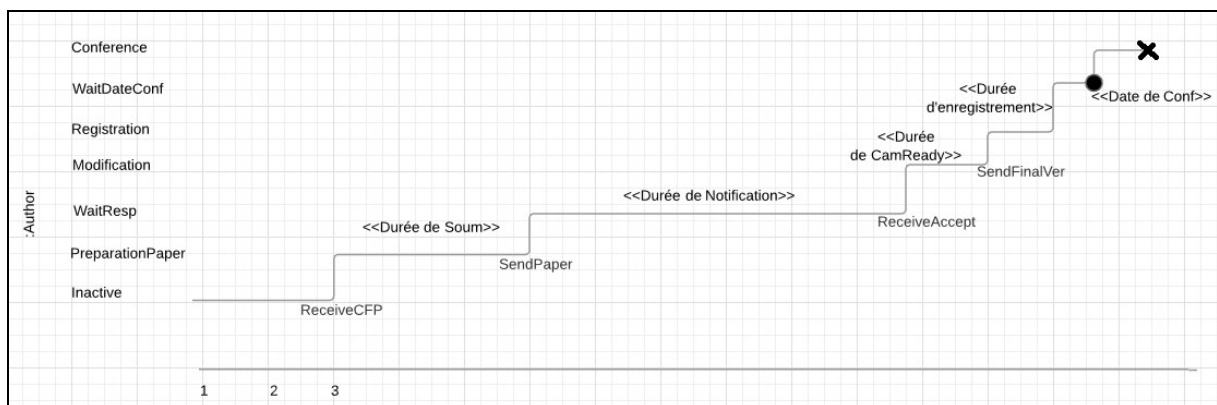


Figure 6.38. Diagramme de temps de l'agent Author.



Figure 6.39. Diagramme de temps de l'agent Reviewer.

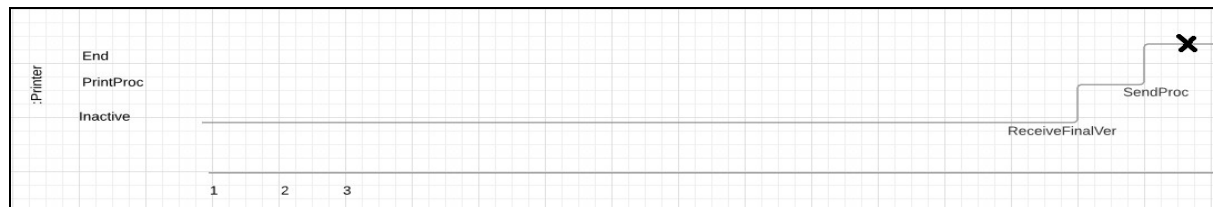


Figure 6.40. Diagramme de temps de l'agent Printer.

3.4. Application du Processus de Translation

Dans cette section, on présente le processus de translation appliqué au framework formel décrit précédemment.

On commence par définir le comportement interne des agents, selon ce qui a été défini au sein des diagrammes d'états-transitions associés à chacun des agents du SMATR. Comme on a six diagrammes, un pour chaque agent, on doit définir six modules. Ces modules sont:

- PCCHAIR-INTRA-BEHAVIOR.
- DATABASE-INTRA-BEHAVIOR.
- AUTHOR-INTRA-BEHAVIOR.
- PCMEMBER-INTRA-BEHAVIOR.
- REVIEWER-INTRA-BEHAVIOR.
- PRINTER-INTRA-BEHAVIOR.

La Figure 6.41 décrit le module **AUTHOR-INTRA-BEHAVIOR** qui correspond au comportement interne de l'agent temps réel Author. Les autres modules Maude du reste des agents du SMATR ont la même structure que celui-ci.

Le module **SYSTEM_USE_CASES** qui correspond au module principal du framework, implémente l'ensemble des fonctionnalités du SMATR ainsi que les comportements individuels et collectifs des agents à travers un ensemble de règles de réécriture RT-Maude. Les contraintes temporelles sont décrites par des règles de réécriture temporisé (tick rules). Ce module importe les modules **AGENTS**, **REAL-TIME-AGENTS**, et **AGENT-INTER-BEHAVIORS**. Pour des raisons d'espaces la Figure 6.42 montre qu'une partie du module **SYSTEM_USE_CASES** que nous décrivons ci-après :

- Pour exprimer l'existence d'un événement, nous avons défini le message *Event*.
- Quant aux fonctions `ExpireTimerOfSoum1`, `ExpireTimerOfReg`, `ExpireTimerOfCamR`, `ExpireTimerOfNotif`, `ExpireTimerOfEndExam`, `ExpireTimerOfConf`, `TimerOfSoum1`, `TimerOfReg`, `TimerOfCamR`, `TimerOfNotif`, `TimerOfEndExam` et `TimerOfConf`, ils sont utilisées pour indiquer l'expiration des dates limites de : *Soumission*, *Enregistrement*, *CameraReady*, *Notification*, *Examination* et *de Conference*.

- Dans notre cas d'étude, le comportement dynamique des agents internes et externes au système (c.à.d. le fait de changer d'état) est implémenté par un ensemble de règles de réécriture "instantanées" et de six règles temporisées ([tick1], [tick2], [tick3], [tick4], [tick5], [tick6]). Ces règles temporisées permettent d'assurer l'écoulement du temps dans le système.
- Les deux règles de réécritures [ExpireTimeOfSoum01] et [ExpireTimeOfSoum02], permettent d'informer les agents temps réel PCChair, DataBase et Author de l'expiration de *la date limite* qu'a fixée chacun d'eux en déclenchant un événement. Ce dernier va leurs permettre de changer d'état.
- Les autres règles de réécriture (instantanées) ont été classées selon le séquençement des interactions explicitées dans le diagramme de protocole AUML (Figure 6.34) :
 - En effet, lorsqu'un des agents veut envoyer un message vers ses accointant, il va exécuter la règle qui lui correspond, par exemple la règle [Send-PCC1]. Chaque règle exprime une boucle conditionnelle dans laquelle l'agent envoie le message à chaque élément de sa liste d'accointance par *SendMessage* en déclenchant un événement et cela en utilisant les deux opérations : *HeadA* et *TailA* qui déterminent respectivement l'entête et le reste de la liste.
 - Cependant, quand un des agents PCChair, DataBase, Reviewer, Printer, Author, ou PCMember, reçoit un message, il le met dans son MailBox comme illustré dans la règle [Recieve-Auth1], et passe à l'état suivant après la mise à jour de sa liste d'accointances à travers l'opération *UpdateAcqList*.

```

(fmod AUTHOR-INTRA-BEHAVIOR is
  protecting AGENT-STATE-CHART .
  protecting GOAL .
  ***** User Part
  ***** Goal
    ops ParticipatetoConference CameraReadyVersion AuthorsRegistration EndGoalA : -> Goal .
  ***** State
    ops InactiveA PreparationA WaitRespA ModificationA RegistrationA FailureA SuccessA : -> NameState .
    ops AgentState (InactiveA)
      AgentState (PreparationA)
      AgentState (FailureA)
      AgentState (SuccessA) : -> State .

  ***** Condition and Actions
    ops ReceiveCFP FinishedPaper ReceiveRefusePCC FinishedCameraPaper ReceiveAcceptPCC NotDoneA DoneA : -> Condition .

    ops SendPaper SendFinalVertoDB RegisteredToConference : -> Action .

  ***** Target State
    eq TargetState(AgentState (InactiveA), ReceiveCFP) =
      AgentState(PreparationA) .
    ...
  ***** Action To Accomplish
    eq ActionToAccomplish(AgentState(PreparationA), FinishedPaper) =
      SendPaper .
    ...
  ***** NextGoal
    eq NextGoal(ParticipatetoConference, SendPaper) = CameraReadyVersion .
    ...
  ***** Action Type
    eq IsSendingAction(SendPaper) = true .
    ...
endfm)

```

Figure 6.41. Le module fonctionnel AUTHOR-INTRA-BEHAVIOR.

```

(comod SYSTEM_USE_CASES is
  including AGENTS .
  including REAL-TIME-AGENTS .
  including AGENT-INTER-BEHAVIORS .

***** User Part
  subsort String < Identifier .
  ops Event : Identifier State Condition -> Msg .
  op HavePB : -> Msg .
  op CFP : RTConstraintList -> Msg .
  ops ExpireTimerOfSoum ExpireTimerOfSoum1 ExpireTimerOfNotif ExpireTimerOfEndExa : Bool -> Msg .
  ...

  vars A A1 A2 A3 A4 A5 A6 : Identifier .      vars S S1 S2 S3 S4 S5 : State .
  vars TimeOutOfSoum TimeOutOfSoum1 TimeOutOfReg TimeOutOfCamR TimeOutOfNotif TimeOutOfEndExa TimeOutOfConf : Time .
  ...

  eq UpDateAcqList(MB) = if MB == EmptyMailBox then EmptyAcquaintanceList
                        else GetIdent(FrontMBox(MB)) : UpDateAcqList(RemoveMBox(MB))
                        fi .

-----|Expire Timer Of Soumission1|-----
cr1 [ExpireTimeOfSoum1] : TimerOfSoum1(TimeOutOfSoum1)
  < A : Author | CurrentState : S, RTConstList : RTCL > < A1 : PCChair | CurrentState : S1 >
  < A2 : DataBase | CurrentState : S2 > < A3 : Reviewer | > < A4 : Printer | > < A6 : PCMember | >
  => ExpireTimerOfSoum1(true)
  < A : Author | CurrentState : S, RTConstList : RTCL > < A1 : PCChair | CurrentState : S1 >
  < A2 : DataBase | CurrentState : S2, RTConstList : RTCL > < A3 : Reviewer | > < A4 : Printer | > < A6 : PCMember | >
  if (TimeOutOfSoum1 == 0).

r1 [ExpireTimeOfSoum12] : ExpireTimerOfSoum1(true)
  < A : RTAgent | CurrentState : S, PlayRole : Author >
  < A1 : RTAgent | CurrentState : S1, PlayRole : PCChair >
  < A2 : RTAgent | CurrentState : S2, PlayRole : DataBase >
  => Event (A, S, FinishedPaper)
  Event (A1, S1, ExpiredTimeOfSoumissionC)
  Event (A2, S2, ExpiredTimeOfSoumissionDB)
  < A : RTAgent | CurrentState : S, PlayRole : Author >
  < A1 : RTAgent | CurrentState : S1, PlayRole : PCChair >
  < A2 : RTAgent | CurrentState : S2, PlayRole : DataBase > .
  ...

-----|Sending action (PCChair)|-----
cr1 [Send-PCC1] : Event(A, S, Cond)
  CFP (RTS : RTN : RTCA : RTR : RTCO )
  < A : PCChair | CurrentState : S, AcqList : ACL, AgentGoal : Go, RTConstList : RTPC >
  => < A : PCChair | CurrentState : TargetState(S,Cond), AcqList : TailA(ACL),
      AgentGoal : NextGoal(Go, ActionToAccomplish(S, Cond)), RTConstList : RTN >
  SendMessage(A : ActionToAccomplish(S, Cond) : HeadA(ACL))
  Event(HeadA(ACL), CorrespondingState(S), CorrespondingCondition(ActionToAccomplish(S, Cond)))
  CFP (RTS : RTN : RTCA : RTR : RTCO )
  if (IsSendingAction(ActionToAccomplish(S, Cond)) == true) and (Cond == IsInitializedConfC).

-----|Receiving action (Author)|-----
cr1 [Receive-Auth1] : Event(A, S, Cond)
  SendMessage(A1 : Cont : A)
  CFP (RTS : RTN : RTCA : RTR : RTCO )
  < A : Author | CurrentState : S, AcqList : ACL, RTConstList : RTCL >
  => < A : Author | CurrentState : TargetState(S, Cond), AcqList : ACL,
      RTConstList : (RTS : RTCA : RTR) >
  if (Cond == ReceiveCFP) .

```

```

crl [Receive-Auth2] : Event(A, S, Cond)
    SendMessage(A1 : Cont : A)
    < A : Author | CurrentState : S, AcqList : ACL >
    => < A : Author | CurrentState : TargetState(S, Cond), AcqList : ACL >
    if (Cond == ReceiveRefusePCC)
    or (Cond == ReceiveAcceptPCC) .

-----| Target State DataBase|-----
crl [Target-DataBase1] : Event(A, S, Cond)
    < A : DataBase | CurrentState : S, AcqList : ACL >
    => < A : DataBase | CurrentState : TargetState(S,Cond), AcqList : ACL >
    if (Cond == IsInitializedConfDB) .

-----|Target State PCMember|-----
crl [Target-PCMember1] : Event(A, S, Cond)
    < A : PCMember | CurrentState : S, AcqList : ACL >
    => < A : PCMember | CurrentState : TargetState(S,Cond), AcqList : ACL >
    if (Cond == IsInitializedConfM) .
...
endtom)

```

Figure 6.42. Le module temporisé orienté objet SYSTEM_USE_CASE.

3.5. Validation de la Description Générée

Pour valider cet exemple de Système de Gestion de Conférence, on a choisi *deux scénarios d'interaction* :

- Dans le premier scénario (Figure 6.43), tous les agents atteindront leurs buts, ce qui permettra d'accomplir le but global du SMATR. Dans ce cas les agents finiront dans leurs états de succès (*success*) comme illustré à la Figure 6.44.
- Par contre dans le deuxième scénario (Figure 6.45), la conférence a été créée mais un problème a survécu avant que les reviewers ont terminé leurs examinations des papiers. Dans ce cas, l'ensemble des agents subissent un échec et se terminent dans leurs états *failure*. Alors que l'agent Printer qui est sollicité qu'à la fin, reste dans son état Inactive. Comme indiqué dans la Figure 6.46.

Pour implémenter ces deux scénarios, on a défini le module temporisé orienté objet CONFERENCE_MANAGEMENT_SYSTEM qui importe le module RTMOISE-BASED-MAS comme illustré dans la Figure 6.43.

```

(tomod CONFERENCE_MANAGEMENT_SYSTEM is
    extending RTMOISE-BASED-MAS .
    op Initstate : -> GlobalSystem .

*****CONFIGURATION OF "CMS" *****
*****| First Scenario |*****

eq Initstate = { Event("PCChair", AgentState(InactiveC), IsInitializedConfC)
    Event("DataBase", AgentState(InactiveDB), IsInitializedConfDB)
    Event("PCMember", AgentState(InactiveM), IsInitializedConfM)
    Event("PCChair", AgentState(PartitionC), OrganizeAbs)
    Event("PCMember", AgentState(AssignReviewerM), FindRev)
    Event("Reviewer", AgentState(DecisionR), Accepted)
    Event("Reviewer", AgentState(RequestPaperR), NeedPaperR)
    CFP (20 : 5 : 10 : 30 : 20)

    < "PCChair" : PCChair | PlayRole : PCChair, CurrentState : AgentState(InactiveC),
        AgentGoal : CreateConference, GrType : Interne, AcqList : ("Author" : "DataBase" : "PCMember" : "Author"),
        RTConstList : EmptyRTConstraintList, MBox : EmptyMailBox >
    < "Author" : Author | PlayRole : Author, CurrentState : AgentState(InactiveA),
        AgentGoal : ParticipateToConference, GrType : Externe,
        RTConstList : EmptyRTConstraintList, MBox : EmptyMailBox, AcqList : ("DataBase" : "DataBase") >
    < "DataBase" : DataBase | PlayRole : DataBase, CurrentState : AgentState(InactiveDB),
        AgentGoal : Collectpapers, GrType : Interne,
        RTConstList : EmptyRTConstraintList, MBox : EmptyMailBox, AcqList : ("PCChair" : "Reviewer" : "Printer" ) >
    < "PCMember" : PCMember | PlayRole : PCMember, CurrentState : AgentState(InactiveM),
        AgentGoal : AssignReviewer, GrType : Interne,
        MBox : EmptyMailBox, AcqList : ("Reviewer" ) >
    < "Reviewer" : Reviewer | PlayRole : Reviewer, CurrentState : AgentState(InactiveR),
        AgentGoal : GetassignsAbs, GrType : Interne,
        RTConstList : EmptyRTConstraintList, MBox : EmptyMailBox, AcqList : ("PCMember" : "DataBase" : "PCChair") >
    < "Printer" : Printer | PlayRole : Printer, CurrentState : AgentState(InactiveP),
        AgentGoal : PrintProceeding, GrType : Interne,
        RTConstList : EmptyRTConstraintList, MBox : EmptyMailBox, AcqList : ("PCChair") >
    TimerOfSouml(20) TimerOfReg(30) TimerOfCamR(10) TimerOfNotif(5) TimerOfEndExa(5) TimerOfConf(20)
    } .

```

Figure 6.43. Configuration Initiale du Scenario 1.

```

Result `[ClockedSystem`] :
{< "Author" : Author | AcqList : EmptyAcquaintanceList, AgentGoal : EndGoalA,
  CurrentState : AgentState(SuccessA), GrType : Externe, MBox : EmptyMailBox,
  PlayRole : Author, RTConstList : (20 : 10 : 30) > < "DataBase" : DataBase |
  AcqList : EmptyAcquaintanceList, AgentGoal : EndGoalDB, CurrentState :
  AgentState(SuccessDB), GrType : Interne, MBox : EmptyMailBox, PlayRole :
  DataBase, RTConstList : (20 : 10 : 30) > < "PCChair" : PCChair | AcqList :
  EmptyAcquaintanceList, AgentGoal : EndGoalC, CurrentState : AgentState(
  SuccessC), GrType : Interne, MBox : EmptyMailBox, PlayRole : PCChair,
  RTConstList : 5 > < "PCMember" : PCMember | AcqList :
  EmptyAcquaintanceList, AgentGoal : EndGoalM, CurrentState : AgentState(
  SuccessM), GrType : Interne, MBox : EmptyMailBox, PlayRole : PCMember > <
  "Printer" : Printer | AcqList : EmptyAcquaintanceList, AgentGoal : EndGoalP,
  CurrentState : AgentState(SuccessP), GrType : Interne, MBox : EmptyMailBox,
  PlayRole : Printer, RTConstList : 20 > < "Reviewer" : Reviewer | AcqList :
  EmptyAcquaintanceList, AgentGoal : EndGoalR, CurrentState : AgentState(
  SuccessR), GrType : Interne, MBox : EmptyMailBox, PlayRole : Reviewer,
  RTConstList : 5 > } in time 90

```

Figure 6.44. Exécution de la configuration Initiale du Scenario 1.

```

*****| Second Scenario |*****
eq Initstate = { Event("PCChair", AgentState(InactiveC), IsInitializedConfC)
  Event("DataBase", AgentState(InactiveDB), IsInitializedConfDB)
  Event("PCMember", AgentState(InactiveM), IsInitializedConfM)
  Event("PCChair", AgentState(PartitionC), OrganizeAbs)
  Event("PCMember", AgentState(AssignReviewerM), FindRev)
  Event("Reviewer", AgentState(DecisionR), Accepted)
  Event("Reviewer", AgentState(RequestPaperR), NeedPaperR)
  HavePB
  CFP (20 : 5 : 10 : 30 : 20)

  < "PCChair" : PCChair | PlayRole : PCChair, CurrentState : AgentState(InactiveC),
    AgentGoal : CreateConference, GrType : Interne, AcqList : ("Author" : "DataBase" : "PCMember" : "Author"),
    RTConstList : EmptyRTConstraintList, MBox : EmptyMailBox >
  < "Author" : Author | PlayRole : Author, CurrentState : AgentState(InactiveA),
    AgentGoal : ParticipateToConference, GrType : Externe,
    RTConstList : EmptyRTConstraintList, MBox : EmptyMailBox, AcqList : ("DataBase" : "DataBase") >
  < "DataBase" : DataBase | PlayRole : DataBase, CurrentState : AgentState(InactiveDB),
    AgentGoal : Collectpapers, GrType : Interne,
    RTConstList : EmptyRTConstraintList, MBox : EmptyMailBox, AcqList : ("PCChair" : "Reviewer" : "Printer") >
  < "PCMember" : PCMember | PlayRole : PCMember, CurrentState : AgentState(InactiveM),
    AgentGoal : AssignReviewer, GrType : Interne,
    MBox : EmptyMailBox, AcqList : ("Reviewer") >
  < "Reviewer" : Reviewer | PlayRole : Reviewer, CurrentState : AgentState(InactiveR),
    AgentGoal : GetassignsAbs, GrType : Interne,
    RTConstList : EmptyRTConstraintList, MBox : EmptyMailBox, AcqList : ("PCMember" : "DataBase" : "PCChair") >
  < "Printer" : Printer | PlayRole : Printer, CurrentState : AgentState(InactiveP),
    AgentGoal : PrintProceeding, GrType : Interne,
    RTConstList : EmptyRTConstraintList, MBox : EmptyMailBox, AcqList : ("PCChair") >
  TimerOfSoum1(20) TimerOfReg(30) TimerOfCamR(10) TimerOfNotif(5) TimerOfEndExa(5) TimerOfConf(20)
  } .

```

Figure 6.45. Configuration Initiale du Scenario 2.

```

Result `[ClockedSystem`] :
{< "Author" : Author | AcqList : "DataBase", AgentGoal : CameraReadyVersion,
  CurrentState : FailureA, GrType : Externe, MBox : EmptyMailBox, PlayRole :
  Author, RTConstList : (20 : 10 : 30) > < "DataBase" : DataBase | AcqList : (
  "Reviewer" : "Printer"), AgentGoal : Distributeassignspapers, CurrentState :
  FailureDB, GrType : Interne, MBox : EmptyMailBox, PlayRole : DataBase,
  RTConstList : (20 : 10 : 30) > < "PCChair" : PCChair | AcqList : "Author",
  AgentGoal : CollectReviews, CurrentState : FailureC, GrType : Interne, MBox :
  EmptyMailBox, PlayRole : PCChair, RTConstList : 5 > < "PCMember" : PCMember |
  AcqList : "Reviewer", AgentGoal : AssignReviewer, CurrentState : FailureM,
  GrType : Interne, MBox : EmptyMailBox, PlayRole : PCMember > < "Printer" :
  Printer | AcqList : "PCChair", AgentGoal : PrintProceeding, CurrentState :
  AgentState(InactiveP), GrType : Interne, MBox : EmptyMailBox, PlayRole :
  Printer, RTConstList : EmptyRTConstraintList > < "Reviewer" : Reviewer |
  AcqList : ("PCMember" : "DataBase" : "PCChair"), AgentGoal : GetassignsAbs,
  CurrentState : FailureR, GrType : Interne, MBox : EmptyMailBox, PlayRole :
  Reviewer, RTConstList : EmptyRTConstraintList > Event("DataBase", AgentState(
  WaitPaperDB), ReceivePaperA) Event("PCMember", AgentState(AssignReviewerM),
  FindRev) Event("Reviewer", AgentState(DecisionR), Accepted) Event("Reviewer",
  AgentState(RequestPaperR), NeedPaperR) } in time 20

```

Figure 6.46. Exécution de la configuration Initiale du Scenario 2.

3.6. Vérification de la Description Générée

Dans cette partie, on va décrire le processus de vérification formelle de notre framework RT-Maude du SGC validé précédemment à travers le modèle checker offert par RT-Maude. On commence par définir un ensemble de propriétés liées au *comportement individuel* des agents de notre SMATR dans les paragraphes ci-dessous.

3.6.1. Propriétés du comportement individuel

Huit propriétés sont proposées pour procéder à la vérification des agents du système. Parmi ces propriétés, *les propriétés 1 et 2* tentent de vérifier le comportement de l'agent PCChair. Alors que *les Propriété 3 et 4*, vérifient le comportement de l'agent Author. La *Propriétés 5*, quant à elle, vérifie le comportement de l'agent DataBase, et la *Propriété 6* fait de même pour l'agent PCMember. Tandis que, *les Propriétés 7 et 8* le font pour les agents Reviewer et Printer respectivement.

▪ Propriété 1

Formule LTL : $PCChair-In-InactiveC \mid \rightarrow O \text{ } PCChair -In-WaitPaperC$

- **Description:** En démarrant la vérification à partir de la configuration initiale *initialStateC* (Figure 6.48), cette propriété exprime le fait que si l'agent PCChair est dans son état *InactiveC*, la propriété *PCChair-In-WaitPaperC*, exprimant que PCChair sera dans l'état *WaitPaperC*, est vraie dans le prochain état. Noter que *O* est l'opérateur temporel *Next* et que $\rightarrow$ est l'opérateur temporel *Implies*.

▪ Propriété 2

Formule LTL: $PCChair-In-InactiveC \mid \rightarrow \diamond PCChair -In-WaitAbsC$

- **Description:** En démarrant la vérification à partir de la configuration initiale *initialStateC1* (Figure 6.48) cette propriété tente de vérifier que si l'agent PCChair est dans son état *InactiveC*, il sera éventuellement dans son état *WaitAbsC*. Noter que $\diamond$ est l'opérateur temporel *Eventually*.

▪ Propriété 3

Formule LTL : $Author-In-InactiveA \mid \rightarrow O \text{ } Author -In-PreparationA$

- **Description:** En démarrant la vérification à partir de la configuration initiale *initialStateA* (Figure 6.48), cette propriété exprime le fait que si l'agent Author se trouve dans son état *InactiveA*, la propriété *Author -In-PreparationA*, exprimant le fait que l'Author sera dans son état *PreparationA*, est vraie dans le prochain état.

▪ Propriété 4

Formule LTL: $[] \text{ } Author -In-InactiveA$

- **Description:** En démarrant la vérification à partir de la configuration initiale *initialStateA* (Figure 6.48), cette propriété tente de vérifier s'il est toujours vrai que l'Author est dans son état *InactiveA*. Noter que cette propriété est *non désirable* au sein du système étudié.

▪ Propriété 5

Formule LTL : $DataBase-In-InactiveDB \mid \rightarrow O \text{ } DataBase -In-WaitPaperDB$

- **Description:** En démarrant la vérification à partir de la configuration initiale *initialStateDB* (Figure 6.48), cette propriété vérifie que si l'agent DataBase est dans son état *InactiveDB*, il sera alors dans son état *WaitPaperDB* dans le prochain état.

▪ Propriété 6

Formule LTL : $\text{PCMember-In-InactiveM} \rightarrow \text{O PCMember-In-WaitAbsM}$

- **Description:** En démarrant la vérification à partir de la configuration initiale `initialStateM` (Figure 6.48), cette propriété vérifie que si l'agent `PCMember` est dans son état `InactiveM`, il sera alors dans son état `WaitAbsM` dans le prochain état.

▪ Propriété 7

Formule LTL : $\text{Reviewer-In-InactiveR} \rightarrow \text{O Reviewer-In-DecisionR}$

- **Description:** En démarrant la vérification à partir de la configuration initiale `initialStateR` (Figure 6.48), cette propriété exprime le fait que si l'agent `Reviewer` est dans son état `InactiveR`, il sera dans son état `DecisionR` dans le prochain état.

▪ Propriété 8

Formule LTL : $\text{Printer-In-InactiveP} \rightarrow \text{O Printer-In-PrintProcP}$

- **Description:** En démarrant la vérification à partir de la configuration initiale `initialStateP` (Figure 6.48), cette propriété exprime le fait que si l'agent `Printer` est dans son état `InactiveP`, il sera dans son état `PrintProcP` dans le prochain état.

Afin d'énoncer ces 8 propriétés, plusieurs prédicats ont été utilisés. La Figure 6.47 présente une partie du module **RTMOISE-BASED-MAS-PREDICATES** dans lequel sont définis tous ces prédicats. Les lignes 1, 2 et 3 sont les prédicats définis pour les états `InactiveC`, `WaitPaperC` et `WaitAbsC` respectivement.

```
(mod RTMOISE-BASED-MAS-PREDICATES is
  including RTMOISE-BASED-MAS .
  including SATISFACTION .
  subsort Configuration < State .
  ops InitialStateC InitialStateC1 InitialStateA InitialStateDB InitialStateM InitialStateR InitialStateP : -> Configuration .
  var conf : Configuration .

  ops PCChair-In-InactiveC
    PCChair-In-WaitPaperC
    PCChair-In-WaitAbsC
    PCChair-In-PartitionC
    PCChair-In-WaitReviewC
    PCChair-In-DecisionC
    PCChair-In-WaitProcC
    PCChair-In-WaitFinalVerC
    PCChair-In-FailureC
    PCChair-In-SuccessC : -> Prop .

  ops Author-In-InactiveA
    Author-In-PreparationA
    Author-In-WaitRespa
    ....
  ops Printer-In-InactiveP
    Printer-In-PrintProcP
    Printer-In-SuccessP : -> Prop .

  eq < "PCChair" : PCChair | PlayRole : PCChair, CurrentState : InactiveC > conf |= PCChair-In-InactiveC = true .   ***[1]
  eq < "PCChair" : PCChair | PlayRole : PCChair, CurrentState : WaitPaperC > conf |= PCChair-In-WaitPaperC = true . ***[2]
  eq < "PCChair" : PCChair | PlayRole : PCChair, CurrentState : WaitAbsC > conf |= PCChair-In-WaitAbsC = true .   ***[3]
  ....
endm)
```

Figure 6.47. Une partie du module RTMOISE-BASED-MAS-PREDICATES.

3.6.2. Vérification des propriétés

Lorsque les huit propriétés ont été énoncées, il a été question de plusieurs états initiaux par lesquels débutent les vérifications de chacune de ces propriétés. En plus d'introduire ces configurations initiales, le module **RTMOISE-BASED-MAS-CHECK** décrit à la Figure 6.48, établit le lien final entre le SMATR étudié et le vérificateur de modèles de Maude. Étant donné le grand nombre de configurations initiales utilisées pour la vérification des 8 propriétés, seulement quelques-unes sont montrées dans cette Figure. Ainsi, les lignes 1 et 2 montrent les états initiaux utilisés pour la vérification du comportement individuel de l'agent temps réel PCChair.

Avec le module **RTMOISE-BASED-MAS-CHECK** tous les éléments nécessaires à la vérification formelle du système de Gestion de Conférence sont maintenant réunis. Il ne reste plus qu'à lancer la vérification à l'aide du *modèle checker* de Maude. Pour cela, on doit faire appel à la fonction *modelCheck* du module **MODEL-CHECKER** de Maude. Cette fonction a comme particularité de retourner un résultat positif (*True*) lorsqu'une propriété est vérifiée, ou un résultat de *contre-exemple* lorsqu'une propriété ne l'est pas. La fonction *modelCheck* de Maude nécessite l'utilisation de deux paramètres :

- *State*: Ce paramètre est en réalité l'état initial à partir duquel la vérification doit commencer;
- *Formulae*: Ce paramètre est en réalité la formule LTL qui doit être vérifiée au sein du système.

```
(mod RTMOISE-BASED-MAS-CHECK is
  pr RTMOISE-BASED-MAS-PREDICATES .
  inc MODEL-CHECKER .
  inc LTL-SIMPLIFIER .
  ops InitialStateC InitialStateC1 InitialStateDB InitialStateM InitialStateR InitialStateA InitialStateP : -> GlobalSystem .

eq InitialStateC = { Event("PCChair", InactiveC, IsInitializedConfC)          ***[1]
                    < "PCChair" : PCChair | PlayRole : PCChair >
                    }.

eq InitialStateC1 = { Event("PCChair", InactiveC, IsInitializedConfC)          ***[2]
                    Event("PCChair", WaitPaperC, ExpiredDateOfSoumissionC)
                    < "PCChair" : PCChair | PlayRole : PCChair >
                    }.
                    ...

endm)
```

Figure 6.48. Une partie du module RTMOISE-BASED-MAS-CHECK.

Étant donné qu'on a huit propriétés, il faut lancer huit appels à la fonction *modelCheck* pour obtenir tous les résultats de l'évaluation des propriétés. La Figure 6.49 montre quelques-uns des appels à cette fonction.

```
(red in RTMOISE-BASED-MAS-CHECK : modelCheck (InitialStateC, PCChair-In-InactiveC |-> 0 PCChair-In-WaitPaperC .)
(red in RTMOISE-BASED-MAS-CHECK : modelCheck (InitialStateC1, PCChair-In-InactiveC |-> <> PCChair-In-WaitAbsC .)
(red in RTMOISE-BASED-MAS-CHECK : modelCheck (InitialStateA, Author-In-InactiveA |-> 0 Author-In-PreparationA .)
(red in RTMOISE-BASED-MAS-CHECK : modelCheck (InitialStateA, [] Author-In-InactiveA .)
...
```

Figure 6.49. Quelques appels à la fonction modelCheck.

Une partie des résultats de l'évaluation de ces huit propriétés est illustrée dans la Figure 6.50.

```

rewrites: 13631 in 4341148309ms cpu (122ms real) (0 rewrites/second)
reduce in RTMOISE-BASED-MAS-CHECK :
  modelCheck(InitialStateC,PCChair-In-InactiveC |-> O PCChair-In-WaitPaperC)
result Bool :
  true

rewrites: 13628 in 4341153418ms cpu (31ms real) (0 rewrites/second)
reduce in RTMOISE-BASED-MAS-CHECK :
  modelCheck(InitialStateC1,PCChair-In-InactiveC |-> <> PCChair-In-WaitAbsC)
result Bool :
  true

rewrites: 13631 in 4341149543ms cpu (37ms real) (0 rewrites/second)
reduce in RTMOISE-BASED-MAS-CHECK :
  modelCheck(InitialStateA,Author-In-InactiveA |-> O Author-In-PreparationA)
result Bool :
  true

rewrites: 13564 in 4341146573ms cpu (32ms real) (0 rewrites/second)
reduce in RTMOISE-BASED-MAS-CHECK :
  modelCheck(InitialStateA,[ ]Author-In-InactiveA)
result ModelCheckResult :
  counterexample(nil,{InitialStateA,deadlock})

rewrites: 13631 in 4341151609ms cpu (37ms real) (0 rewrites/second)
reduce in RTMOISE-BASED-MAS-CHECK :
  modelCheck(InitialStateDB,DataBase-In-InactiveDB |-> O
    DataBase-In-WaitPaperDB)
result Bool :
  true

rewrites: 13631 in 4341146913ms cpu (35ms real) (0 rewrites/second)
reduce in RTMOISE-BASED-MAS-CHECK :
  modelCheck(InitialStateM,PCMember-In-InactiveM |-> O PCMember-In-WaitAbsM)
result Bool :
  true

rewrites: 13631 in 4341145912ms cpu (31ms real) (0 rewrites/second)
reduce in RTMOISE-BASED-MAS-CHECK :
  modelCheck(InitialStateR,Reviewer-In-InactiveR |-> O Reviewer-In-DdecisionR)
result Bool :
  true

rewrites: 13631 in 4341147632ms cpu (37ms real) (0 rewrites/second)
reduce in RTMOISE-BASED-MAS-CHECK :
  modelCheck(InitialStateP,Printer-In-InactiveP |-> O Printer-In-PrintProcP)
result Bool :
  true

```

Figure 6.50. Les résultats des appels à la fonction *modelCheck*.

4. Conclusion

Dans ce chapitre, nous avons présenté notre approche permettant la spécification et la validation de système multi-agent temps réel centré organisation. Plus précisément, un modèle organisationnel particulier de système multi-agents est spécifié qui est "Moise+". Ce dernier a été étendu pour intégrer la dimension temporelle composante importante des SMATR. Le nouveau méta-modèle obtenu est noté RT-Moise+ pour Real Time Moise+.

Au début nous avons utilisé le formalisme de modélisation graphique Agent UML vu que ce dernier offre plusieurs modèles conceptuels pour : décrire les besoins fonctionnels ; les

protocoles d'interaction; et le comportement interne et collectif des agents du SMATR ; etc. Dans notre contexte, on a utilisé les diagrammes de cas d'utilisation, de classe, de communication, d'état transition, de protocoles, et de temps, pour décrire ainsi les aspects fonctionnels, statique et dynamique des agents. Après, ces diagrammes sont translatés dans le langage formel Real Time Maude qui est basé sur la logique de réécriture pour obtenir un framework de spécification formel du modèle RT-Moise+ sous forme d'un ensemble de modules RT-Maude. Ensuite ce framework est vérifié en définissant un ensemble de propriétés dans le modèle checker fournit par RT-Maude pour s'assurer que notre formalisation du modèle RT-Moise+ est correcte. Sachant que ces étapes de développement sont basées sur le SMATR de l'exemple du Système de Gestion de Conférence qui est utilisé tout au long de ce chapitre.

Cette formalisation de RT-Moise+ montre que notre approche peut être appliquée à d'autres modèles organisationnels qui n'ont pas été analysés formellement au début.

Conclusion & Perspectives

1. Bilan

Actuellement, l'ingénierie orientée agent est un domaine de recherche très actif. Depuis de nombreuses années, les concepteurs de système multi-agents (SMA) ont des méthodologies de développement et des langages de modélisation qui ne reflètent pas les différentes contraintes temporelles qu'un système multi-agents temps réel (SMATR) peut avoir. En effet, il n'est pas facile de concevoir les concepts de SMATR en utilisant des approches d'ingénierie logicielle conventionnelles. Par conséquent, un problème de recherche critique pour la communauté des agents temps réel est celui de la définition d'un modèle organisationnel approprié pour l'analyse et la conception de leurs propriétés principales.

Chen Xueguang [Che02] affirme que les agents sont des objets actifs intelligents, avec certains objectifs donnés; ils peuvent former des SMA à travers les comportements interactifs. C'est un mode efficace pour décomposer, résumer et organiser des problèmes distribués ou complexes. Les organisations sociales humaines sont de bonnes comparaisons pour la modélisation de SMA. Selon les caractères inhérents aux agents, les hommes sont une sorte d'agents; les organisations sociales, composées d'hommes, peuvent naturellement correspondre au SMA. C'est-à-dire que les organisations peuvent être étendues en tant que communautés sociales, où des entités autonomes intelligentes (agents) interagissent les unes avec les autres pour atteindre certains objectifs. Ainsi, le processus de modélisation du SMA est le processus de représentation de l'organisation de l'agent.

Ainsi, une organisation peut être caractérisée par différents rôles au sein d'elle avec leurs relations statiques et dynamiques. La théorie de modélisation orientée agent et les différentes représentations dans AUML sont disponibles pour décrire les systèmes, suivant ces concepts. Par conséquent, des propositions tels que AGR [Fer03a], Message [Cai01], Moise+ [Hub02a], Ingenias [Gom05], et Gaia [Woo00b] ont été présentées afin d'organiser les SMA.

Les progrès dans ce domaine ont été lents mais généralement stables. La plupart des propositions considèrent une organisation comme une entité qui a un but concret ou un objectif global sans prendre en compte les restrictions temporelles des agents temps réel du système. Elles utilisent un ensemble de concepts organisationnels, comme rôles, normes, groupes, interactions, etc. pour contrôler et modifier les comportements des agents par rapport à l'objectif global de l'organisation. À notre avis, cette vision de l'organisation est incomplète. Nous proposons de l'étendre pour tenir compte des contraintes temporelles en étendant les diagrammes AUML comme illustrés dans [Lao13]. En particulier, nous croyons que le concept de restriction temporelle qu'un agent temps réel doit respecter au cours de son cycle de vie est très important pour être inclus dans l'organisation.

Un modèle organisationnel, tel que le terme l'indique, est une technique de modélisation utilisée pour représenter l'organisation sociale d'un SMA [Luc05]. Ce concept d'organisation sociale décrit les contraintes qui réalisent les actions et les interactions d'une collection d'agents. Au cours du processus de modélisation de SMA, comment établir un modèle organisationnel et comment l'étendre de manière appropriée en fonction des caractéristiques de SMATR est-ce notre problématique? Notre approche illustre la représentation des concepts sociaux conjointement avec les restrictions temporelles des agents comme la base de la modélisation du système multi-agents temps réel, proposant ainsi un nouveau modèle organisationnel pour spécifier de tels systèmes.

Premièrement, nous avons présenté dans [Lao13] une étude comparative de certains modèles organisationnels (AGR [Fer03a], Moise+ [Hub02a], Ingenias [Gom05], Gaïa [Woo00b] et RT-Message [Jul04]) et un méta-modèle organisationnel a été proposé, afin d'inclure principalement les concepts: Temporal Organization; Agent-O; Real Time Agent-O; et External Agent-O avec les concepts conventionnels de rôle; Objectif; Tâche; et norme sociale (en particulier les restrictions temporelles) qui ont été extraites des approches organisationnelles étudiées précédemment; pour être intégré dans un framework facilitant la modélisation de SMATR centré organisation. Ainsi, ce méta-modèle décrit les composants de l'organisation, leurs relations et connexions. Par conséquent, en considérant les propriétés de SMATR, ces méthodes reflètent séparément les deux vues (vue organisationnelle et vue temporelle). Il serait intéressant de les combiner en une seule technique lors de la spécification du modèle organisationnel, de sorte qu'une meilleure analyse du système puisse être faite.

En effet, la modélisation de SMATR est fréquemment liée à des concepts d'organisation dans ce sens que ces concepts offrent une base pour décrire les propriétés des agents, en appliquant un ensemble de techniques de génie logiciel. A notre avis, ces descriptions organisationnelles peuvent être modélisées par des diagrammes étendus AUMML [Lao13]. Le modèle organisationnel proposé diffère des autres approches de modélisation par l'utilisation des extensions de langage AUMML [Lao13] pour décrire, dans un premier temps, les concepts organisationnels de SMATR. En effet, le formalisme AUMML décrit plusieurs vues complémentaires du même système, et il faut souligner que son méta-modèles est basé sur le méta-modèle Agent-Groupe-Rôle (AGR).

L'approche proposée permet de représenter les aspects fonctionnels (diagramme de cas d'utilisation temporel AUMML centré organisation), statiques (diagramme de classe temporel AUMML centré organisation), et dynamiques (diagrammes de protocole AUMML avec les diagrammes d'états transition) de SMATR suivant la perspective organisationnelle. Les caractéristiques temporelles des systèmes temps réel peuvent être modélisées à l'aide d'AUMML à travers les stéréotypes, tagged values et les contraintes en général. Nous croyons que l'approche proposée est particulièrement appropriée à la modélisation de SMATR. De plus, aucune des méthodologies analysées ne prend en compte la formalisation du modèle organisationnel du futur système. La qualité de l'analyse du modèle est d'une grande importance pour le reste des phases du processus de développement. Leur spécification formelle et leur validation permettent d'éviter de nombreux problèmes pouvant affecter la qualité du développement.

Dans ce contexte, d'après [Ode00] les modèles AUMML souffrent d'un manque de sémantique formelle. Ils peuvent contenir des incohérences difficiles à détecter manuellement. Les méthodes formelles représentent une solution intéressante à ce problème. Les spécifications formelles permettent d'éliminer les ambiguïtés dans l'interprétation des modèles. Pour cela, la combinaison du formalisme AUMML avec un langage de spécification formel permettra de valider formellement le modèle organisationnel développé. Formaliser les aspects organisationnels de SMATR nous paraît un moyen important pour les activités d'analyse et de conception. De plus, la conception de SMATR nécessite l'implication de langages formels. Parmi ces langages on trouve Real Time Maude (RT-Maude) [Olv07] qui est probablement le langage le plus connu et le plus utilisé pour la programmation fonctionnelle et orientée objet. En outre, RT-Maude est très puissant en termes de spécification, de validation et de vérification des systèmes concurrents, ce qui en fait un bon candidat pour la

spécification et la validation de SMATR. Sachant cela, il n'y a actuellement aucun travail qui applique RT-Maude aux spécifications organisationnelles formelles de SMATR.

En effet, dans [Lao14] nous avons présenté une approche systématique supportant la translation des exigences organisationnelles de SMATR représentée par les diagrammes AUMML étendus proposés dans [Lao13] vers une spécification formelle en langage RT-Maude pour intégrer la validation formelle de la cohérence du modèle organisationnel dès la phase d'analyse et aussi pour produire des descriptions précises et offrir un meilleur support à leurs processus de vérification et de validation, en intégrant les possibilités offertes par RT-Maude pour vérifier certaines propriétés de spécification organisationnelle du système multi-agent temps réel. Ce framework organisationnel formel représente une approche importante pour la modélisation de SMATR, c'est une extension récente qui combine les diagrammes AUMML étendus [Lao13], avec la spécification RT-Maude [Lao14] pour supporter la spécification formelle de SMATR organisationnel. Dans ce framework, les points forts des deux approches sont unifiés, mais il manque encore plus de détails organisationnels à prendre en considération.

Cependant, les résultats présentés dans [Lao13, Lao14] étaient préliminaires et il fallait les approuver, ce qui nous a amené à enrichir le méta-modèle et par conséquent le framework de spécification formelle à travers les deux modèles organisationnels Agent-Groupe-Rôle (AGR) [Fer03a] et Model of Organization for multi-agent SystEms (Moise+) [Hub02a]. De plus, il y a actuellement peu de travaux appliquant la logique de réécriture aux modèles d'organisation de systèmes multi-agents et en particulier aux modèles AGR et Moise+. Par conséquent, Maude n'a pas été largement utilisé dans la communauté de système multi-agents organisationnel, et en particulier dans le domaine des modèles d'organisation d'agents. Ainsi, l'approche proposée qui est basée sur la logique de réécriture se distingue des autres travaux en traitant explicitement de la formalisation d'un modèle organisationnel avec le langage Maude. Néanmoins, certains travaux [Ast08, Ast09, Das08a, Van06] ont utilisé la logique de réécriture et Maude mais dans le contexte des langages de programmation d'agents ayant des fondements organisationnels.

Ainsi, une approche basée sur la logique de réécriture est décrite pour prototyper le modèle Agent-Groupe-Rôle de systèmes multi-agents centré organisation afin de pouvoir raisonner, d'une part, sur le modèle organisationnel et d'autre part sur sa sémantique générale. Cette approche formelle permet la validation et la vérification et présenté dans le cadre de l'étude de cas portant sur la Gestion de Chaîne Logistique. Ainsi, pour décrire le modèle Agent-Groupe-Rôle en termes de la logique de réécriture, il faut d'abord définir la syntaxe du modèle, puis décrire la sémantique correspondante en termes de langage formel Maude. Ce dernier est choisi comme langage de spécification formel puisqu'il est bien adapté pour le prototypage du modèle organisationnel AGR et qu'il est fourni avec un vérificateur de modèle (modèle checker) basé sur la Logique Temporel Linéaire. De plus, le concept d'importation de modules Maude aide à construire des applications modulaires à partir de modules courts, ce qui facilite le débogage, la maintenance ou l'extension. Dans [Lao17], les constructions syntaxiques de base qui sont nécessaires pour comprendre comment la logique de réécriture est incorporée dans le cadre d'Agent-Groupe-Rôle sont présentées.

En bref, cette formalisation telle que décrite dans [Lao17] montre que notre approche peut être appliquée à des modèles organisationnels qui n'ont pas été analysés formellement dès le début. Notre approche permet de capturer les aspects fonctionnels, statiques et dynamiques

de système multi-agent basé sur le modèle organisationnel Agent-Groupe-Rôle à l'aide de diagrammes AUML étendus, puis de traduire cette description graphique dans le langage Maude. Par ailleurs, la possibilité d'inclure une dimension temporelle pour traiter les problèmes de contraintes temporelles des agents a été étudiée dans le modèle organisationnel Moise+ afin de spécifier le modèle concret du système multi-agent temps réel en utilisant le langage Real Time Maude.

En effet, le deuxième modèle organisationnel Moise+ qui est plus développé qu'AGR a été étendu pour intégrer la dimension temporelle avec ses différentes dimensions déjà existantes afin de mieux tenir compte des spécificités des SMATR. Dans cette contribution on a suivi la même démarche que pour la formalisation d'AGR. Cette extension de Moise+ on l'a notée RT-Moise+ pour Real Time Moise+. Donc, dans un premier temps, une spécification formelle de RT-Moise+ est proposée. Elle adopte une approche basée sur la logique de réécriture pour décrire les contraintes temporelles d'organisation liées au SMATR, à travers la définition d'un ensemble de modules RT-Maude. L'un des avantages d'une telle spécification est qu'elle est systématique. Afin de décrire formellement le modèle RT-Moise+ en utilisant le langage RT-Maude, la représentation formelle du méta-modèle introduit dans [Lao14] sera employée. Ensuite, vient la phase de validation qui permet de simuler les spécifications avec des scénarios d'interaction prédéfinis, dans notre cas on a choisi l'étude de cas du Système de Gestion de Conférence. Ce qui nous a permis d'avoir d'un côté un modèle organisationnel facilitant la conception des SMATR et de l'autre côté un framework de spécification formelle offrant une base pour leur processus de formalisation.

Les avantages de l'approche présentée dans les deux chapitre 5 et 6 sont, premièrement, que le modèle organisationnel proposé soit décrit dans des termes formels et non ambigus et deuxièmement que la spécification générée décompose le modèle en différents modules formel RT-Maude qui peuvent être réutilisés dans des applications spécifiques. Ainsi, l'utilisation de notations formelles pour spécifier le modèle organisationnel permet de produire des descriptions organisationnelles précises. Cela offre également un meilleur support à leurs processus de vérification et de validation.

Pour résumer, nous avons proposé que dès la phase d'analyse initiale, un modèle organisationnel soit défini, détaillant les principaux acteurs, rôles, objectifs et dépendances. Dans ce modèle, plusieurs éléments organisationnels sont considérés (structure simple, rôles d'agent et objectifs) et pendant le processus de translation, nous avons développé un framework formel, où plusieurs modules RT-Maude sont générés, comme illustré dans [Lao14, Lao17], pour intégrer la validation formelle de la cohérence du modèle organisationnel, dès la phase d'analyse.

2. Perspectives

Comme nous l'avons souligné, les résultats rapportés dans cette thèse sont très prometteurs, mais il reste pas mal de pistes à explorer. Parmi elles: (i) nous prévoyons faire l'expansion de ce modèle organisationnel pour englober les différents concepts, se trouvant le long des dimensions identifiées au sein des diverses méthodes d'organisation d'agent analysées. Nous planifions aussi, d'une part: (ii) la spécification d'autres modèles organisationnels multi-agents en suivant la même approche décrite ici. Cela nous permettra d'avoir une bibliothèque de modèles multi-agents génériques et réutilisables. Ainsi que,

(iii) le développement de techniques d'analyse formelles supplémentaires, y compris l'analyse temporelle pour vérifier certaines propriétés collectives liées aux comportements des agents du système, en intégrant les possibilités offertes par Real-Time Maude. Et d'autre part, (iv) Nous essayons d'exploiter la plate forme de développement de système multi-agents "*Jade*" telle qu'elle ou avec des extensions si nécessaire pour déployer notre Framework organisationnel de système multi-agent temps réel afin de mieux manipuler les contraintes temporelles spécifiées dès le début dans les diagrammes AUML.

Bibliographie

- [Abr91]** J.R. Abrial, The B Method for Large Software Specification, Design and Coding. S. Prehn and W.J. Toetenel. Eds. VDM'91 : Formal Software Development Methods, 1991.
- [Ake02]** D. Akehurst, J. Derrick, et E. Borten, A Framework for UML Consistency. In Proceedings of the IEEE Workshop on Consistency Problems in UML-based Software Development, Department of Software Engineering and Computer Science of Blekinge Institute of Technology, 30–45, 2002.
- [Alk04]** A.B. Alkhodre, Développement formel de systèmes temps réel à l'aide de *SDL* et *IF* (Compilation pour système temps réel), Thèse de doctorat, L'Institut National des Sciences Appliquées de Lyon-france, 2004.
- [Alt02]** K. Altisen, G. Goessler, et J. Sifakis, Scheduler modeling based on the controller synthesis paradigm Journal of Real-Time Systems. Special Issue on Control Approaches to Real-Time Computing, vol. 23, 55-84. ISSN 0302-9743, 2002.
- [Alu91]** R. Alur, C. Courcoubetis, et D. Dill, Model Checking for Probabilistic Real Time Systems. In Proceeding of the Colloquium on Automata, Languages and Programming, Madrid, Spain, 115-126, 1991.
- [Amn01a]** T. Amnell, G. Behrmann, J. Bengtsson, P.R. Argenio, A. David, A. Fehnker, Th. Hune, B. Jeannet, K.G. Larsen, O.M. Möller, P. Pettersson, C. Weise, et W. Yi, Uppaal - Now, Next, and Future. In F. Cassez, C. Jard, B. Rozoy, et M. Ryan, editors, Modelling and Verification of Parallel Processes, number 2067 in Lecture Notes in Computer Science Tutorial,. Springer, Verlag, 100-125, 2001.
- [Amn01b]** T. Amnell, A. David, E. Fersman, M.O. Möller, P. Petterson, W. Yi, Tools for Real-Time UML : Formal Verification and Code Synthesis. SIVOE'S'2001, Budapest Hungary, 18-22 June 2001.
- [And01]** C. André, Paradigmes objets et synchrones dans les systèmes temps-réel. Journée Objets Temps Réel du Club SEE Systèmes Informatiques de Confiance, Paris, 18 janvier 2001.
- [And04]** J.E. Anderson, B. Tanner, et J. Baltes, Dynamic coalition formation in robotic soccer. In: Soh, L.-K., Anderson, J.E. (Eds.), AAAI-04 Workshop on Forming and Maintaining Coalitions and Teams in Adaptive Multiagent Systems. AAAI Press, San Jose, CA, 2004.
- [Arc05]** J.L. Arcos, M. Esteva, P. Noriega, J.A Rodríguez-Aguilar, et C. Sierra, Engineering open environments with electronic institutions. Eng. Appl. Artif. Intell, Vol. 18, 191–204, 2005.
- [Arg04]** E. Argente, A. Giret, S. Valero, V. Julian, et V. Botti, Survey of MAS methods and platforms focusing on organizational concepts. In: Vitrià, J., Radeva, P., Aguiló, I. (Eds.), Setè Congrés Català d'Intel·ligència Artificial, CCIA 2004. Recent Advances in Artificial Intelligence Research and Development. IOS Press, Barcelona, Catalonia, 309–316, 2004.
- [Arg06]** E. Argente, V. Julian, et V. Botti, Multi-Agent System Development Based on Organizations. Departamento Sistemas Informaticos y Computacion Polytechnic University of Valencia, Spain, Electron. Notes Theor. Comput. Sci, Vol. 150, 55–71. 2006.
- [Arg08]** E. Argente, V. Julian, et V. Botti, MAS Modeling based on Organizations. Departamento de Sistemas Informaticos y Computacion, Universidad Politecnica de Valencia, Spain, 2008.
- [Ast08]** L. Astefanoaei, M. Dastani, J.J.C. Meyer, et F.S. de Boer, A Verification Framework for Normative Multi-Agent Systems, in proceedings of 11th Pacific Rim International Conference on Multi-Agents (PRIMA), Hanoi, Vietnam, 54–65, 15-16 December 2008.

- [Ast09]** L. Astefanoaei, M. Dastani, J.J.C. Meyer, et F.S. de Boer, On the Semantics and Verification of Normative Multi-Agent Systems, in J. UCS, Vol. 15(13), 2629–2652, 2009.
- [Ast98]** E. Astesiano, UML as Heterogeneous Multiview Notation. Strategie for a Formal Foundation, In L. Andrade, A. Moreira, A. deshpande, et S. Kent, editors, Proceedings of the Conference on Object Oriented programming, Systems, Languages and Applications (OOPSLA'98)- Workshop on Formalizing UML. Why ? How ?, Canada, 1998.
- [Aud06]** L. Audibert, UML 2.0, Institut Universitaire de Technologie de Villetaneuse– Département Informatique, 2006.
- [Aza07]** S. Azaiez, Approche dirigée par les modèles pour le développement de systèmes multi-agents. Thèse de doctorat de l'université de SAVOIE, 2007.
- [Bab00]** J.Ph. Babau, Guide méthodologique pour PROSEUS, Rapport de recherche, CITI, INSA de Lyon, 35, 2000.
- [Bab01]** J.Ph. Babau, et A. Alkhodre. A development method for PROtotyping embedded SystEms by using UML and SDL (PROSEUS), workshop SIVOEES 2001 – ECOOP, Budapest, 6, 2001.
- [Bae05]** J. Baez, T. Stratulat, et J. Ferber, Un modèle institutionnel pour SMA organisationnel. In Proceedings of Journées Francophones sur les Systèmes Multi-Agents (JFSMA'05), Calais - France, Hermès-Lavoisier, Novembre 2005.
- [Bae96]** C. Baeijs et Y. Demazeau, Les organisations dans les systèmes multi-agents. Actes de la Journée sur les Systèmes Multi-Agents - PRC-GDR Intelligence Artificielle, Toulouse, Février 1996.
- [Bar02]** M. Le Bars, J.M. Attonaty, et S. Pinson, An agent-based simulation for water sharing between different users, In International Joint Conference on Autonomous Agents and Multi Agent Systems, Bringing people and agents together, 211–213, Bologne ACM, 2002.
- [Bar03]** M. Le Bars, Un simulateur multi-agent pour l'aide à la décision d'un collectif: Application à la gestion d'une ressource limitée agro-environnementale. Thèse de doctorat, Université Paris IX Dauphine, mai 2003.
- [Bar89]** H. Barringer, M. Fisher, D. Gabbay, G. Gough, et R. Owens, METATEM: A framework for programming in temporal logic, In REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness, LNCS, Vol. 430, 94-129. Springer-Verlag: Heidelberg, Germany, June 1989.
- [Bau00]** B. Bauer, J. P. Muller, et J. Odell, An extension of UML by protocols for multi-agent interaction, in International Conference on Multi-Agent Systems (ICMAS'00), Boston Massachussetts, USA, 207–214, July 2000.
- [Bau01a]** B. Bauer, J.P. Muller, et J. Odell. Agent UML: A Formalism for Specifying Multiagent Software Systems, The International Journal of Software Engineering and Knowledge Engineering, 2001.
- [Bau01b]** B. Bauer, UML Class Diagrams Revisited in the Context of Agent-Based Systems, Wooldridge M., Ciancarini P., Weiss G, Eds, Proceedings of Agent- Oriented Software Engineering (AOSE 01), nbr 2222 LNCS, Montreal, Canada, Springer-Verlag, 1–8, May 2001.
- [Bau05]** B. Bauer, et J. Odell. UML 2.0 and Agents: How to Build Agent-based Systems with the new UML Standard, 2005.

- [Bel99]** F. Bellifemmine, A. Poggi, et G. Rimassa, JADE-A FIPA compliant agent framework, In 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, 1999.
- [Ber04]** C. Bernon, M. Cossentino, M.P. Gleizes, P. Turci, et F. Zambonelli, A study of some Multi-Agent Meta-Models, In Proceedings of the fourth international workshop on software engineering for large-scale multi-agent systems, 2004.
- [Ber05]** D. Bertolini, A. Perini, A. Susi, et H. Mouratidis, The Tropos visual modeling language, A MOF 1.4 compliant meta-model, Contribution for the AOSE TFG meeting, 2005.
- [Boi04]** O. Boissier, S. Gitton, et P. Glize, Caractéristiques des systèmes et des applications. Observatoire Français des Techniques Avancées, Rapport de synthèse du Groupe Systèmes Multi-Agents, Février 2004.
- [Boi05]** O. Boissier, B. Gateau, D. Khadraoui, et E. Dubois, MoiseInst: An Organizational Model for Specifying Rights and Duties of Autonomou Agents. Published by Elsevier Science B. V, 2005.
- [Bor10]** A. Boronat et J. Meseguer, An algebraic semantics for mof. Formal Aspects of Computing, Vol. 22(3-4), 269–296, 2010.
- [Bot95]** V. Botti, F. Barber, A. Crespo, E. Onaindia, A. Garcia-Fornes, I. Ripoll, D. Gallardo, et L. Hernández, A temporal black-board for a multi-agent envirnment, Data and Knowledge Engineering, Vol. 15, 189-211, 1995.
- [Bot00]** P. Bottoni, M. Koch, F. Parisi-Presicce, et G. Taentzer, Consistency checking and visualization of OCL constraints, In The Unified Modeling Language Advancing the standard, third international conference, 294–308, 2000.
- [Bow03]** J. Bowen, Formal Specification and Documentation Using Z: A Case Study Approach. 2003.
- [Bra97]** F.M.T. Brazier, B. Dunin Keplicz, N. Jennings, et J. Treur, DESIRE: Modelling multi-agent systems in a compositional formal framework, Int. J. Coop. Inform. Syst, Vol. 6(1), 67-94, 1997.
- [Cai01]** G. Caire, W. Coulier, F.J. Garijo, J. Gomez, J., Pavon, F. Leal, P. Chainho, P.E. Kearney, J. Stark, R. Evans, et P. Massonet, Agent oriented analysis using message/UML. In AOSE, 119-135, 2001.
- [Cam94]** V. Camps, Une évaluation de la méthode de la relaxation restreinte pour l'auto-organisation dans les systèmes multiagents. Master's thesis, IRIT -Toulouse, 1994.
- [Can06]** D. Cansell, et D. Méry, Tutorial on the Event-based B Method, In FORTE 2006: Proceedings of the 26th International Conference on Formal Methods for Networked and Distributed Systems, Paris, France, 2006.
- [Car62]** A.P. Carl, Kommunikation mit Automaten, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM 2, 1962.
- [Car66]** A.P. Carl, Kommunikation mit Automaten. New York: Griffiss Air Force Base, Technical Report RADC-TR-65-377, Vol 1(1), 1966. Version anglaise.
- [Car01]** T. Carron, Des Systèmes Multi-Agents temporels pour des systèmes industriels dynamiques, thèse de doctorat, l'université Jean Monnet, 2001.

- [Cer04a]** L. Cernuzzi, T. Juan, L. Sterling, et F. Zambonelli, The Gaia methodology: basic concepts and extensions, In Bergenti, F., Gleizes, M.P., Zambonelli, F. (Eds.), *Methodologies and Software Engineering for Agent Systems*. Springer, US, 69–88, 2004.
- [Cer04b]** R. Cervenka, et I. Trencansky, *Agent Modeling Language: Language Specification*, Technical report, Version 0.9, Whitestein Technologies, 2004.
- [Cer05a]** L. Cernuzzi, M. Cossentino, et F. Zambonelli, Process models for agent-based development, *Journal of Engineering Applications of Artificial Intelligence*, Elsevier. Vol. 18(2), 205–222, March 2005.
- [Cer05b]** R. Cervenka, I. Trencansky, M. Calisti, et D. Greenwood, AML : Agent Modeling Language, Toward Industry-Grade Agent-Based Modeling. In In Odell, J., Giorgini, P., Muller, J., eds., *Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004*, Springer-Verlag, 2005.
- [Cha99]** B. Chaïb-Draa, *Agents et Systèmes Multiagents (IFT 64881A)*, Notes de cours, Département d’informatique, Faculté des sciences et de génie, Université Laval, Québec. Novembre 1999.
- [Cha01]** B. Chaib-draa, I. Jarras et B. Moulin, *Systèmes multiagents: Principes généraux et applications*, Editions J.P. Briot, Y. Demazeau, Agent et systèmes multiagents, Hermès, 2001.
- [Cha09]** G. Chalkiadakis, E. Elkind, M. Polukarov, et N.R. Jennings, The price of democracy in coalition formation, In, Decker, K., Sichman, J.S., Sierra, C., Castelfranchi, C. (Eds.), *8th International Conference on Autonomous Agents and Multiagent Systems*, ACM Press, Budapest, Hungary, 401–408, 2009.
- [Cha13]** W. Chama, R. Elmansouri et A. Chaoui, A modeling and verification approach based on graph transformation, In *Lecture Notes on Software Engineering*, Vol. 1(1), 39–43, 2013.
- [Che93]** V. Chevrier et C. Brassac, Non littéralité et diffusion de l’information dans les systèmes multi-agents, *Actes de la Conférence Européenne sur les Sciences Cognitives*, Saint-Malo, Avril 1995.
- [Che96]** M.A. Chechik, *Automatic Analysis of Consistency Between Requirements and Designs*. PhD thesis, University of Maryland at College Park, 1996.
- [Che99]** Y. Chen, Y. Peng, T. Finin, Y. Labrou, S. Cost, B. Chu, J. Yao, R. Sun, R. Willhelm, A negotiation-based multi-agent system for supply chain management. In *Workshop on Agent based Decision-Support for Managing the Internet-Enabled Supply-Chain*, Seattle, Third Conference on Autonomous Agents, May 1999.
- [Che01]** M. Chechik, et J.D. Gannon, Automatic Analysis of Consistency Between Requirements and Designs, In *Software Engineering*, Vol. 27(7), 651–672, 2001.
- [Che02]** G. Chen Xueguang, Exploiting Object-Oriented Methodologies to Model MAS on Organizations. Institute of Systems Engineering, Huazhong University of Science and Technology, Wuhan 430074, P.R.China. (ACM), Vol. 27(1), 58-62. 2002.
- [Cim00]** A. Cimatti, E. Clarke, F. Giunchiglia, et M. Roveri, Nusmv: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, Vol. 2, 2000, 2000.
- [Cla81]** E.M. Clarke, et E.A. Emerson, Synthesis of synchronization skeletons for branching time temporal logic. In *Workshop on Logic of Programs*, Springer-Verlag, Vol. 131, 52–71, 1981.

- [Cla96]** E.M. Clarke, J. M. Wing, R. Alur, R. Cleaveland, D. Dill, A. Emerson, S. Garland, S. German, J. Guttag, A. Hall, T. Henzinger, G. Holzmann, C. Jones, R. Kurshan, N. Leveson, K. McMillan, J. Moore, D. Peled, A. Pnueli, J. Rushby, N. Shankar, J. Sifakis, P. Sistla, B. Steffen, P. Wolper, J. Woodcock, et P. Zave, Formal Methods: State of the Art and Future Directions. In ACM Computing Surveys, Vol. 28(4), 626–643, 1996.
- [Cla02]** M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, et J. F. Quesada, Maude: specification and programming in rewriting logic, Theor. Comput. Sci, Vol. 285(2), 187–243, 2002.
- [Cla03]** E. M. Clarke, Model Checking Overview, Presentation Slides, 2003.
- [Cla05]** M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, et C. Talcott, Maude Manual (version 2.2). SRI International, Menlo Park, CA 94025, USA. December 2005
- [Cla06a]** M. Clavel et M. Egea, Equational specification of uml+ocl static class diagrams. <http://maude.sip.ucm.es/~clavel/pubs/clavel-egea06a.pdf>, 2006.
- [Cla06b]** M. Clavel et M. Egea, Itp/ocl: A rewriting-based validation tool for uml+ocl static class diagrams, In 11th International Conference on Algebraic Methodology and Software Technology AMAST'06, 2006.
- [Cla07a]** M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, et C. Talcott, All About Maude – A High-Performance Logical framework, LNCS, 4350, 2007.
- [Cla07b]** M. Clavel, F. Duran, J. Hendrix, S. Lucas, J. Meseguer, et P. Ölveczky, The Maude Formal Tool Environment Manuel, in International Conference on Algebra and Coalgebra in Computer Science, Springer, Berlin Heidelberg, Germany, 173–178, 2007.
- [Cla11]** M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, et C. Talcott, Maude manual (version 2.6), <http://maude.cs.uiuc.edu/maude2-manual/maude-manual.pdf>, 2011.
- [Col98]** J.C. Collis, D.T. Ndumu, H.S. Nwana, et L.C. Lee, The ZEUS agent building toolkit, BT Technology Journal, Vol. 16(3), 60-68, 1998.
- [Cos01]** M. Cossentino, et C. Potts, PASSI: a process for specifying and implementing Multi-Agent Systems Using UML, Vol. 8, 2007, 2002.
- [Cou05]** L.R. Coutinho, J.S. Sichman, et O. Boissier, Modeling Organization in MAS: A Comparison of Models, in first workshop on Software Engineering for Agent-Oriented Systems (SEAS'05), 1–10, 2005.
- [Cou09]** L.R. Coutinho, J. Sichman, et O. Boissier, Modelling Dimensions for Agent Organizations, in Dignum, V. (Ed.). Handbook of research on multi-agent systems: semantics and dynamics of organizational models, Vol. 2, 18–50, 2009.
- [Das02]** J.L.T. da Silva, et Y. Demazeau, Vowels coordination model, Proceedings of the first international joint conference on Autonomous agents and multi-agent systems AAMAS'02. ACM Press, Vol. part 3, 1129–1136, july 2002.
- [Das08a]** M. Dastani, Programming multi-agent systems, in International Workshop on Computational Logic in Multi-Agent Systems. Springer, Berlin Heidelberg, Germany, 13–16, 2008.
- [Das08b]** M. Dastani, D. Grossi, N. Tinnemeier, et J.J. Meyer, Normative Multi-agent Programs and Their Logics, in proceedings of First International Workshop, KRAMAS, Sydney Australia, 16–31, 17 September 2008.

- [Das09]** M. Dastani, N. Tinnemeier, et J.J. Meyer, A programming language for normative multi-agent systems, in *Handbook of Research on Multi-Agent Systems: semantics and dynamics of organizational models*, IGI Global, 397–417, 2009.
- [Dav11]** I. David, S. David, et M. Antonio, Organizational structures supported by agent-oriented methodologies, *The Journal of Systems and Software*, Vol. 84(2), 169–184, 2011.
- [Del02]** M. del-Mar Gallardo, P. Merino, et E. Pimentel, Debugging UML Designs Wit Model Checking, In *Journal of Object Technology*, Vol. 1(2), 101–117, 2002.
- [Del04]** S.A. DeLoach, The MaSE methodology, In: Bergenti, F., Gleizes, M.P., Zambonelli, F. (Eds.), *Methodologies and Software Engineering for Agent Systems: The Agent-oriented Software Engineering Handbook*. Kluwer Academic Publishers, 107–125, 2004.
- [Del10]** S.A. DeLoach, et J.C. García-Ojeda, O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems. *Int. J. Agent-Oriented Software Engineering*, Vol. 4(3), 244–280, 2010.
- [Dem95]** Y. Demazeau, From interactions to collective behaviour in agent based systems, Dans *Proceedings of the European Conference on Cognitive Science*, Saint-Malo, France, avril 1995.
- [Dem01]** Y. Demazeau, Voyelles. Mémoire d'Habilitation à Diriger des Recherches, Institut National Polytechnique de Grenoble INPG, Avril 2001.
- [Dig02]** V. Dignum, J. Meyer, H. Weigand, et F. Dignum, An Organization-oriented Model for Agent Societies. In: *Proceedings of International Workshop on Regulated Agent-Based Social Systems (RASTA'02) at AAMAS*, 2002.
- [Dig04]** V. Dignum, A Model for Organizational Interaction: based on Agents, founded in Logic, SIKS Dissertation Series 2004-1, SIKS, PhD Thesis, 2004.
- [Dig97]** F. Dignum, et R. Kuiper, Combining dynamic deontic logic and temporal logic for the specification of deadlines, In J. R. Sprague, editor, *Proceedings of thirtieth HICSS*, Wailea, Hawaii, 336, 1997.
- [Dod89]** R. Dodhiawala, N. S. Sridharan, P. Raulefs, et C. Pickering. Real-time AI systems: a definition and an architecture, *Proc, 11th IJCAI*, Détroit, 256–264, 1989.
- [Dup92]** J.P. Dupuy, *Introduction aux sciences sociales. (Logique des phénomènes collectifs)*. Ellipses, 1992.
- [Dup00]** S. Dupuy, Y. Ledru, M. Chabre-Peccoud, Vers une intégration utile de notations semi-formelles et formelles: une expérience en UML et Z, *Revue l'Objet*, Hermès, Paris, Vol. 6(1), 9–32, 2000.
- [Dur11]** F. Durán, M. Gogolla et M. Roldán, Tracing properties of uml and ocl models with maude, *Proceedings of International Workshop on Algebraic Methods in Model-Based Software Engineering (AMMSE 2011)*, *Electronic Proceedings in Theoretical Computer Science*, Vol. 56, 81–97, 2011.
- [Dur12]** F. Durán et M. Roldán, Validating ocl constraints on maude prototypes of uml models, 2012.
- [Duv01]** C. Duvallet, Des systèmes d'aide à la décision temps réel et distribuées : modélisation par agents. Thèse de doctorat. Université du Havre, octobre 2001.

- [Edw97]** S. Edwards, L. Lavagno, E.A. Lee, et A. Sangiovanni Vincentelli, Design of Embedded Systems: Formal Models, Validation, and Synthesis, Proc Of The IEEE, Vol. 85(3), 366—390, 1997.
- [Eke04]** S. Eker, J. Meseguer et A. Sridharanarayanan, The maude {LTL} model checker, Electronic Notes in Theoretical Computer Science, Vol. 71, 162 – 187, 2004.
- [Ekr02]** S. Eker, J. Meseguer, et A. Sridharanarayanan, The Maude LTL Model Checker. F. Gaducci and U. Montanari, editors, Proc of the 4th International Workshop on Rewriting Logic and its Applications, 230-234, 2002.
- [Ele08]** D. Elenius, G. Denker et M.O. Stehr, A semantic web reasoner for rules, equations and constraints. In Diego Calvanese et Georg Lausen, éditeurs, Web Reasoning and Rule Systems, Vol. 5341, 135–149. 2008.
- [Eme90]** E.A. Emerson, Temporal and modal logic, Int J. van Leeuwen, editor, Handbook of Theoretical Computer Science, Elsevier Science Publishers B.V: Amsterdam, The Netherlands, 996-1072, 1990.
- [Esf96]** B. Esfandiari, et D. Quinqueton, An interface agent for network supervision, In Proceedings of ECAI, Budapest, 1996.
- [Ete00]** K. Etessami, G. Holzmann, Optimizing Büchi automata, In Proc of the 11th International Conference on Concurrency Theory (CONCUR'2000), Lecture Notes in Computer Science Springer Verlag, of Lecture Notes in Computer Science, vol. 1877, 153–167, 2000.
- [Eva01]** R. Evans, P. Kearney, G. Caire, F. Garijo, J. Gomez Sanz, J. Pavon, F. Leal, P. Chainho, PT. Inovação et P. Massonet, MESSAGE: Methodology for engineering systems of software agents. EURESCOM, EDIN, 0223-0907. 2001.
- [Fer95]** J. Ferber, Les systèmes multi-agents: vers une intelligence collective. Informatique, Intelligence Artificielle, InterÉditions, Vol. 1, 1995.
- [Fer96]** J. Ferber, Reactive distributed Artificial Intelligence: principles and application, in Foundations of Distributed Artificial Intelligence, 287-314, 1996, Wiley & Sons.
- [Fer98]** J. Ferber, et O. Gutknecht, A meta-model for the analysis and design of organizations in MAS, In Proceedings Third International Conference on Multi Agent Systems, Paris, France, IEEE Press, 128-135, 03-07 July 1998.
- [Fer99]** J. Ferber, et G. Weiss, Multi-Agent System: An Introduction to Distributed Artificial Intelligence, Harlow: Addison Wesley Longman, ISBN: 0-201-36048-9, Vol.1, 1999.
- [Fer00]** J. Ferber et O. Gutknecht, Madkit: A generic multi-agent platform, In 4th International Conference on Autonomous Agents, 78-79, 2000.
- [Fer03a]** J. Ferber, O. Gutknecht, et F. Michel, From Agents to Organizations: an Organizational View of Multi-Agent Systems, in International Workshop on Agent-Oriented Software Engineering, Springer, Berlin Heidelberg, Germany, 214–230, 2003.
- [Fer03b]** J. Ferber, O. Gutknecht, et F. Michel, Agent/Group/Roles: simulating with organization, in Jean-Pierre Muller and Martina-M. Sidel editors, 4th workshop on agent based simulation, 34-39, 2003.

- [Fer05]** J. Ferber, F. Michel, et J. Baez, Agre: Integrating environments with organizations, In Environments for Multi-agent Systems, of Lecture Notes in Computer Science, Springer, Vol. 3374, 48–56, 2005.
- [Fin95]** T. Finin, Y. Labrou, et J. Mayfield, KQML as an agent communication language, Jeff Bradshaw (Ed.), Software Agents, MIT Press, Cambridge, 347-360, 1995.
- [Fip97]** FIPA, Foundation for Intelligent Physical Agents, FIPA 97 Specification, Part 1, Agent Management, Specification 10 Octobre, 1997.
- [Fip02]** FIPA ACL Message Structure Specification, Foundation for Intelligent and Physical Agents (FIPA), Geneva, Switzerland. 2002.
- [Fis94]** M. Fisher, A survey of Concurrent METATEM - the language and its applications, In D. M. Gabbay and H. J. Ohlbach, editors, Temporal Logic- Proceedings of the First International Conference LNAI, Springer-Verlag: Heidelberg, Germany, Vol. 827, 480-505, July 1994.
- [Foi98]** R. Foisel, Modèle de réorganisation de systèmes multi-agents: Une approche descriptive et opérationnelle, Thèse de Doctorat de l'université Henri Poincaré-Nancy, Vol 1, 1998.
- [Fox81]** M.S. Fox, An organizational view of distributed systems, IEEE Transactions on Systems, Man and Cybernetics, SMC, Vol. 11 (1), 70-80, January 1981.
- [Fue10]** R. Fuentes-Fernández, I. García-Magariño, A.M. Gómez-Rodríguez, et J.C. González-Moreno, A technique for defining agent-oriented engineering processes with tool support, Eng. Appl. Artif. Intell, Vol. 23(3), 432–444, 2010.
- [Gag07]** P. Gagnon, Vérification formelle de diagrammes UML: une approche basée sur la logique de réécriture, Thèse de doctorat, Université du québec. Août 2007
- [Gar03]** T. Garneau, La Programmation Orientée-Agent DMAS Builder: Un environnement de Développement De Systèmes Multi-Agents Totalemt Distribué, Département de mathématiques et d'informatique, Thèse de doctorat, Université du Québec à Trois-Rivières , Août 2003.
- [Gar09]** E. Garcia, A. Giret, et V. Botti, Masev (Multiagent System Software Engineering, Evaluation Framework), Departamento de Sistemas Informaticos y Computacion, Universidad Politecnica de Valencia, 277-290, 2009.
- [Gas01]** L. Gasser, Organizations in multi-agent systems, in Pre-Proc of the 10th European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW'2001), Annecy, France, 2001.
- [Gas92]** L. Gasser, An overview of distributed artificial intelligence, in Distributed Artificial Intelligence: Theory and Praxis, Kluwer academic publishers, Vol. 5, 1992.
- [Gat07a]** B. Gâteau, Modélisation et Supervision d'Institutions Multi-Agents. Thèse de doctorat de l'Ecole Nationale Supérieure des Mines de Saint Etienne, 2007.
- [Gat07b]** M. Gatti, V. Staa, C. Lucena, AUML-BP: A Basic Agent Oriented Software Development Process Model Using AUML. Pontifícia Universidade Católica Do Rio De Janeiro. Vol. 21(07), 25, August 2007.
- [Gau07]** N.A. Gaud, Systèmes multi-agents holoniques: de l'analyse à l'implantation: méta-modèle, méthodologie, et simulation multi-niveaux, Thèse de doctorat, Besançon, 2007.

- [Ghb00]** M. Greaves, H. Holmback, et J. Bradshaw, What is a conversation policy? Issues in Agent Communication, F. Dignum, M. Greaves editors, LNAI 1916, Springer Verlag, 118–131, 2000.
- [Ger97]** J. H. Gerard, The Model Checker SPIN. Software Engineering, Vol. 23(5), 279-295, 1997.
- [Gim06]** H. Gimpel, N.R. Jennings, G.E. Kersten, A. Ockenfels, et C. Weinhardt, Market Engineering: A Research Agenda, International Seminar Negotiation, Auctions and Market Engineering, Springer Verlag, Dagstuhl Castle, Germany, 1–15, 2006.
- [Giu02]** F. Giunchiglia, J. Mylopoulos, et A. Perini, The Tropos software development methodology: Processes, models and diagrams, In: AAMAS'02 Workshop on Agent Oriented Software Engineering (AOSE-2002), 63–74, 2002.
- [Gli05]** B. Glisse, Schémas d'interactions dans les systèmes multiagents. 7^{ème} Rencontres de Théo Quant, Besançon, Janvier 2005.
- [Gom05]** J. Gomez, R. Fuentes, et J. Pavon, The INGENIAS Methodology and Tools, Agent oriented Methodologies, Idea Publishing Group, 236–276, 2005.
- [Gue96]** Z. Guessoum, Un environnement opérationnel de conception et de réalisation de systèmes multi-agents. These de doctorat, Université Pierre et Marie Curie. mai 1996.
- [Gue00]** A. LE Guennec, Méthodes formelles avec UML: Modélisation, validation et génération de tests, Actes du 8^{ème} Colloque Francophone sur l'Ingénierie des Protocoles CFIP'2000, Toulouse, Paris, Editions Hermès, Paris, 151-166, 17-20 octobre 2000.
- [Gue03]** Z. Guessoum, Modèles et architectures d'agents et de systèmes multi-agents adaptatifs, Rapport de soutenance de HDR, Université Pierre et Marie Curie, 2003.
- [Gue11a]** M. Gueffaz, S. Rampacek et C. Nicolle: RDF2N μ SMV: mapping semantic graphs to N μ SMV model checker. In AFIN 2011, The Third International Conference on Advances in Future Internet, 49–53, 2011.
- [Gue11b]** M. Gueffaz, S. Rampacek et C.e Nicolle: Rdf2spin : Mapping semantic graphs to spin model checker. In Digital Information and Communication Technology and Its Applications, Vol.167, 591–598, 2011.
- [Gue12]** M. Gueffaz, ScaleSem: Model Checking et Web Sémantique, Thèse de doctorat, Université de Bourgogne, 2012.
- [Gul13]** S. Gulan, S. Johr, R. Kretschmer, S. Rieger, et M. Ditzte, Graphical modelling meets formal methods, In Industrial Informatics (INDIN), 11th IEEE International Conference, 716–721, 2013.
- [Gut99]** O. Gutknecht, et J. Ferber, Un méta-modèle organisationnel pour l'analyse, la conception et l'exécution de systèmes multi-agents. Laboratoire d'Informatique, Robotique et Micro-électronique de Montpellier. C.N.R.S. - Université Montpellier II, Vol. 98, 128-135, 1999.
- [Gut01]** O. Gutknecht, Proposition d'un modèle organisationnel générique de systèmes multi-agents et examen de ses conséquences formelles, implémentatoires et méthodologiques. Thèse de doctorat. Université Montpellier II-Sciences et Techniques du Languedoc. 2001.
- [Hab68]** J. Habermas, The Idea of the Theory of Knowledge as Social Theory, Knowledge & Human Interest, 1968.
- [Hal87]** D. Hatley, I. Pirbhai, Strategies for Real-time System Specification. Dorset House, New York, ISBN 0-932633-11-0, 408, 1987,

- [Han02]** M. Hannoun, MOISE: un modèle organisationnel pour les systèmes multi-agents, Thèse de doctorat, Ecole Nationale Supérieure Mines de Saint-Etienne & Université Jean Monnet Saint-Etienne, 2002.
- [Hei00]** C. Heinze, M. Papasimeon, et S. Goss, Specifying Agent behaviour with use case, 2000.
- [Her04]** I.R. Hernandez, Modélisation, spécification formelle et vérification de protocoles d'interaction: une approche basée sur les rôles, Thèse doctorat, Institut National Polytechnique de Grenoble - INPG, 2004.
- [Hik03]** A.M. Hikolo, Analyse, conception, spécification et développement d'un système multi-agents pour le soutien des activités en formation à distance, Thèse de doctorat, Université de Franche-Comté, France, 2003.
- [Hil04]** V. Hilaire, O. Simonin, A. Koukam, et J. Ferber, A Formal Approach to Design and Reuse Agent and Multiagent Models, in proceedings of Agent-Oriented Software Engineering V, 5th International Workshop, AOSE, New York, USA, 142–157, 2004.
- [Hoj93]** R. Hojati, T.R. Shiple, R.K. Brayton, R.P. Kurshan, A unified approach to language containment and fair CTL model checking, Proc of the 30th international on Design automation conference, ISBN 0-89791-577-1, 475 – 481, July 1993.
- [Hol91]** G.J. Holzmann, The Spin model checker, IEEE Transactions on Software Engineering, , vol. 23 (5), 279–295, May 1997.
- [Hol03]** G. Holzmann, Spin Model Checker, the Primer and Reference Manual, Addison-Wesley Professional, first édition, 2003.
- [Hor05]** B. Horling, et V. Lesser, A survey of multi-agent organizational paradigms, Knowl. Eng. Rev, Vol. 19, 281–316, 2005.
- [How01]** N. Howden, R. Ronnquist, A. Hodgson, et A. Lucas, JACK intelligent agents Summary of an Agent Infrastructure, 5th International Conference on Autonomous Agents 2001.
- [Hua09]** N. Huang, X. Wang et C. Rocha: Formal semantics of owl-s with rewrite logic. JSEA, Vol. 2(1), 25–33, 2009.
- [Hub02a]** J.F. Hubner, J.S. Sichman, et O. Boissier, A model for the structural, functional, and deontic specification of organizations in multi-agent systems. In G. Bittencourt et G. L. Ramalho, editors, Proceedings of the 16th Brazilian Symposium on Artificial Intelligence (SBIA'02), LNAI 2507, Porto de Galinhas, PE, Brazil, Springer, 118-128, 2002.
- [Hub02b]** J.F. Hübner, J. S. Sichman, et O. Boissier, Spécification structurelle, fonctionnelle et déontique d'organisations dans les SMA. In JFSMA, 205-216, 2002.
- [Hub07]** J.F. Hübner, J.S. Sichman, et O. Boissier, Developing organised multi-agent systems using the Moise+ model: programming issues at the system and agent level, Int. J. Agent-Orient. Softw. Eng, Vol. 1(3-4), 370–395, 2007.
- [Hug02a]** M.P. Huget, An Application of Agent UML to Supply Chain Management, in proceedings of Agent Oriented Information System (AOIS-02), Giorgini, P. et al (Eds.), (short presentation of the technical report ULCS-02-015), Bologne, Italie, 1-14, 2002.
- [Hug02b]** M.P. Huget, Extending agent UML protocol diagrams, In F. Giunchiglia, J. Odell, et G. Weiss, editors, AAMAS Workshop on Agent-Oriented Software Engineering (AOSE), Bologna, Italy, July 2002.

- [Hug02c]** M.P. Huget, Agent UML class diagrams revisited, Technical Report, Department of Computer Science, University of Liverpool, 1-13, 2002.
- [Hug02d]** M. P. Huget, Design Agent Interaction As A Service To Agent, In M.P. Huget, F. Dignum, et J.L. Koning, editors, AAMAS Workshop on Agent Communication Languages and Conversation Policies (ACL2002), Bologna, Italy, July 2002.
- [Hui00]** K. Huizing, et R. Kuiper, Verification of Object Oriented Programs Using Class Invariants, In FASE '00, Proceedings of the Third International Conference on Fundamental Approaches to Software Engineering, , London UK, Springer-Verlag, 208–221, 2000.
- [Igl99]** C.A. Iglesias, M. Garijo, et J.C. Gonzalez, A survey of agentoriented methodologies, In Proceeding of the 15th International Workshop on Agent Theories Architectures and Languages, Budapest, Hungary, sept 1999.
- [Jah86]** F. Jahanian, et A. K. Mok, Safety analysis of timing properties in real time systems, IEEE Transactions on software Engineering, ISSN: 0098-5589. SE-12, Vol. 12(9), 890-904, September 1986.
- [Jar02]** I. Jarras et B. Chaib-Draa, Aperçu sur les systèmes multiagents. Série Scientifique, Montréal, 2002.
- [Jen98a]** N. R. Jennings et M. Wooldridge, Applications of Intelligent Agents. Agent Technology: Foundations, Applications, and Markets (eds. N. R. Jennings and M. Wooldridge) Springer-Verlag: Berlin, Germany, 3-28, 1998.
- [Jen98b]** N. R. Jennings, K. Sycara, et M. Wooldridge, A Roadmap of Agent Research and Development. Journal of Autonomous Agents and Multi-Agent Systems, Vol. 1(1), 7-38, 1998.
- [Joh96]** W. Johnston, A Type Checker for Object-Z. Technical Report 96–24, SVRC, University of Queensland, September 1996.
- [Jon01]** C. M. Jonker, J. Treur, et W. de Vries, Temporal requirements for anticipatory reasoning about intentional dynamics in social contexts, In Y. Demazeau and F. J. Garijo, editors, Proceedings of the 10th European Workshop on Multi-Agent Systems, MAAMAW'01, LEIBNIZ, (electronic publishing), 2001.
- [Jul01]** L. Julio, M. Pasaje, M. G. Harbour, et J. M. Drake, MAST Real-Time View: A Graphic UML Tool for Modeling Object-Oriented Real-Time Systems, 22nd IEEE Real-Time Systems Symposium (RTSS'01) London, England, 245-256, 03 - 06 December 2001.
- [Jul04]** V. Julian, et V. Botti, Developing Real-Time Multi-Agent Systems, Integrated Computer-Aided Engineering, Vol.11, 135–149, 2004.
- [Kel96]** S. Kelly, K. Lyytinen, et M. Rossi, MetaEdit+: a fully configurable multi-user and multi-tool CASE and CAME environment, In Constantopoulos, P., Mylopoulos, J., Vassiliou, Y. (Eds.), Advanced Information Systems Engineering, Proc of 8th International Conference, CAiSE 1996, Springer Berlin-Heidelberg, Heraklion, Crete, Greece, 1–21, 1996.
- [Kin92]** D. Kinny, M. Ljungberg, A. S. Rao, E. Sonenberg, G. Tidhar, et E. Werner, Planned team activity, In C. Castelfranchi and E. Werner, editors, Artificial Social Systems Selected Papers from the Fourth European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW-92 LNAI, Springer-Verlag : Heidelberg, Germany, Vol. 830, 226-256, 1992.

- [Kle09]** F. Klein, Contrôle d'un système multi-agents réactif par modélisation et apprentissage de sa dynamique globale, Rapport de thèse, Université de Nancy 2, Décembre 2009.
- [Kle93]** M. Klein, A practitioner's Handbook for Real-Time Analysis, Boston, Kluwer Academic Publishers, ISBN 0792393619, 712, 1993.
- [Kon84]** K. Konolige, A deductive model of belief, In Proceedings of the Eighth International Joint Conference on Artificial Intelligence, 377-381, August 1984.
- [Lal01]** R. Laleau, et A. Mammar, An automatic generation of b specifications from well-defined UML notations for database applications, In Int. Symp. on Programming Systems, Alger, Algérie, 2001.
- [Lao10]** M.A. Laouadi, F. Mokhati, et H. Seridi-Bouchelaghem, A Novel formal specification approach for Real Time Multi-Agent System Functional Requirements. In Proc. Mates10, Leipzig Germany, Springer, 2010.
- [Lao13]** M.A. Laouadi, F. Mokhati, et H. Seridi-Bouchelaghem, Towards an Organizational Model for Real Time Multi-Agent System Specification, in proceedings of Science and Information Conference (SAI), London UK, 577-584, 7-9 October 2013.
- [Lao14]** M.A. Laouadi, F. Mokhati, et H. Seridi-Bouchelaghem, A Novel Organizational Model for Real Time MAS: Towards a Formal Specification, in Chen, L. et al (Eds.), Intelligent Systems for Science and Information, Studies in Computational Intelligence, Vol. 542, 171-180, 2014.
- [Lao17]** M.A. Laouadi, F. Mokhati, et H. Seridi-Bouchelaghem, A formal framework for organization- centered multi-agent system specification: A rewriting logic based approach, In Journal of Multi-agent and Grid Systems, Vol. 13(4), 395-419, 2017.
- [Lar00]** F. Laroussinie, et F. Cassez, Model-checking for hybrid systems by quotienting and constraints solving, In E. Allen Emerson and A. Prasad Sistla, editors, 12th International Conference on Computer-Aided Verification, LNCS 1855, Springer-Verlag, ISBN 3-540-67770-4, Vol. 1855, 373 - 388, July 2000.
- [Led01]** H. Ledang, et J. Souquière, Formalizing UML behavioral diagrams with b, In Tenth OOPSLA Workshop on Behavioral Semantics, Back to Basics, 2001.
- [Lem98]** C. Lemaitre, et C.B. Excelente, Multi-agent organization approach, in proceedings 2nd Iberoamerican Workshop on Distributed Artificial Intelligence and Multi-Agent Systems, Toledo, USA, 7-16, 1998.
- [Len94]** C. Lenay, Organisation émergente dans les populations: biologie, éthologie, systèmes artificiels. Intellectica, Vol. 19(2), 9-17, 1994.
- [Lev01]** V. Levin, F. Xie, et J.C. Browne, Integrating Model Checking Into Object-Oriented Software Development Processes, Technical Report TR-01-03, UTCS, 2001.
- [Lin99]** J. Lind, MASSIVE: Software Engineering for Multi-agent Systems. PhD Thesis, DFKI, Germany, 1999.
- [Lin07]** C. Lin, et K.M. Kavi, A Methodology to Evaluate Agent Oriented Software Engineering Techniques. Dept of Computer Science and Engineering, University of North Texas, Abercrombie Computational Sciences and Engineering Oak Ridge National Laboratory, IEEE, 2007.
- [Lju92]** M. Ljunberg, et A. Lucas, The OASIS air traffic management system, In Proceedings of the Second Pacific Rim International Conference on AI (PRICAI-92), Seoul, Korea, 1992.

- [Loo00] P. De Loor, et P. Chevaillier, Generation of agent interactions from temporal logic specifications, In M. Deville and R. Owens, editors, 16th MACS World Congress, August 2000.
- [Luc05] R. Luciano, L.R. Coutinho, J.S. Sichman, et O. Boissier, Modeling Organization in MAS: A Comparison of Models, Laboratorio de Tecnicas Inteligentes, Universidade de Sao Paulo (USP), 02. 2005
- [Maa04] T.M. Maarouk, Spécification Formelle Des Systèmes Mobiles Temps Réel, Equipe vision et infographie, laboratoire LIRE, université mentouri Constantine, 2004.
- [Mal87] T. Malone, Modeling coordination in organizations and markets, In Management science, Vol. 33(10), 1317-1332, 1987.
- [Mal05] H. Malgouyres, S.W. Jean-Pierre, et G. Motet, Identification de Règles de Cohérence d'UML 2.0. Journée SEE "Systèmes Informatiques de Confiance" sur le thème "Vérification de la cohérence de modèles UML", Paris, France, Mars 2005.
- [Mam98] Z. Mammeri, Expression et dérivation des contraintes temporelles dans les applications temps réel, APII-JESA, Vol. 32(5-6), 609-644, 1998.
- [Mam00] Z. Mammeri, SDL Modélisation de protocoles et systèmes réactifs, germes, paris, ISBN 2746201666, 623, juillet 2000.
- [Man07] S. Mansour, et J. Ferber, Agent Groupe Rôle et Service: Un modèle organisationnel pour les systèmes multi-agents ouverts, JFSMA, 2007.
- [Mar97] P. Marcenac, et S. Calderoni, Self-organisation in agent-based simulation, In Proceedings of MAAMW' 97, Ronnedy, Sweden, 1997.
- [Mar02] N. Martí-Oliet, et J. Meseguer, Rewriting logic as a logical and semantic framework. In J. Meseguer, éditeur, Handbook of Philosophical Logic, de Handbook of Philosophical Logic, Elsevier Science Publishers, Vol. 9, 1-87, 2002.
- [Maz13] H. MAZYAD, Une Approche Multi-agents à Architecture P2P pour l'Apprentissage Collaboratif, Thèse de doctorat, Université du Littoral Côte d'Opale, 2013
- [McC69] J. McCarthy, et P.J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, In B. Meltzer and D. Michie, editors, Machine Intelligence 4, Edinburgh University Press, 1969.
- [Mcm91] K. McMillan, et J. Schwalbe, Formal verification of the gigamax cache consistency protocol, In Proc of the International Symposium on Shared Memory Multiprocessing, Tokyo, Japan, Inf. Process. Soc, 242-251, 1991.
- [Mcm92] K.L. McMillan, The SMV system, symbolic model checking, Technical Report CMU-CS, Carnegie Mellon University, 92-131, 1992.
- [Mel05] S.F. Melián, I. Marsá, M. Ukrania, D.R. Miguel, A.L. Carmona, Extending the BDI ASDP methodology for real time, 2005.
- [Mer00] S. Merz, Model checking, Proc of the summer school on Modeling and Verification of Parallel processes - MOVEP'2k, 51-70, 2000.
- [Mes90] J. Meseguer, A logical theory of concurrent objects, OOPSLA/ECOOP '90 Proceedings of the European conference on object-oriented programming on Object-oriented programming systems, languages and applications, Vol.25(10), 101-115, 1990.

- [Mes02]** J. Meseguer, Software Specification and Verification in Rewriting Logic. Germany, August 2002.
- [Mes10]** J. Meseguer, Twenty years of rewriting logic, In Rewriting Logic and Its Applications, Springer Berlin Heidelberg, Vol. 6381, 15–17, 2010.
- [Mic04]** F. Michel, Formalisme, outils et éléments méthodologiques pour la modélisation et la simulation multi-agents, Thèse de Doctorat, Université des Sciences et Techniques du Languedoc, 2004.
- [Mil01]** E. Milgrom, P. Chainho, Y. Deville, R. Evans, P. Kearney, et P. Massonet, MESSAGE-Methodology for Engineering Systems of Software Agents (Final Guidelines for the Identification of Relevant Problem Areas where Agent Technology is Appropriate. Technical report, EURESCOM, 2001.
- [Min79]** H. Mintzberg, The Structuring of Organizations, a synthesis of the research, Prentice-Hall, 1979.
- [Mok07]** F. Mokhati, P. Gagnon, et M. Badri, Verifying UML diagrams with model checking: a rewriting logic based approach, in: A. Mathur, W.E. Wong (Eds.), Proceedings of the 7th International Conference on Quality Software, QSIC 2007, Portland, Oregon, USA, IEEE Computer Society, 356–362, 11–12 October, 2007.
- [Mok09]** F. Mokhati, et M. Badri, Generating maude specifications from UML Use Case Diagrams. Journal of Object Technology, Vol.8(2), 119–136, 2009.
- [Mok10]** F. Mokhati, B. Sahraoui, S. Bouzaher, et M.T. Kimour, A tool for specifying and validating agents interaction protocols, from Agent UML to Maude, Journal of Object Technology, Vol. 9(3), 59–77. 2010.
- [Mon08]** D. Monticolo, Une approche organisationnelle pour la conception d’un système de gestion des connaissances fondé sur le paradigme agent, Thèse de doctorat, Université de Technologie de Belfort Montbéliard et Université de Franche Comté, 2008.
- [Mor00]** A. Moreira, J. M. Bruel, J. Lilius, et B. Robert, Defining Precise Semantics for UML, In ECOOP’2000 Workshop Reader, number 1964. Springer-Verlag, November 2000.
- [Mou96]** B. Moulin, et B. Chaib-Draa, An overview of distributed artificial intelligence, G.M.P.O’Hare and N.R. Jennings, editors, Foundations of Distributed AI, John Wiley & Sons, Chichester, England, 3-54, 1996.
- [Moy06]** T.H. Moyaux, B. Chaib-draa, et S. D’Amours, Supply Chain Management and Multiagent Systems: An Overview, in Chaib-draa, B. and Muller, J. (Eds.), Multiagent Based Supply Chain Management, Studies in Computational Intelligence, Vol.28, 1–27, 2006.
- [Mul00]** J.P. Muller, Modélisation organisationnelle en systèmes multiagents, In 7^{ème} école d’été de l’ARCO, Bonas, 2000.
- [Mul02]** J. P. Müller, Des systèmes autonomes aux systèmes multi-agents, Interaction, émergence et systèmes complexes, 2002.
- [Mur14a]** F.T. Muramatsu, T.M. Vitorello, et A.A.F. Brandao, Towards organizational interoperability through artifacts, in Agent Environments for Multi-Agent Systems 10 Years Later, Paris, E4MAS@AAMAS-2014, IFAAMAS, Liverpool, UK, 1–14, 2014.

- [Mur14b]** F.T. Muramatsu, T.M. Vitorello, et A.A.F. Brandao, Towards Organizational Interoperability Through the Environment, in Revised Selected and Invited Papers of the 4th International Workshop on Agent Environments for Multi-Agent Systems IV, Springer-Verlag, New York, USA, 209–231, 2014.
- [Mus03]** J. Muskens, L. Somers, C. Lange, M. Chaudron, et E. Dortmans, An Empirical Investigation in Quantifying Inconsistency and Incompleteness in UML Designs, In Proceedings of the IEEE Workshop on Consistency Problems in UML-Based Software Development II, 26–34, 2003.
- [Naw10]** E. Nawarecki, et J. Koźlak, Building multi-agent models applied to supply chain Management, Control and Cybernetics, Vol. 39(1), 149–176, 2010.
- [Ngu98]** H. P. Nguyen, Derivation des spécifications formelles B à partir de spécifications semi-formelles. Thèse de doctorat, Centre d'Études et De Recherche en Informatique du CNAM (CEDRIC), 1998.
- [Nwa96]** H. Nwana, L. Lee, et N. Jennings, Coordination in software agent systems, BT Laboratories international report, 1994.
- [Obe03]** I. Ober, S. Graf, et I. Ober, Validating timed UML models by simulation and verification, In Proc of the SVERTS'03 Workshop (satellite of UML'03 Conference), San Francisco, California, 12, 2003.
- [Obo11]** E.A. Obonyo, et C.J. Anumba, Organization-oriented multi-agent systems for construction supply chains, ITcon, Vol. 16, 727–744, 2011.
- [Ode00]** J. Odell, P. Van Dyke, et B. Bauer, Extending UML for Agents, Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, Gerd Wagner, Yves Lesperance, and Eric Yu eds., Austin, TX, 3-17, 2000.
- [Olv06]** P.C. Olveczky, Real-Time Maude 2.2 Manual, Department of Informatics, University of Oslo, 2006.
- [Olv07]** P.C. Olveczky, Real-Time Maude 2.3 Manual, Department of Informatics, University of Oslo, 2007.
- [Pad02]** L. Padgham, et W. Winikoff, Prometheus: A methodology for developing intelligent agents, In: AAMAS'02 Workshop on Agent Oriented Software Engineering, 135–145, 2002.
- [Pal99]** M. Paludetto, J. Delatour, UML et les réseaux de Petri, vers une sémantique des modèles dynamiques et une méthodologie de développement des systèmes temps réel, Revue L'objet, Editions Hermes, vol. 5(3-4), 443–467, 1999.
- [Pap00]** M. Papasimeon, et C. Heinze, Specifying Requirement in Multi-agent system with use cases, march 2000.
- [Par02]** H.V.D. Parunak, et J. Odell, Representing Social Structure in UML, in Agent-Oriented Software Engineering II, Springer, Montreal, Canada, 1–16, 2002.
- [Pav05]** J. Pavón, J.J. Gómez-Sanz, et R. Fuentes, The INGENIAS methodology and tools, In: Henderson-Sellers, B., Giorgini, P. (Eds.), Agent-oriented Methodologies. Idea Group, 236–276, 2005.

- [**Pes97**] S. Pesty, C. Brassac, et P. Ferrent, Ancrer les agents cognitifs dans l'environnement, In : Actes des 5^{ème} Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents, La Colle sur Loop, Avril 1997.
- [**Pet84**] F. Petit, Introduction à la psychosociologie des organisations, Privat, 1984.
- [**Pet02**] D. Petriu, Applying SPT profile for performance analysis, MDA for Embedded System Development, Brest, 16-20 September 2002.
- [**Pic04**] G. Picard, Méthodologie de développement de systèmes multiagents adaptatifs et conception de logiciels à fonctionnalité émergente, Institut de Recherche en Informatique de Toulouse(IRIT), décembre 2004.
- [**Que02**] R. Querrec, Les systèmes multi-agents pour les environnements virtuels de formation, application à la sécurité civile, Thèse de doctorat de l'université de Bretagne Occidentale, 2002.
- [**Rao95**] A.S. Rao, et M.P. Georgeff, BDI agents, from theory to practice, In Victor Lesser, editor, Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, 312-319, 1995.
- [**Reg99**] G. Reggio, et R. Wieringa, Thirty One Problems in the Semantics of UML 1.3 Dynamics, In Conference on Object Oriented programming, Systems, Languages and Applications (OOPSLA'99)-Workshop "Rigorous Modeling an Analysis of the UML Challenges and Limitations", 1999.
- [**Ric02**] P.M. Ricordel, et Y.V. Demazeau, a Vowels-Oriented Multi-agent Platform. Revised Papers from the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems, Springer-Verlag, 253–262, 2002.
- [**Riv09**] J.E. Rivera, F. Durán, et A. Vallecillo, Formal specification and analysis of domain specific models using maude, Simulation, Vol. 85(11-12), 778–792, 2009.
- [**Rol10**] M. Roldan, et F. Duran, Dynamic validation of ocl constraints with modcl. In Workshop on OCL and Textual Modelling, Vol. 44, 2010.
- [**Rom07**] J.R. Romero, J.E. Rivera, F. Durán, et A. Vallecillo, Formal and tool support for model driven engineering with maude, Journal of Object Technology, Vol. 6(9), 187–207, 2007.
- [**Rou98**] J. Rouchier, et F. Bousquet, Non-merchant economy and multi-agent system: an analysis of structuring exchanges, Springer Verlag, 111–112, 1998.
- [**Rus13**] V. Rusu, Embedding domain-specific modeling languages into maude specifications. Software and Systems Modeling, Vol.12(4), 847–869, 2013.
- [**Sad04**] B. Sadeg, Contributions à la gestion des transactions dans les SGBD temps réel, Université Du Havre Ufr Des Sciences Et Techniques, 26 Novembre 2004.
- [**Sad14**] D. Sadoun, Des spécifications en langage naturel aux spécifications formelles via une ontologie comme modèle pivot, Thèse de doctorat, Université Paris Sud-Paris XI, 2014.
- [**Saw07**] J. Sawamoto, H. Tsuji, K. Hiratsuka, et H. Koizumi, A proposal of multi-agent system development framework for cooperative problem solving and its experimental evaluation, Int. J. High Perform. Comput. Network, 5, 2007.
- [**Sch98**] P.C. Scholl, M.C. Fauvet, J.F. Canavaggio, Un modèle d'historique pour un SGBD temporel, TSI, Vol. 17(3), 379-399, 1998.

- [Sco02] W.R. Scott, *Organizations, Rational, Natural and Open Systems*. 5th Edition, Prentice Hall, 2002.
- [Sel94] B. Selic, G. Gullekson, et P. T. Ward, *Real-Time Object-Oriented Modeling*, New York, Wiley, Hardcover, ISBN 0471599174, 560, 1994.
- [Sen05] R. Senanayake, G. Denker, et J. Pearce, Towards integrated specification and analysis of machine-readable policies using maude 1, In *Workshop on Semantic Web and Policies at International Semantic Web Conference*, 2005.
- [Sho98] P.C. Scholl, M.C. Fauvet, J.F. Canavaggio, Un modèle d'historique pour un SGBD temporel, *TSI*, Vol. 17(3), 379-399, 1998.
- [She99] W. Shen, et D.H. Norrie, Agent-based systems for intelligent manufacturing: A state of the art survey, *Knowledge and information systems*, Vol. 1(2), 129-156, 1999.
- [Sic05] J.S. Sichman, V. Dignum, et C. Castelfranchi, Agents organizations: a concise overview, *Braz. Comp. Soc., Spec. Issue Agents Organ*, Vol. 11, 1-8, 2005.
- [Sie97] C. Sierra, N. R. Jennings, P. Noriega, et S. Parsons. A framework for argumentation-based negotiation, In: *International Workshop on Agent Theories, Architectures, and Languages*, Springer, Berlin, Heidelberg, 177-192, 1997.
- [Sin94] M.P. Singh, *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications-LNAI* Springer, Verlag, Heidelberg, Germany, Vol. 799, 1994.
- [Son12] Y. Song, R. Chen, et Y. Liu, A non-standard approach for the owl ontologies checking and reasoning, *Journal of Computers (JCP)*, Vol.7(10), 2454-2461, 2012.
- [Spi92] J.M. Spivey, *The Z Notation: A Reference Manual*, Prentice Hall International Series in Computer Science, 1992.
- [Sta88] J.A. Stankovic, Misconceptions about real-time computing: a serious problem for the next-generation systems, *IEEE Computer*, vol. 21(10), 10-19, 1988.
- [Str95] E. Le Strugeon, *Une méthodologie d'auto-adaptation d'un système multi-agents cognitifs*. Thèse de doctorat, Université de Valenciennes, 1995.
- [Stu03] A. Sturm, D. Dori, et O. Shehory. Single-model method for specifying multi-agent systems, In *Proceedings of AAMAS'03*, 2003.
- [Tam97] M. Tambe, Towards flexible teamwork. *Journal of Artificial Intelligence Research*, Vol.7, 83-124, 1997.
- [Tra07] J. Tranier, *Vers une vision intégrale des systèmes multi-agents: Contribution à l'intégration des concepts d'agent, d'environnement, d'organisation et d'institution*. Thèse de doctorat. Université Montpellier II-Sciences et Techniques du Languedoc. 2007.
- [Tre97] J.P. Treuil, et C. Mullon, *Expérimentation sur mondes artificiels: pour une réflexion méthodologique*, Editions Scientifiques et Médicales ELSEVIER, 1997.
- [Tro11] J. Troya, et A. Vallecillo, A Rewriting Logic Semantics for ATL, *Journal of Object Technology*, Vol. 10(5), 1-29, 2011.
- [Tru06] N.T. Truong, *Utilisation de B pour la vérification de spécifications UML et le développement formel orienté objet*, Thèse de doctorat, Université Nancy II, 2006.

- [Tve01]** A. Tveit, A survey of agent-oriented software engineering, In NTNU Computer Science Graduate Student Conference, Norwegian University of Science and technology, May 2001.
- [Van06]** M.B. Van Riemsdijk, F.S. De Boer, M. Dastani, et J.J.C. Meyer, Prototyping 3APL in the Maude term rewriting language, in proceedings of the 7th International Workshop on Computational logic in multi-agent systems (CLIMA VII), Hakodate, Japan, 95–114, 8-9 May 2006.
- [Ver05]** A. Verdejo, N. Martí-Oliet, T. Robles, J. Salvachúa, L. Llana, et M. Bradley, Transforming information in rdf to rewriting logic, In Formal Methods for Open Object-Based Distributed Systems, Vol. 3535, 227–242, 2005.
- [Wag03]** G. Wagner, et F. Tulba, Agent-Oriented Modeling and Agent-Based Simulation, In Proceedings of 5th Int Workshop on Agent-Oriented Information Systems (AOIS-2003), ER2003 Workshops, Springer-Verlag, LNCS. In P. Giorgini and B. Henderson-Sellers (Eds.), 2003.
- [Wah03]** T. Wahl, Introduction to Model Checking. Presentation Slides, Spring 2003. ISRN I3S/RR-2003-26-FR
- [War99]** J. B. Warmer, A. G. Kleppe, The Objec Constraint Language– Precise Modeling with UML. Addison-Wesley, 1999.
- [Wil04]** S. Wilson, B. Olivier, S. Jeyes, A. Powell, et T. Franklin, A Technical Framework to Support e-Learning, Technical report, JCLT, 2004.
- [Woo99]** M. Wooldridge, Intelligent Agents, In: Weiss, G. (Ed.): Multiagent Systems– A Modern Approach to Distributed Artificial Intelligence, MIT Press, Cambridge, Massachusetts, 27-78, 1999.
- [Woo00a]** M. Wooldridge, N.R. Jennings, Agent-Oriented Software Engineering, in Handbook of Technology (ed. J. Bradshaw) AAAI/MIT Press, 2000.
- [Woo00b]** M. Wooldridge, N.R. Jennings, et D. Kinny, The Gaia Methodology for Agent-Oriented Analysis and Design, Journal of Autonomous Agents and MAS, Vol. 3(3), 285-312, 2000.
- [Woo00c]** M. Wood, Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems, Air Force Institute of Technology, MS Thesis, 2000.
- [Woo01]** M. Wooldridge, P. Ciancarini, Agent-oriented software engineering: the state of the art. Editeurs, P. Ciancarini and M. Wooldridge, Agent-Oriented Software Engineering, Springer-Verlag, Vol. AI, Janvier 2001.
- [Woo02]** M. Wooldridge, An Introduction to Multiagent Systems, Editeurs: John Wiley & Sons Ltd, England, ISBN 047149691X, 2002.
- [Xia95]** C. Xiaohui, J. Song, W.S. Liu, Maintaining Temporal Consistency: Pessimistic vs. Optimistic Concurrency Control. IEEE transactions on knowledge and data engineering, 1041-4347/95, October 1995, Vol. 7(5), 786-796.
- [Zam98]** F. Zambonelli, N. Jennings, et M. Wooldridge, Developing multiagent systems: The Gaia methodology, ACM Trans Software Eng. Meth, Vol. 12(3), 417 – 470, 2003.
- [Zam00]** F. Zambonelli, N. R. Jennings, M. Wooldridge, Organisational Abstractions for the Analysis and Design of Multi-Agent Systems, Proc 1st Int Workshop on Agent-Oriented Software Engineering, Limerick, Ireland, 127-141, 2000.

[Zam05] F. Zambonelli, N.R. Jennings, M. Wooldridge, Multi-Agent Systems as Computational Organizations: The Gaia Methodology, In Henderson-Sellers, B., Giorgini, P. (Eds.), Agent-oriented Methodologies. Idea Group, 136–171, 2005.

[Zha06] L. Zhang, Development Method for Multi-Agent Real Time Systems, Faculty of Computer Science and Technology Guangdong, University of Technology, International Journal of Information Technology, Vol.12(6), 2006.

Liste des Productions Scientifiques

Publications internationales

1/. Un Article Intitulé: A formal framework for organization- centered multi-agent system specification: A rewriting logic based approach.

Journal: Multi-agent and Grid Systems, vol. 13, no. 4, pp. 395-419, 2017

URL : <https://content.iospress.com/articles/multiagent-and-grid-systems/mgs277>

Publisher: IOS PRESS.

Communications Internationales

2/. Un article intitulé: Towards an Organizational Model for Real Time Multi-Agent System Specification. Publié dans les proceedings de la conférence "Science and Information (SAI) 2013" London UK, indexée sous **IEEE Xplore**.

3/. Un book chapter intitulé: A Novel Organizational Model for Real Time MAS: Towards a Formal Specification. Publié dans Springer Book Series - Studies in Computational Intelligence (Volume 542, 2014, pp 171-180).

Publisher: Springer Verlag.

URL : <https://www.scopus.com/record/display.uri?eid=2-s2.0-84958527918&origin=inward&txGid=B5F6D897093EBB3C2C5402E1C276396E.wsnAw8kcdt7IPYLO0V48gA%3a2#>

Communications Nationales

4/. Un short paper intitulé: A Formal Specification Approach for Real Time Multi-Agent System Organizational Requirements. Présenté aux journées d'étude sur les systèmes complexes JESC'2015 à l'université d'Oum El Bouaghi-Algérie 28-29 Avril 2015.

الملخص

حالياً، تعتبر هندسة البرمجيات الموجهة للأعوان، مجال بحثي نشط للغاية. لسنوات عديدة، لدى مصممي الأنظمة متعددة الأعوان منهجيات تطوير ولغة نمذجة دون أن تعكس القيود الزمنية المختلفة التي قد يكون عليها الأنظمة متعددة الأعوان ذات الوقت الحقيقي. في الواقع، ليس من السهل تصوّر مفاهيم الأنظمة متعددة الأعوان ذات الوقت الحقيقي باستخدام أساليب هندسة البرمجيات التقليدية. وبناء على ذلك، فإن قضية بحثية حرجية لمجتمع عامل الوقت الحقيقي تمثلت في تعريف النموذج التنظيمي المناسب لتحليل وتصميم خصائصها الأساسية. لقد كان التقدم في هذا المجال بطيئاً ولكنه ثابت بشكل عام. تعتبر معظم المقترحات المنظمة ككيان لها غرض محدد أو هدف عام دون الأخذ بعين الاعتبار التقييد الزمني لعامل الوقت الحقيقي. ثم يستخدمون مجموعة من المفاهيم التنظيمية، على سبيل المثال، الأدوار، الأعراف، المجموعات، التفاعلات، وما إلى ذلك؛ للسيطرة على سلوكيات الوكلاء وتعديلها فيما يتعلق بالهدف العام للمنظمة.

في رأينا، هذه الرؤية للتنظيم غير كاملة. نقترح تمديدها بقيد زمني من خلال توسيع مخططات عون لغة النموذج الموحدة [Lao13]. وعلى وجه الخصوص، نعتقد أن مفهوم التقييد الزمني الذي يجب على وكيل الوقت الحقيقي احترامه أثناء دورة حياته أمر مهم للغاية ليتم إدراجه في المنظمة.

نهجنا يأخذ تصوير الشخصيات الاجتماعية جنباً إلى جنب مع وكيل القيود الزمنية باعتبارها جوهر نمذجة الأنظمة متعددة الأعوان ذات الوقت الحقيقي ويقدم أول فكرة عن النموذج التنظيمي لتحديد الأنظمة متعددة الأعوان ذات الوقت الحقيقي. إنه يسمح بتمثيل الجانب الوظيفي والجوانب الثابتة والجوانب الديناميكية لـ الأنظمة متعددة الأعوان ذات الوقت الحقيقي بناءً على المنظور التنظيمي. ومع ذلك، فإن إضفاء الطابع الرسمي على الجوانب التنظيمية لـ الأنظمة متعددة الأعوان ذات الوقت الحقيقي هو في رأينا طريقة مهمة لكل من أنشطة التحليل والتصميم. علاوة على ذلك، يتطلب تصميم الأنظمة متعددة الأعوان ذات الوقت الحقيقي المشاركة باللغات الرسمية. من بين هذه اللغات: ربما تكون آرتي مود [Olv07] أفضل اللغات المعروفة والأكثر استخداماً لكل من البرمجة الوظيفية. علاوة على ذلك، آرتي مود قوية للغاية من حيث المواصفات والتحقق من الأنظمة المترامنة، مما يجعلها مرشحاً جيداً للمواصفات والتحقق من الأنظمة متعددة الأعوان ذات الوقت الحقيقي. مع العلم أنه لا يوجد حالياً أي عمل يطبق آرتي مود على المواصفات التنظيمية الرسمية للأنظمة متعددة الأعوان ذات الوقت الحقيقي وعلى كل من تطبيقات الوقت الحقيقي.

في الواقع، يمثل هذا الإطار التنظيمي الرسمي نهجاً هاماً لنمذجة الأنظمة متعددة الأعوان ذات الوقت الحقيقي، وهو امتداد حديث يجمع بين مخططات عون لغة النموذج الموحدة الموسعة [Lao13]، مع مواصفات آرتي مود [Lao14] لدعم المواصفات الرسمية لـ الأنظمة متعددة الأعوان ذات الوقت الحقيقي التنظيمية. في هذا الإطار، يتم توحيد نقاط القوة في كلا النهجين، ولكنها لا تزال تفتقر إلى المزيد من التفاصيل التنظيمية التي يجب أخذها بعين الاعتبار. لذلك، تم وصف منهج قائم على إعادة كتابة المنطق لنماذج النموذج الوكيل [Fer03a] و نموذج مويس+ [Hub02a] لمؤسسة أنظمة متعددة العوامل لتكون قادرة على التفكير، من ناحية، حول النموذج التنظيمي، ومن ناحية أخرى، حول دلالاتها العامة. علاوة على ذلك، تم بحث إمكانية تضمين البعد الزمني للتعامل مع العوامل المعيارية للقيود الزمنية في نموذج مويس+ لتحديد نموذج ملموس لنظام الأنظمة متعددة الأعوان ذات الوقت الحقيقي من خلال لغة مود في الوقت الفعلي. سمح لنا هذا بأن يكون لدينا في جانب واحد نموذجاً تنظيمياً يسهل تصميم الأنظمة متعددة الأعوان ذات الوقت الحقيقي وعلى الجانب الآخر إطاراً للمواصفات رسمياً يوفر دعماً أفضل لعمليات التحقق الخاصة بهم. من أجل التحقق من صحة هذا العمل، قمنا بتطبيق عملية الترجمة لمحاكاة المواصفات الرسمية مع سيناريوهات التفاعل المحددة مسبقاً، وفي حالتنا اخترنا مثالين، أحدهما يتعامل مع إدارة السلاسل اللوجستية والآخر مع إدارة المؤتمرات.

الكلمات الرئيسية : النموذج التنظيمي؛ الأنظمة متعددة الأعوان ذات الوقت الحقيقي؛ مواصفات رسمية؛ عون لغة النموذج الموحدة؛ آرتي مود ؛ منطق إعادة الكتابة ؛ إدارة السلاسل اللوجستية؛ نظام إدارة المؤتمرات.

Abstract

Currently, Agent Oriented Software Engineering (AOSE) is a very active research domain. For many years, MAS designers have development methodologies and modeling language without reflects the different temporal restrictions that RT-MAS may have. Indeed, it is not easy to conceptualize RT-MAS concepts using conventional agent software engineering approaches. Consequently, a critical research issue for the real time agent community has been the definition of suitable organizational model for analyzing and designing their main properties. The progress in this area has been slow but generally steady. Most of the proposals consider an organization as an entity that has some concrete purpose or global objective without taking in to account the temporal restriction of real time agent. Then they use a set of organizational concepts, e.g. roles, norms, groups, interactions, etc; to control and modify the dynamics or agents behaviors with regard to the global objective of the organization.

In our opinion, this vision of organization is incomplete. We propose to extend them with temporal constraint by extending AUML diagrams [Lao13]. In particular, we believe that the concept of temporal restriction that real time agent must respect during their life cycle is very important to be included in organization.

Our approach takes the depiction of social characters combined with agent temporal restriction as the core of RTMAS modeling and presents the first idea of an organizational model to specify RTMAS. It allows representing functional aspect, static aspects, and dynamic aspects of RTMAS based on organizational perspective. However formalizing the organizational aspects of RT-MAS is in our opinion, an importance way for both analysis and design activities. Furthermore, the RT-MAS design requires the involvement with formal languages. Among these languages: RT-Maude [Olv07] is probably the best known and most widely used languages for both functional and object oriented programming. Furthermore, RT-Maude is very powerful in terms of specification, validation and verification of concurrent systems, making it a good candidate for specification and validation of RT-MAS. Knowing that, there is currently no work applying RT-Maude to RT-MAS formal organizational specification and both to real time applications.

In fact, this formal organizational framework represents an important approach for modeling RT-MAS, it is a recent extension which combines the AUML extended diagrams [Lao13], with RT-Maude specification [Lao14] to support the formal specification of RT-MAS organizational requirements. In this framework, the strengths of both approaches are unified, but it still lacks more organizational details to be considered. So, a rewriting logic based approach is described for prototyping the Agent-Group-Role [Fer03a] and Moise+ [Hub02a] models for multi-agent systems organization to be able to reason, on the one hand, about the organizational model, and on the other hand, about its general semantics. Furthermore, the possibility of including a temporal dimension to deal with agents temporal constraints issues was investigated in Moise+ model to specify the concrete model of Real-Time Multi-Agent System through Real-Time Maude language. This allowed us to have on one side an organizational model facilitating the design of the RTMAS and on the other side a formal specification framework providing providing better support for their verification processes. For the validation of this work we applied the translation process to simulate the formal specifications with predefined interaction scenarios, in our case we chose two examples, one dealing with supply chain management and the other with conference management.

Keywords: Organizational model; Real-Time Multi-Agent System ; Formal Specification; Agent UML; Real-Time Maude; Rewriting Logic; Supply Chain Management; Conference Management system.

Résumé

Actuellement, l'ingénierie orientée agent est un domaine de recherche très actif. Depuis de nombreuses années, les concepteurs de Système Multi-Agents (SMA) ont des méthodologies de développement et des langages de modélisation qui ne reflètent pas les différentes contraintes temporelles qu'un Système Multi-Agents Temps Réel (SMATR) peut avoir. En effet, il n'est pas facile de conceptualiser les concepts SMATR en utilisant des approches d'ingénierie logicielle conventionnelles. Par conséquent, un problème de recherche critique pour la communauté des agents temps réel est celui de la définition d'un modèle organisationnel approprié pour l'analyse et la conception de leurs propriétés principales. Les progrès dans ce domaine ont été lents mais généralement stables. La plupart des propositions considèrent une organisation comme une entité qui a un but concret ou un objectif global sans prendre en compte les restrictions temporelles des agents temps réel du système. Elles utilisent un ensemble de concepts organisationnels, comme rôles, normes, groupes, interactions, etc. pour contrôler et modifier les comportements des agents par rapport à l'objectif global de l'organisation.

À notre avis, cette vision de l'organisation est incomplète. Nous proposons de les étendre pour tenir compte des contraintes temporelles en étendant les diagrammes AUML [Lao13]. En particulier, nous croyons que le concept de restriction temporelle qu'un agent temps réel doit respecter au cours de son cycle de vie est très important pour être inclus dans l'organisation.

Notre approche illustre bien la représentation des concepts organisationnels conjointement avec les restrictions temporelles des agents comme la base de la modélisation du SMATR, proposant ainsi un nouveau modèle organisationnel pour spécifier les SMATR. Elle permet de représenter les aspects fonctionnels, statiques et dynamiques du système suivant la perspective organisationnelle. Cependant, formaliser les aspects organisationnels de SMATR est, à notre avis, un moyen important pour les activités d'analyse et de conception. De plus, la conception de SMATR nécessite l'implication de langages formels. Parmi ces langages: Real Time Maude (RT-Maude) [Olv07] est un bon candidat pour la spécification de SMATR car il est très puissant en termes de spécification, de validation et de vérification formel des systèmes concurrents orientés objet. Sachant qu'il n'y a pas de travail qui applique RT-Maude aux spécifications organisationnelles formelles de SMATR.

En effet, le framework organisationnel formel obtenu représente une extension récente qui combine les diagrammes AUML étendus [Lao13], avec la spécification RT-Maude [Lao14] pour supporter la spécification formelle de SMATR organisationnel. Dans ce framework, les points forts des deux approches sont unifiés mais il manque encore plus de détails organisationnels à prendre en considération. Pour palier à cela nous avons appliqué notre approche basée sur la logique de réécriture pour prototyper les modèles Agent-Groupe-Rôle(AGR) [Fer03a] et Moise+ [Hub02a] de SMA centré organisation afin de pouvoir raisonner, d'une part, sur le modèle organisationnel et d'autre part sur sa sémantique générale. Toutefois, nous avons étudié la possibilité d'inclure une dimension temporelle pour traiter les problèmes de contraintes temporelles dans le modèle Moise+ afin de spécifier le modèle concret du SMATR en utilisant le langage RT-Maude ce qui nous a permis d'avoir d'un côté un modèle organisationnel facilitant la conception des SMATR et de l'autre côté un framework de spécification formelle offrant un meilleur support à leurs processus de vérification. Pour la validation de ce travail nous avons appliqué le processus de translation pour simuler les spécifications formelles avec des scénarios d'interaction prédéfinis, dans notre cas on a choisi deux exemples l'un portant sur la gestion de chaîne logistique et l'autre concernant la gestion de conférence.

Mots clés : Modèle organisationnel ; Système Multi-Agents temps réel; Spécification formelle; Agent UML; Real Time Maude ; Logique de réécriture ; Gestion de chaîne logistique; Système de Gestion de conférence.