République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique
Université Mentouri – Constantine

Faculté des Sciences de l'Ingénieur
Département d'Informatique

N° d'ordre : 40 / te / 2006
Série : 02 / inf / 2006

Thèse de Doctorat d'état en Informatique

Thème

Modélisation multi-agent du Processus logiciel

Présentée par :
Monsieur Ammar LAHLOUHI

Dirigée par :
Professeur Zaidi SAHNOUN

Soutenue le 18 Juin 2006

Devant le jury :

Mahmoud BOUFAIDA	Professeur, Université de Constantine	Président
Zaidi SAHNOUN	Professeur, Université de Constantine	Rapporteur
Zahia GUESSOUM	Maître de conférences (HDR), Laboratoire LIP6	Examinatrice
Mohamed Tayeb LASKRI	Professeur, Université de Annaba	Examinateur
Mohamed AHMED-NACER	Professeur, Université USTHB d'Alger	Examinateur

REMERCIEMENT

Je tiens à remercier les membres du jury d'avoir consacré de leur temps à examiner ma thèse malgré leurs nombreuses responsabilités :

Zaidi SAHNOUN, Professeur de l'université de CONSTANTINE, directeur du laboratoire d'informatique répartie LIRE et directeur de ma thèse. Au moment où il a refusé beaucoup de demandes, il a accepté de me diriger et, dès le début, il m'a intégré dans tous ses projets (équipes de recherche, projets de coopération scientifique franco-algérienne avec le CNAM de Paris, projet européen CRISTEL, etc.). Je tiens à lui exprimer ma reconnaissance et ma sincère gratitude.

Mahmoud BOUFAIDA, Professeur et président du conseil scientifique de la faculté des sciences de l'université de CONSTANTINE. Je le remercie pour avoir lu la première version de ma thèse et avoir accepté de présider le jury.

Zahia GUESSOUM, membre du laboratoire d'informatique de Paris 6 (LIP6) de l'université Pierre et Marie Curie, France. Elle m'a encouragé et elle a été toujours disposée à m'aider. Je la remercie pour avoir accepté d'examiner ma thèse tout en lui portant une attention particulière.

Mohamed-Tayeb Laskri, Professeur et recteur de l'université de ANNABA. J'ai toujours trouvé auprès de lui respect et gentillesse. Il a déjà fait partie du jury de mon magister. Je le remercie alors une autrefois pour avoir accepté de faire partie du jury de ma thèse.

Mohamed AHMED-NACER, Professeur de l'université des sciences et des technologies Houari BOUMEDIENNE d'Alger. Ses travaux à l'institut d'informatique et mathématiques appliquées de Grenoble (IMAG), France, sur l'évolution des schémas m'ont beaucoup servi durant mon Magister. Nos rencontres dans le cadre du projet CRISTEL ont été toujours agréables. Je le remercie pour avoir accepté de rapporter ma thèse.

Je tiens aussi à remercier les personnes qui m'ont aidé et/ou influencé ma vision de recherche :

Said GHOUL de l'université Philadelphia de Jordanie. En tant que mon encadreur de magister, il m'a initié à la recherche. J'ai trouvé auprès de lui, à chaque fois que j'en y besoin, conseils et chaleur humaine.

Mohamed BETTAZ, Professeur et doyen de la faculté de la technologie de l'information de l'université Philadelphia de Jordanie. Il a toujours considéré nos discussions avec sérieux, respect et gentillesse. J'ai appris de lui de partir toujours des origines des idées et d'exprimer mes pensés en terme de modèles bien fondés avant de les valider formellement et/ou expérimentalement.

Malik MALLEM, professeur de l'université d'Evry (France). Qu'il trouve ici l'expression de ma reconnaissance pour tout ce qu'il a fait pendant le stage que j'ai passé dans son équipe au laboratoire des systèmes complexes (LSC). Malgré que la problématique de mes recherches soit différente de ses domaines d'intérêt, il a tout fait pour m'aider.

Karim DJOUANI, professeur de l'université Paris 12, et Kamal BARKAOUI, Professeur du centre national des arts et métiers CNAM de PARIS. Ils m'ont toujours accueilli chaleureusement dans nos rencontres du projet CRISTEL.

Jean Michel ILIE, chercheur du laboratoire LIP6. Il a toujours porté un intérêt particulier à ce que je lui présente au point que, durant notre dernière rencontre au LIP6, il a préféré poursuivre notre discussion au détriment de son déjeuné.

Yves DEMAZEAU, chercheur du CNRS de Grenoble, France. Son appréciation de ma démarche m'a beaucoup encouragée.

Les membres du laboratoire LIRE où j'ai effectué ma thèse. J'ai toujours trouvé chez eux un accueil chaleureux. Je cite, en particulier, mon ami Ramdhane MAAMERI qui m'a beaucoup aidé dans la préparation de ma soutenance.

Soheib BAARIR, doctorant à LIP6. Il était toujours disponible à chaque fois que j'ai besoins de lui.

Enfin, je tiens à remercie toutes les autres personnes qui m'ont aidé de près ou de loin et que je n'ai pas cité ici.

RESUME

Le processus logiciel peut être considéré comme un système logiciel auquel on peut appliquer des techniques et des méthodes utilisées dans le développement des logiciels. Les travaux relatifs à la modélisation du processus logiciel ne prennent pas en considération, de façon cohérente, les développeurs de logiciels et leur coopération. De plus, ces travaux le modélisent soit par des modèles classiques de développement de logiciels soit par des modèles multi-agent mais de façon empirique. Dans notre thèse, nous considérons le processus logiciel comme un système multi-agent et nous utilisons une méthodologie multi-agent pour sa modélisation. Les méthodologies multi-agent existantes ne permettent pas de prendre en charge convenablement les développeurs et leur coopération. Nous avons alors développé une nouvelle méthodologie, appelé MASA-Method, que nous avons utilisé dans la modélisation multi-agent du processus logiciel. Comparée à certaines méthodologies multi-agent, MASA-Method présente certains avantages. Elle est complète et mixte. Elle prend en considération l'intégration des agents logiciels et humains et la détection de la terminaison des systèmes multi-agent. En plus de son utilisation dans le développement de plusieurs systèmes multi-agent spécifiques aux environnements de développement de logiciels, MASA-Method a été utilisée également dans la réalisation de systèmes multi-agent de différentes natures incluant la simulation multi-agent de la robotique collective, la recherche coopérative d'informations distribuées impliquant des agents mobiles et l'intégration multi-agent de robots et d'humains.

A ma famille

Table des matières

Liste des Figures

Liste des Tables

Chapitre 1

Introduction

Les logiciels sont d'une importance capitale dans la vie quotidienne et professionnelle. Chaque jour, on découvre de nouvelles applications de logiciels au point que nous sommes entourés par toutes sortes d'équipements électroniques équipés de logiciels (manufactures, espace, Web, mobile, bureau, maison, jouets, montres...). Cependant, l'industrie logicielle a connue depuis toujours de nombreuses difficultés. Les logiciels qu'elle produit sont de mauvaise qualité et délivré en retards avec des budgets dépassant les prévisions. De telles difficultés associées à l'accroissement de la complexité des logiciels font augmenter les coûts des logiciels. Dans sa thèse [Wang 01], Wang rapporte qu'approximativement 6 % du produit national brut (PNB) des nations industrielles est dépensé dans l'industrie logicielle chaque année. Ces dépenses avoisinent 1000 milliards de dollars US. Le génie logiciel vise l'étude des approches et des méthodes scientifiques rigoureuses permettant de réduire les coûts de développement de logiciels et d'améliorer leur qualité.

Au début de l'ère du génie logiciel, on pensait que l'amélioration des formalismes d'expression des modèles de logiciel, tels que les formalismes mathématiques et les notations graphiques, est suffisante pour remédier aux difficultés de l'industrie logicielle. Avec le cumule des expériences et des échecs, on réalisât l'incapacité de ces modèles à eux seuls de remédier aux difficultés de l'industrie logicielle et on découvrit, par là, l'importance du processus logiciel. Le processus logiciel doit être modélisé pour le simuler, l'amélioré, l'utiliser et, l'automatiser. L'objectif général de cette thèse est la modélisation multi-agent du processus logiciel.

1.1 Contexte et motivations

Les thèmes abordés dans cette thèse sont relatifs à la modélisation multi-agent du processus logiciel. Ils sont ceux des processus logiciel, des systèmes multi-agent et de

la modélisation multi-agent du processus logiciel.

1.1.1 Processus logiciel

Le mot "processus logiciel"[1] a été inventé en 1969 par Lehman [Lehman 69] qui l'a qualifié de "processus de programmation"[2]. Lehman l'a défini comme étant "toute la collection de technologies et activités qui transforment le germe d'une idée en robinet binaire e de programme"[3]. Le processus logiciel a été inspiré de la chaîne de production industrielle. Il a été modélisé pour le comprendre, guider les personnes impliquées, l'améliorer et l'automatiser partiellement. C'est durant la période entre le milieu des années quatre-vingt et le milieu des années quatre-vingt dix que le processus logiciel a connu un intérêt croissant. Plusieurs travaux ont essayé de proposer des approches au problème de modélisation du processus logiciel. Dans cette optique, Osterweil a publié en 1987 un papier intitulé "Software processes are software too" [Osterweil 87]. Il stipule que le processus logiciel est un système logiciel complexe auquel on peut appliquer des méthodes et des techniques utilisées dans le développement de logiciels. Le logiciel "processus logiciel" exige l'accomplissement d'activités de diverses natures que nous qualifions, par la suite, de moyens automatiques. D'autres activités, par contre, ne sont pas automatisables. Elles doivent être accomplies exclusivement par des humains. Le processus logiciel doit intégrer les moyens automatiques et les humains et supporter leur coopération.

Trois problématiques intéressantes doivent être prises en considération par la modélisation du processus logiciel. Ces problématiques sont celles relatives à l'intégration des humains et des moyens automatiques et à leur coopération ainsi que leur évolution. Actuellement, ces problématiques ne sont pas prises en charge de façon naturelle. Dans leur papier [Cugola 98], Cugola et Ghezzi déclarent que "Les processus centrés humain sont caractérisés par deux aspects cruciaux qui ont été en grande partie ignorés par la plupart des recherches en processus de logiciel : Ils doivent supporter la coopération des personnes, et ils doivent être fortement flexibles"[4]. Robert Balzer [Balzer 00] déclare également que les deux principaux problèmes de la technologie existante du processus logiciel sont qu'elle ne donne pas aux développeurs de logiciels le support du processus coopératif adapté et elle ne fait pas face aux changements du processus logiciel. Ces deux visions étaient encore supportées par Conradi, Fugetta et Jaccheri

[1] Traduction française du terme "Software process"

[2] Traduction française du terme "programming process"

[3] Traduction par l'auteur du text "the total collection of technologies and activities that transform the germ of an idea into a binary program tape"

[4] Traduction par l'auteur du text "Human-centred processes are characterised by two crucial aspects that were largely ignored by most software process research: They must support cooperation among people, and they must be highly flexible."

[Conradi 98] qui proclament que le processus de logiciel a échoué de s'adapter aux parties du développement de logiciel impliquant des humains.

1.1.2 Systèmes multi-agent

Le concept d'agent est hérité de celui de l'humain tout en essayant de tenir compte des facultés de l'humain tels que l'activité, l'intelligence et l'action sur l'environnement. Inclure toutes ces facultés dans un seul modèle est une tâche difficile voir impossible. Plusieurs communautés d'agents ont été alors émergées. Chacune des communautés définit une catégorie d'agents (intelligents, logiciels, mobiles, adaptatifs, robotiques, etc.) qu'elle étudie en considérant seulement certaines caractéristiques considérées intéressantes pour les applications de leurs intérêts. Plusieurs définitions de l'agent ont été alors proposées sans qu'elles soient complètement concordantes.

Un SMA (pour système multi-agent) est un ensemble d'agents qui communiquent pour accomplir une tâche collective de façon coopérative. Les agents d'un SMA sont autonomes mais ils doivent travailler en coopération. Un SMA a agents hétérogènes intègre par coopération des agents de diverses natures (humains, robots et logiciels). La coopération dans un SMA à agents hétérogènes peut être entre des humains tels qu'elle est faite habituellement dans la communauté des humains ou entre des agents logiciels. Elle peut aussi être entre les humains et les agents logiciels.

Les SMA ont connu un intérêt croissant pendant les quinze dernières années. Cependant, dans la plupart du temps, le développement de SMA a été fait empiriquement. Un tel développement ne permet pas de justifier de façon claire les décisions prises durant la conception. L'utilisation d'une méthodologie de développement de SMA est, non seulement, souhaitable mais elle est nécessaire. Cette nécessité devient encore plus importante quand il s'agit d'un développement de grands logiciels. La justification des connaissances méthodologiques est une tâche difficile. Elle doit utiliser des connaissances empruntées de plusieurs disciplines. De plus, ces connaissances exigent des évaluations sérieuses et de longues expérimentations dans le développement de systèmes qui doivent être de diverses natures. Une méthodologie doit aussi inclure des aspects pratiques permettant de simplifier le développement d'applications. Elle doit aussi disposer d'outils d'aides au développement spécifiques à elles. Ces facteurs ont fait que le développement d'une méthodologie est difficile et long. Enfin, l'intégration des agents hétérogènes dans un seul SMA a été détournée dans le développement de ces systèmes. On pensait que la considération des humains et des moyens automatiques comme agents suffira à définir la coopération entre eux de façon cohérente.

1.1.3 Modélisation multi-agent du processus logiciel

La section 1.1.1 a permis de souligner trois principales problématiques des processus logiciels qu'il est intéressant de résoudre. Ces problématiques n'ont pas été prises en considération d'une manière cohérente dans les travaux sur le processus logiciel. Celles-ci sont liées à l'intégration des humains et des moyens automatiques et à leur coopération aussi bien que leur évolution. Certains travaux sur le processus logiciel considèrent les aspects liés aux problématiques des processus logiciels (communication, coopération, etc.) indépendamment des SMA. Par exemple, [Amiour 97, Amiour 98, Amiour 99] considère la communication dans les processus logiciels. D'autres travaux, tels que [Ciancarini 93, Alloui 96, Sahnoun 97, Wang 01], modélisent empiriquement le processus logiciel par un SMA.

Comme cela a été expliqué dans la section 1.1.2, la modélisation multi agent considère les humains et les moyens automatiques comme des agents. Ceci amène à des modèles de SMA à agents hétérogènes dont l'intégration est basée sur la coopération. Pour la modélisation multi-agent du processus logiciel, une évolution des moyens automatiques et du processus logiciel sera une évolution du SMA. Cette évolution est simplifiée par l'autonomie des agents. La modélisation multi-agent permet alors de mieux répondre aux besoins du processus logiciel.

1.2 Problématique et objectifs

Théoriquement, la modélisation multi-agent du processus logiciel permet de satisfaire ses besoins. Cependant, certains de ces besoins sont actuellement des problématiques des SMA. Dans une modélisation, le méta-modèle est supposé satisfaisant, i.e., il répond aux besoins du sujet de modélisation. S'il n'est pas satisfaisant, une solution consiste a développer un premier modèle et de le compléter par des éléments spécifiques manquants. Cependant, il est plus cohérent de faire évoluer le méta-modèle de telle sorte à ce qu'il réponde aux besoins de modélisation dans un cadre plus général. Il est alors plus judicieux de traiter les problématiques des SMA dans le cadre des SMA. En plus de la cohérence de la méthode suivie, plusieurs avantages peuvent être tirés d'une telle approche. Dans cette approche, nous pouvons proposer des solutions plus radicales puisqu'elles seront abordées dans un cadre plus global. Nous pouvons également proposer des solutions plus satisfaisant un besoin plus large puisque la modélisation multi-agent concerne des systèmes autres que le processus logiciel.

Le génie logiciel vise l'application de méthodes scientifiques rigoureuses au développement de logiciels. Il est alors particulièrement important d'utiliser une méthodologie multi-agent pour la modélisation du processus logiciel. Au moins, une méthodologie

permet de justifier les décisions prises durant la modélisation. Au début du travail de cette thèse, de telles méthodologies étaient loin de la maturité nécessaire pour être utilisées dans une telle modélisation. D'ailleurs, cette situation persiste encore malgré l'évolution de ces méthodologies. Il a fallu alors choisir entre deux alternatives. La première consiste à attendre la maturation de ces méthodologies pour choisir celle qui satisfera les besoins du processus logiciel en terme de modélisation. Cette alternative est dangereuse puisque l'attente est susceptible d'être longue. De plus, ces méthodologies peuvent ne pas satisfaire nos besoins particuliers. La deuxième consiste à fonder une nouvelle méthodologie. Cette alternative est également risquée puisque la fondation d'une nouvelle méthodologie est difficile, risque de prendre beaucoup de temps et peut ne pas aboutir.

Dans cette thèse, nous proposons une approche de modélisation multi-agent du processus logiciel et montre sa faisabilité. Elle ne propose pas une réalisation à grande échelle du logiciel "processus logiciel". En ce qui concerne les insuffisances des SMA actuels à modéliser le processus logiciel, nous avons choisi de les aborder dans le cadre des SMA eux-mêmes. Pour l'usage d'une méthodologie, nous avons développé une nouvelle méthodologie appelée MASA-Method [Lahlouhi 02a, Lahlouhi 02b, Lahlouhi 05]. Cette méthodologie répond mieux à nos besoin en modélisation du processus logiciel. L'objectif de cette thèse est la description de la méthodologie MASA-Method, la clarification et la justification de ses aspects et son utilisation dans la modélisation d'un environnement de développement de logiciels. Une méthodologie comporte deux dimensions principales: Un méta-modèle et un processus méthodologiques. Le méta-modèle décrit les concepts à utiliser dans la description des modèles. Le processus méthodologique décrit l'identification des instances de ces concepts et leur combinaison pour former des modèles plus cohérents. Tout au long de cette proposition, Nous avons évité d'introduire de nouveaux concepts. L'intérêt sera alors porté sur les aspects du processus méthodologique, c'est-à-dire, la manière d'identifier des instances de concepts, de les instancier et de les combiner. Notons aussi qu'en ce qui concerne les SMA, on se limitera aux SMA fermés. Le nombre d'agents d'un SMA fermé n'évolue pas durant son fonctionnement. De plus, durant ce fonctionnement, l'environnement d'un SMA fermé ne peut être modifié par des agents autres que ceux de ce SMA.

1.3 Organisation de la thèse

Cette thèse est organisé en six chapitres qui seront consacrés à la description de la méthodologie MASA-Method, à la clarification et à la justification de ses aspects et à son utilisation dans la modélisation d'un environnement de développement de logiciels.

Le chapitre 2 est intitulé "Méthodologies de développement de systèmes multi-

agent". Ce chapitre est un survol des méthodologies existantes et une identification de certaines de leurs incohérences. Il clarifie certains concepts et notions utilisés dans les méthodologies et signale certaines de leurs mauvaises utilisations. Il propose également une classification des méthodologies multi-agent. Cette classification est utilisée pour classer des insuffisances de certaines méthodologies actuelles. Ce chapitre présente aussi notre vision des processus méthodologiques, celle des agents et celles des SMA.

Le chapitre 3 est intitulé "Fondements de MASA-Method". Il décrit le processus méthodologique et les formalismes à utiliser dans le développement des organisations. En particulier, ce chapitre se concentre sur la description procédurale des tâches des organisations et la structuration des organisations en sous organisations. Il présente l'ensemble des formalismes utilisés dans le développement de ces organisations. Ce chapitre décrit aussi les agents d'interfaces et leur utilisation dans l'intégration méthodologique des agents hétérogènes.

Le chapitre 4 est intitulé "Outils pratiques pour l'utilisation de MASA-Method". Ce chapitre présente l'ensemble des moyens de description des différents modèles de MASA-Method et décrit les processus de leurs développements.

Le chapitre 5 est intitulé "Modélisation multi-agent du processus logiciel". Ce chapitre décrit la modélisation multi-agent d'un environnement de développement de logiciels. Il se concentre sur l'utilisation de MASA-Method dans le développement du modèle multi-agent de "développement de logiciels". Il traite, en particulier, l'intégration des agents hétérogènes et le développement des organisations composées. Ce chapitre sert plus particulièrement à la clarification des aspects de MASA-Method et la manière de les utiliser.

Le chapitre 6 est une conclusion. Ce chapitre présente le bilan du travail de thèse sur la modélisation multi-agent du processus logiciel. Il se concentre, plus particulièrement, sur ce qui a été accomplie dans MASA-Method. Il présente ses contributions et effectue une comparaison avec des méthodologies similaires. Finalement, il fournit quelques persepectives du travail proposé.

Chapitre 2

Méthodologies de développement de systèmes multi-agent

Le développement des SMA se fait, très souvent, empiriquement en se basant sur l'expérience et les habilités spécifiques des personnes. Une telle pratique coûte cher et les systèmes résultat sont, d'une manière générale, de mauvaise qualité. De plus, il ne permet pas de justifier d'une manière claire les décisions prises durant la conception. L'utilisation d'un processus de développement de SMA est, non seulement, souhaitable mais nécessaire. Ceci est particulièrement vrai quand ceci concerne de gros logiciels. Ce qui est l'une des missions des SMA. Le développement de SMA nécessite alors une démarche du génie logiciel.

Au début de l'ère du génie logiciel où des modèles fonctionnels de systèmes sont utilisés exclusivement, on pensait que les activités impliquées dans le processus logiciel sont universelles. Par exemple, on pensait que la conception doit être complètement indépendante de l'implémentation. L'introduction de nouveaux modèles, particulièrement, les modèles orientés objet et multi-agent a changé cette donne. Le processus logiciel traditionnel n'a pas pu répondre aux spécificités de ces nouveaux modèles. Des processus logiciel spécifiques ont été alors proposés. Le génie logiciel actuel devient multiple. Il comporte plusieurs génies logiciels spécifiques tels que les génies logiciel fonctionnel, orienté objet et orienté agent. La majorité des anciens logiciels ont été développés selon un génie logiciel fonctionnel. Actuellement, l'industrie logicielle est dominée par le génie logiciel orienté objet. Cependant, cette discipline est devenue orientée agent qui est prometteur pour le futur.

Le reste de ce chapitre est organisé en huit sections. La section 2.1 explique la notion de méthodologie en tant qu'une association entre un méta-modèle et un processus logiciel. Nous insistons sur le fait qu'une méthodologie de développement de systèmes ne peut être claire, complète et précise que si son méta-modèle le soit. Du fait que

le méta-modèle multi-agent, actuellement ne répond pas aux critères précédents, les méthodologies multi-agent ne le sont pas elles aussi. La section 2.2 présente certaines difficultés rencontrées dans le développement d'une méthodologie de développement de SMA. La section 2.3 présente les aspects relatifs au méta-modèle multi-agent. Cette section se concentre sur les aspects des agents en tant que systèmes. La section 2.4 présente les aspects relatifs aux processus méthodologiques. Elle détaille en particulier le processus méthodologique basé rôle. La section 2.5 présente une classification des méthodologies. L'objectif d'une telle classification est de pouvoir classer les insuffisances des méthodologies et leurs avantages. La section 2.6 présente certaines insuffisances communes aux méthodologies organisationnelles actuelles. La section 2.7 décrit des insuffisances spécifiques à certaines méthodologies organisationnelles proche de MASA-Method. La section 2.8 conclut ce chapitre.

2.1 Notion de méthodologie de développement de systèmes

Les méthodologies classiques sont simples et peu nombreuses. La simplicité a facilité l'adoption de ces méthodologies par de nombreux développeurs. Leur nombre a augmenté la demande par rapport à l'offre.De plus, les méthodologies classiques sont basées sur des méta-modèles et des théories et des formalismes bien fondés qui ont participés au succès de ces méthodologies. Par exemple :

1. Les méthodologies fonctionnelles (telles que PSL/PSA [Teichroew 77]) sont basées sur la théorie des fonctions,

2. Les méthodologies basées données (telles que MERISE [Rochfelo 83] et SADT [Ross 85]) sont focalisées sur la modélisation des bases de données basées généralement sur la théorie solide de l'algèbre relationnelle,

3. Les méthodologies orientées objet cherchent un méta-modèle orienté objet bien fondé pour supporter des méthodologies variées. Ceci a mené au méta-modèle UML.

Les trois facteurs précédents ne sont pas favorables aux méthodologies multi-agent. Dans leur forme actuelle, les méthodologies multi-agent sont difficiles, leur nombre est grand et leurs méta-modèles ainsi que leurs bases théoriques ne sont pas bien fondés.

2.1.1 Méthodologie de développement de systèmes comme association d'un méta-modèle et d'un processus méthodologique

Les méthodologies de développement de systèmes proposent des processus méthodologiques basés sur des méta-modèles tels que les méta-modèles classiques fonctionnels, basés données, orientés objet et multi-agent. Une méthodologie est alors une association d'un méta-modèle et d'un processus méthodologique. Le processus méthodologique dirige les développeurs dans la description des éléments des modèles d'un méta-modèle donné et dans la réalisation de ces modèles. Ces modèles sont des instances des concepts du méta-modèle sur lequel se base la méthodologie.

Certaines méthodologies considèrent soit le processus méthodologique seul soit le méta-modèle seul en omettant l'association qui doit exister entre les deux. Cependant, ces deux composantes sont dépendantes et elles sont étroitement liées. Par exemple, une méthodologie orientée objet ne permet de développer que des systèmes orientés objet. Si une telle méthodologie permet de développer, en plus, des systèmes basés sur d'autres méta-modèles, elle sera considérée comme étant incohérente. Par conséquent, une méthodologie ne sera précise, claire et complète que si son méta-modèle l'est.

Les définitions des concepts du méta-modèle qui seront utilisés dans un processus méthodologique doivent être suffisamment généraux pour englober tous les aspects de la réalité ciblée par le méta-modèle. Ils doivent aussi être suffisamment spécifiques pour ne pas inclure les aspects de la réalité qui ne sont ciblés . Actuellement, les définitions des concepts des SMA utilisées dans les méthodologies sont imprécises et n'ont pas eu le consensus nécessaire de la communauté des SMA. Certaines abstractions sont adoptées dans des méthodologies multi-agent alors qu'elles ne se conforment pas à la réalité de ce modèle.

2.1.2 Confusion des concepts des modèles de systèmes classiques avec ceux des modèles multi-agent

Une des difficultés rencontrées dans la création d'une méthodologie ou dans son utilisation dans le développement des SMA réside dans la distinction des concepts des SMA de ceux des modèles classiques. En plus des difficultés que trouvent les humains à s'adapter à de nouveaux concepts, les raisons de la confusion qui existent entre les concepts classiques et les concepts multi-agent peuvent être cherchées dans :

1. L'évolution des concepts des modèles classiques vers les concepts des modèles multi-agent,

2. L'héritage des concepts des modèles classiques par les modèles multi-agent.

L'évolution des concepts des modèles classiques a fait, le plus souvent, adopter aux modèles classiques les caractéristiques spécifiques des SMA. Par exemple, le concept d'objet demeure un sujet de débat. Pour surpasser les limites des modèles orientés objet, des chercheurs de la communauté des systèmes orientés objet essayent d'équiper les objets de facultés des agents. Par exemple, nous pensons que l'activité est une faculté de l'agent. Il n'est alors pas cohérent de qualifier les objets d'actifs tel qu'on le trouve dans la littérature orientée objet. En conséquence, certaines facultés attribuées aux objets rendent confuses la distinction des notions d'agent de celle d'objet.

L'héritage des concepts des modèles classiques par les modèles multi-agent a fait adopter aux seconds les caractéristiques des premiers. Ceci est la cause des difficultés que trouvent certains chercheurs à se séparer des concepts des modèles classiques dont ils ont utilisés et maîtrisés pour longtemps. De plus, certains essayent de détourner les difficultés qu'ils trouvent dans l'adoption des concepts des SMA. Par exemple, plusieurs travaux utilisent les diagrammes "cas d'utilisation"[1] empruntés des méthodologies orientées objet dans le développement des SMA. L'adoption des concepts des modèles classiques par les modèles multi-agent amène au développement de SMA incohérents. De tels travaux sont qualifiés de multi-agent par leurs auteurs alors qu'ils ne le sont pas réellement.

2.1.3 Definition d'une méthodologie de développement de systèmes multi-agent

Des progrès significatifs ont été réalisés dans les travaux sur les méthodologies multi-agent (voir [Iglesias 98, Wooldridge 01, Lahlouhi 02c]). Ceci est particulièrement vrai pour les méthodologies organisationnelles. Cependant, les aspects de certaines méthodologies sont discutables. La discussion concerne les notions fondamentales des SMA, celles des méthodologies d'une manière générale et celles spécifiques aux méthodologies multi-agent.

L'objectif premier d'une méthodologie multi-agent est de fonder des SMA. Certaines méthodologies sont des modifications des méta-modèles classiques sans tenir compte des attributs de première classe des SMA [Elammari 99]. Ceci est vrai, en particulier, pour des méthodologies qui adoptent les concepts des modèles orientées objet. L'argumentation de tels travaux repose sur la simplification du processus de développement et son acceptation par des développeurs dans une large et longue expérimentation. Tandis que ces deux facteurs sont importants pour une méthodologie, ils sont insuffisants pour qualifier une méthodologie de multi-agent. Une méthodologie qui ne permet pas de développer des systèmes multi-agent ne pourrait être considérée

[1] En anglais "use cases"

comme telle indépendamment de sa simplicité et/ou de sa familiarité. L'ambiguïté sur cette classification est due à l'inexistence d'un consensus sur un méta-modèle multi-agent. D'ailleurs, plusieurs méthodologies sont décrites sans se référer à un méta-modèle multi-agent explicite.

Un des problèmes des méthodologies multi-agent est que la définition du méta-modèle des SMA n'est pas encore complète. Plusieurs méthodologies multi-agent sont basées sur un méta-modèle implicite qui n'est pas généralement admis. Ces méthodologies présentent des processus à suivre pour développer des SMA sans justifier le bien fondé de ces processus. Souvent, ceci créa des confusions et des incompréhensions de la part des développeurs utilisant ces méthodologies. D'autres méthodologies multi-agent repose sur la définition d'agent seule. La définition de la notion d'agent peut être regardée comme un méta-modèle informel. Cependant, une telle définition est insuffisante, à elle seule, de fonder une méthodologie de développement de SMA puisqu'elle ne comporte pas un processus méthodologique, c'est-à-dire, elle n'indique pas comment peut-on identifier les éléments des agents. Les méthodologies basées uniquement sur des aspects du méta-modèle multi-agent sans le processus logiciel ou inversement sont incomplètes. Il leur manque une composante importante de la méthodologie qui est soit le méta-modèle ou le processus méthodologique.

2.2 Problèmes entravant le développement de méthodologies multi-agent

Le développement de méthodologies de développement de systèmes n'est pas une tâche aisée. Le développeur d'une méthodologie de développement de systèmes doit imaginer les besoins des futurs systèmes qui seront développées en suivant sa méthodologie. Il doit ensuite prendre en considération les besoins (pratiques et théoriques) des développeurs de ces systèmes et les associer aux exigences de ces derniers pour faire ressortir une vision unifier répondant aux besoins des développeurs selon les exigences des systèmes. Les SMA regroupe un très large choix de systèmes à développer. Par conséquent, leurs développeurs sont variés. Ils sont de différentes natures et de différents niveaux de compétences. Le développement d'une méthodologie multi-agent pose encore plus difficile. Cette section se concentre alors sur certaines de ces difficultés. Des solutions sont suggérées pour certaines de ces difficultés. D'autres solutions seront données dans le cadre de la méthodologie MASA-Method.

Le reste de cette section examine les concepts de base d'une méthodologie, la notion d'agent, la notion de coopération des agents, les aspects organisationnels et les facultés humaines dans le développement des agents.

2.2.1 Problème du concept de méthodologie

Un premier problème qui oppose les développeurs de méthodologies est la compréhension du concept de méthodologie elle-même. Ceci concerne, en premier lieu, l'inclusion de tous les éléments d'une méthodologie. Ces éléments sont, essentiellement, le méta-modèle et le processus méthodologique qui, tel qu'il a été déjà expliqué précédemment (section 2.1), ces deux éléments sont dépendants. Par conséquent, la définition des seuls modèles de sortie sans le processus méthodologique, comme ça a été fait dans [Omicini 00], ne définit pas une méthodologie. La définition du seul processus méthodologique sans les modèles de sortie, comme celle qui a été faite dans [Lind 99], aussi ne définit pas une méthodologie.

La deuxième difficulté est liée à la confusion qui est faite, généralement, entre les connaissances méthodologiques et les connaissances dépendantes des domaines d'applications. Les premières sont celles des méta-modèles et des processus méthodologiques qui doivent être expliqués d'une manière générale. Les secondes sont des aspects conceptuels et d'implémentation qui pourraient être illustrés dans des exemples.

2.2.2 Problème de la notion d'agent

Actuellement, il y a plus de discussion que de consensus sur les concepts des SMA et nous n'avons pas de méta-modèle admis par tous les membres de la communauté des systèmes multi-agent. Chaque classe de chercheurs de cette communauté présente des points de vue (pouvant être très divergents) qui s'adaptent mieux aux limites et à l'étendue de leurs travaux. Le concept d'agent est alors une source de controverse. Plusieurs caractérisations de l'agent ont été proposées dans la littérature allant d'une simple fonction, passant par une structure de données et des traitements, et arrivant à celle de l'objet puis à celle de l'objet actif.

Dans cette controverse, la première difficulté est celle de la limitation des définitions des concepts des modèles classiques (fonctionnels, basés données et, plus particulièrement, orientés objet). Par exemple, le concept d'objet demeure un sujet de débat. Ceci est plus visible, particulièrement, dans la définition de UML. Dans les modèles orientés objet, les chercheurs essayent d'équiper les objets de facultés d'agents. Ceci se fait, particulièrement, au niveau conceptuel. Par exemple, ce n'est pas juste de qualifier un objet d'actif puisque l'activité est une faculté de l'agent. Cependant, il est important de noter que ceci ne concerne pas les systèmes orientés objet. Ce dernier doit être équipé d'un contrôle qui le maintient actif. Certaines facultés attribuées aux objets rendent flou la limite entre le concept d'agent et celui d'objets. Une long débat est en cours dans la communauté des SMA sur la relation entre le concept d'agent et

celui de l'objet.

La deuxième difficulté se situe dans le fait que certains chercheurs sont contraints (par l'évolution rapide des modèles des systèmes et les besoins d'implémentation) d'emprunter certaines concepts des modèles classiques malgré qu'ils sachent que ces concepts ne sont pas ceux des systèmes multi-agent. Ceci leur permet une rapide satisfaction de leurs besoins. Cependant, une telle pratique a mis certains autres dans la confusion.

La solution à ces différentes difficultés peut être cherchée dans la standardisation des concepts d'agent et multi-agent. Dans ce sens, les efforts des organismes internationaux tels que FIPA et OMG n'ont pas encore atteints ce but en raison de l'ampleur de ce problème.

2.2.3 Problème de l'autonomie des agents dans la coopération

Les agents sont autonomes tandis qu'ils doivent partager certains aspects pour pouvoir coopérer. Ceci pose des difficultés aux développeurs de méthodologies multi-agent. La première de ces difficultés est celle liée au contrôle de l'agent. Chaque agent doit avoir son propre contrôle faisant d'un agent un système complet. Ceci n'est pas le cas des modèles classiques tels que les objets ou les fonctions. Les développeurs de méthodologies multi-agent qui emploient dans leurs méthodologies des diagrammes empruntés aux méthodologies classiques tombent dans un contrôle centralisé sans le savoir. Ceci est le cas, le plus souvent, des modèles orientés objet sans approcher le parallélisme des tâches des agents. Ceci dévie leurs travaux des modèles de SMA.

La deuxième difficulté relative à l'autonomie des agents est celle liée à leur travail collectif. Un travail collectif consiste en un accomplissement d'une tâche commune donnée par un ensemble d'agents. Ce travail ne sera pas réalisé par un contrôle centralisé mais plutôt par une communication entre les accomplissements différentes tâches. Une solution à ce problème peut être fondée sur la distribution de la tâche commune sur des agents de telle sorte que l'accomplissement coordonné des différentes tâches sera équivalent à l'accomplissement de la tâche commune.

La troisième difficulté liée à l'autonomie des agents est celle du partage des ressources. Dans ce cas, on peut adopter l'autonomie des ressources dans laquelle chaque agent possède ses propres ressources qu'il ne partage avec aucun autre agent. Cette solution est possible dans le cas des ressources logicielle bien qu'elle puisse soulever, parfois, certaines difficultés liées à la performance. Cependant, elle n'est pas possible dans des situations plus réalistes, telles que le besoin de l'utilisation collective et simultanée d'un instrument par plusieurs robots. Dans ce cas, les agents deviennent non autonomes du point de vue des ressources. Dans un modèle multi-agent,

il est alors préférable d'avoir l'autonomie des ressources tandis qu'il est nécessaire de supporter leur partage comme c'est le cas dans la réalité humaine.

2.2.4 Problème du retard sur l'adoption des aspects organisationnels

Les auteurs des principales méthodologies multi-agent (telles que Gaia [Wooldridge 00, Zambonelli 00b] et MaSE [DeLoach 00, DeLoach 02b, DeLoach 01a, Sparkman 01, Wood 00, DeLoach 02a]) ont révisé leurs méthodologies pour qu'elles puissent supporter les organisations. Certains auteurs de méthodologies multi-agent n'ont pas compris l'importance des aspects organisationnels dans le développement des SMA. Ceci a retardé le développement de leurs méthodologies.

L'association des SMA et des organisations exige la définition d'un méta-modèle des organisations. Cependant, ceci est un sujet d'une discussion plus large qui est loin d'être résolu. Il n'est pas lié seulement aux SMA mais, plutôt, il inclut également d'autres disciplines telles que la sociologie et l'économie. Les problèmes discutés dans de telles disciplines sont ceux de critères d'évaluation de la performance des organisation et de leurs adéquation à certains environnement. Pour surpasser (plutôt, pour détourner) cette difficulté dans les méthodologies multi-agent, les organisations devraient être considérées comme dépendantes du domaine, c'est-à-dire un problème conceptuel. Une telle considération permet de ne pas prendre en considération tout ses aspects mais plutôt certaines abstractions seulement. Il faut alors proposer des moyens qui permettent de tenir compte des organisations sans être concerné par les critères de sélection d'une organisation particulière. Ce choix sera laissé aux concepteurs des applications.

2.2.5 Problème des facultés humaines dans les systèmes multi-agent

Les humains et les machines physiques (telles que les robots) sont des agents. Les agents de certains systèmes multi-agent sont alors hétérogènes. Ce qui pose le problème de leur intégration. Dans Gaia [Wooldridge 00], l'hétérogénéité est considérée comme étant celle des langages de programmation utilisés pour l'implémentation des agents. Dans Alaaddin [Ferber 98, Ferber 01, Gutknecht 01], l'hétérogénéité a été considérée comme étant l'hétérogénéité des langages d'interaction des agents. Cependant, l'hétérogénéité se situe au niveau du contrôle des agents. Les agents hétérogènes sont des agents avec des contrôles différents. Les méthodologies multi-agent actuelles ne considèrent pas l'intégration d'agents hétérogènes.

Un autre problème est lié au fait que certains chercheurs essayent d'équiper les agents de facultés humaines avancées tels que l'intelligence et certains aspects sociaux avancés. En conséquence, ils ne peuvent pas les mettre en œuvre. De tels travaux

demeurent sans issues pratiques. Si ces travaux sont significatifs dans un certain sens, ceci n'est pas le cas des méthodologies qui sont destinées à être utilisées pour produire des systèmes opérationnels. Il est alors nécessaire de trouver un équilibre entre les facultés humaines et les moyens de réalisation de ces facultés. Pour cela, il faut procéder par raffinement. Tout d'abord, on doit maîtriser les aspects collectifs des SMA (coopération et coordination) pour pouvoir passer, ensuite, avec moins de difficultés, à des niveaux supérieurs.

2.3 Modèles de systèmes multi-agent

Le concept d'agent est hérité de celui de l'humain tout en essayant de tenir compte des facultés de l'humain tels que l'activité, l'intelligence et l'action sur les environnements. Inclure toutes les facultés de l'humain dans un seul modèle est une tâche délicate voir impossible. Plusieurs communautés d'agents ont été alors émergées. Chaque communauté définit une catégorie d'agents (intelligents, logiciels, mobiles, adaptatifs, physiques, etc.) qu'elle étudie en considérant seulement certaines caractéristiques que cette communauté considère comme intéressantes pour ses applications. Plusieurs définitions d'agent ont été alors proposées sans qu'elles soient complètement concordantes. Dans cette section, nous.expliquons les notions de base des agents et des systèmes multi-agent.

2.3.1 Notions de base et caractéristiques des systèmes

Les systèmes multi-agent sont, principalement, des systèmes dynamiques. La notion de système est sujette de controverse. Certains définissent les systèmes de façon descriptive. Les définitions descriptives ne peuvent être poursuivies dans les étapes permettant de produire les aspects opérationnels des systèmes dans un processus de développement. Par exemple, les cybernéticiens définissent un système dynamique (actif) [Ashby 56] comme étant un système dont l'état "s" change dans le temps selon l'équation s(t) = f(s(t-1)) où s(t) (respectivement s(t-1)) représente l'état du système à l'instant t (respectivement t-1). Cette définition ne permet pas, ou ne dit pas comment, développer des systèmes. Elle est d'un très haut niveau d'abstraction. Elle ne considère que l'état du système et ignore les autres aspects. Elle n'est pas d'une grande utilité pour une méthodologie de développement de systèmes qui considère le point de vue constructif.

Dans ce qui suit, nous allons nous concentrer principalement sur les aspects opérationnels des systèmes. Pour cela, on utilise certains aspects empruntés des systèmes à comportement utilisés en robotique pour caractériser les aspects individuels des

Etant donné un ensemble d'effecteurs SE, un système actif AS peut être décrit comme suit :

$$AS = < SE >$$

Figure 2.1: Système actif

Etant donné un ensemble de senseurs SS, un système réactif RS peut être définit comme étant un système actifs disposant de senseurs SS. Il peut alors être décrit comme suit :

$$RS = < AS, SS >$$

Figure 2.2: Système réactif

systèmes (activité, réactivité et autocontrôle). On utilise aussi certains autres aspects empruntés des systèmes distribués pour caractériser les aspects collectifs des systèmes (communication, synchronisation, pro-activité et coopération). Notre objectif est d'aboutir à un modèle de systèmes coopératifs.

Dans le reste de cette sous-section, nous définissons les notions d'activité, de réactivité, d'autonomie de contrôle et d'autonomie de fonctionnement, de communication par acte de langage, de pro-activité et de coopération d'entités autonomes.

Système actif

Un système actif AS (Figure 2.1-Page 16) est un système qui modifie son environnement en utilisant un ensemble d'effecteurs. Le suivi des changements de l'état de l'environnement fait apparaître l'activité du système. La nature des effecteurs dépend de la nature du système et de celle de l'environnement. Par exemple :

1. Les roues d'un robot mobile sont des effecteurs,
2. Un écran et un dispositif d'enregistrement sont des effecteurs d'un système informatique.

Système réactif

Un système réactif RS (Figure 2.2-Page 16) est un système qui modifie son environnement en réponse aux changements de ce même environnement. Il doit alors être un

Etant donnés un contrôle (C) et un fonctionnement (F),
un système automatique QS peut être définit comme
un système réactif disposant d'un contrôle C
et d'un fonctionnement F.
Il peut alors être décrit comme suit :

$$QS = < C, F, RS >$$

Figure 2.3: Système automatique

système actif qui perçoit les changements de l'environnement en utilisant un ensemble de senseurs. La nature de ces senseurs dépend de la nature du système et de celle de l'environnement. Par exemple :

1. Une caméra est un senseur d'un robot,
2. Un dispositif d'entrée et un dispositif de lecture magnétique sont deux senseurs d'un système informatique.

Système automatique

Un système automatique QS (Figure 2.3-Page 17) est un système actif (ou réactif) dont les effecteurs (et les senseurs s'il est réactif) sont soumis à un certain comportement inclus dans le système. Une large classe de systèmes sont qualifiées de systèmes à comportements ou basés fonctionnement. Ces systèmes incluent les systèmes informatiques et les robots à comportement. Les comportements de ces systèmes sont composés de deux parties :

1. Contrôle tel qu'une unité de commande et de contrôle d'un ordinateur,
2. Fonctionnement tel qu'un programme écrit dans un langage directement interprétable par le contrôle.

Le contrôle commande les senseurs pour percevoir l'environnement et les effecteurs pour agir sur ce même environnement. Les commandes à effectuer sont incluses dans l'expression du fonctionnement. Le contrôle interprète ces commandes pour décider de la manière dont il commandera les senseurs et les effecteurs du système pour percevoir l'environnement et/ou agir sur cet environnement.

Etant donnés un ensemble de senseurs de communication CS, un ensemble d'effecteurs de communication CE et un ensemble de procédures d'un protocole de communication CP. Un système communicatif CmS peut être définit comme d'effecteurs de communication CE et d'un protocole de communication CP. un système actif AS disposant de senseurs de communication CS. Il peut alors être décrit comme suit :

$$CmS = < AS, CS, CE, CP >$$

Figure 2.4: Système communicatif

Système communicatif

Un système communicatif CmS (Figure 2.4-Page 18) est un système réactif qui utilise certains de ses senseurs pour recevoir des messages d'autres systèmes. Il peut utiliser aussi certains de ses effecteurs pour soumettre des messages à d'autres systèmes. L'émission et la réception de messages se font selon un protocole de communication préétabli. Un protocole de communication est un ensemble de procédures précisant comment un système S1 agit sur l'environnement pour qu'un autre système différent S2 comprennent que cette modification lui indique quelque chose précis. Les protocoles font partie du fonctionnement du système communicatif.

Un système communicatif classique CCmS est un simple serveur selon un fonctionnement et un protocole de communication préétablis. Un tel protocole constitue ce qui est communément appelé son "interface utilisateur". Son fonctionnement est réduit au savoir-faire. Ce savoir-faire est un ensemble de procédures décrivant la façon dont le système peut réaliser une tâche donnée en utilisant ses senseurs et ses effecteurs. L'utilisateur d'un système classique est un agent qui est, généralement, un humain. L'utilisateur invoque les procédures du savoir-faire selon un plan, pouvant être imprécis, que cet utilisateur détient et que le CCmS ne connaît. Le rôle du contrôle d'un CCmS est limité à :

1. L'interprétation du protocole de communication et des procédures du savoir-faire,

2. La commande des senseurs et des effecteurs selon les demandes de l'utilisateur qui sont externes au CCmS.

Système proactif

L'autonomie du système peut être vue de plusieurs points de vue : énergie, ressources, prise de décision, etc. Du point de vue contrôle, un système automatique classique CCmS peut être considéré comme étant autonome. Sans aucune intervention externe, il commande ses senseurs et ses effecteurs et interprète son savoir-faire et ses protocoles de communication. Cependant, un système CCmS n'est pas fonctionnellement autonome puisqu'il dépend d'un utilisateur qui lui indique ce qu'il doit faire. Un système CCmS n'a pas son propre objectif. L'objectif implicite d'un système classique est celui de servir son utilisateur. Cet objectif peut être décrit comme suit :

1. Attendre à ce que l'utilisateur formule une demande,
2. Répondre à la demande de l'utilisateur,
3. Aller a 1.

Un système fonctionnellement autonome est dit proactif PS. Un système PS peut prendre l'initiative d'agir sur l'environnement. Un tel système peut être obtenu en dotant un système classique d'un objectif permettant à son contrôle de prendre des décisions sur ce qu'il doit faire. Un système PS est alors dirigé par son propre objectif et non pas par les interventions de systèmes externes. Ceci signifie qu'un système PS rejette les demandes externes qui ne sont pas compatibles avec son objectif.

Pour communiquer avec un système classique, les demandes des systèmes externes indiquent seulement la tâche qu'ils demandent au système classique de réaliser. Cependant, dans le cas d'un système proactif PS (Figure 2.5-Page 20), ces demandes doivent être accompagnées d'éléments autres que l'action demandée. Ces éléments permettent au système PS de prendre les décisions sur les demandes des systèmes externes. Ces éléments inclus un performative clarifiant la nature de la demande (souhait, ordre, etc.). Ceci implique que les protocoles de communication avec des systèmes proactifs doivent être basés sur les actes de langage. En conséquence, le fonctionnement d'un système proactif se compose de deux parties : Un savoir-faire et un objectif du système accompagné de protocoles d'actes de langage.

Système coopératif

Deux systèmes sont dits non communicants si chacun d'eux ne tient pas compte, d'une manière directe, des modifications que l'autre fait sur l'environnement. Deux systèmes communicants peuvent communiquer de façon directe ou indirecte Dans une communication directe, les systèmes communicants savent qu'ils communiquent en utilisant

Etant donnés un système communicatif CS (SP)
utilisant un protocole de communication basé sur
les protocoles d'actes de langage SP, et un objectif O.
Un système proactif PS peut être définit comme
un système CS (SP) ayant un objectif O.
Il peut alors être décrit comme suit :

$$PS = < CmS\ (SP),\ O >$$

Figure 2.5: Système proactif

Etant donnés un protocole de synchronisation Z
et un système proactif PS. Un système coopératif
CpS peut être définit comme un système PS
ayant des protocoles de synchronisation Z.
Il peut alors être décrit comme suit :

$$CpS = < PS,\ Z >$$

Figure 2.6: Système coopératif

un protocole de communication préétablit entre eux. Dans une communication indirecte, chacun de ces deux systèmes réagit aux changements de l'environnement fait par l'autre sans utilisation de protocole de communication.

Un système coopère avec d'autres systèmes de son environnement d'une manière directe ou indirecte. La coopération est dite indirecte si le système agit sur l'environnement sans synchroniser ses actions avec celles des systèmes avec lesquels il coopère. Sinon la coopération sera dite directe. La synchronisation utilise la communication. Pour synchroniser ses actions avec celles des autres systèmes, le système suit un protocole de synchronisation Z.

Un système coopératif CpS (Figure 2.6-Page 20) est dit système à capacités sociales. Nous avons détaillé la description d'un système coopératif CpS dans la Figure 2.7-Page 21.

2.3.2 Concept d'agent

Un agent est un système ayant des propriétés spécifiques. Les propriétés d'un système agent, selon [Wooldridge 97], sont l'autonomie, la réactivité, la pro-activité et les capacités sociales. C'est alors un système coopératif CpS tel qu'il est décrit dans la

Etant donnés:
C = Contrôle
K = Savoir-faire
F = Fonctionnement incluant K, O, Z, SP
CS = {e / e est un senseur de communication}
CE = {e / e est un effecteur de communication}
S = {e / e est un senseur} incluant CS
E = {e / e est un effecteur} incluant CE
Z = Protocole de synchronisation,
O = Objective
SP = {cp / cp est une procédure du protocole
communication basé sur les actes du langage}
Un système coopératif CpS peut être décrit comme suit :

CpS = < C, F (K, SP, O, Z), S (SS, CS), E (SE, CE) >

Figure 2.7: Système coopératif détaillé

sous-section 2.2.2. Il peut être présenté comme suit :

Agent = (C, F <K, SP, O, Z>, S <SS, CS>, E <SE, CE>)

Il inclut alors :

1. Contrôle : Sa nature peut être procédurale, déclarative ou intelligente. Le contrôle interprète la tâche de l'agent, ses protocoles de communication, ses protocoles de synchronisation et ses procédures du savoir-faire. De plus, il commande ses senseurs et ses effecteurs. Enfin, il maintient l'agent actif et autocontrôle,

2. Savoir-faire : Il est décrit selon des perceptions et des actions sur l'environnement au moyen des fonctionnalités des senseurs et des effecteurs usuels de l'agent,

3. Tâche (objectif) : Elle est décrite en fonction des procédures du savoir-faire de l'agent. C'est une combinaison des procédures du savoir-faire dans un plan permettant d'accomplir une mission donnée,

4. Etat : Il inclut un modèle de l'environnement de l'agent. Il inclut aussi l'état d'évolution de la progression vers l'objectif de l'agent,

5. Protocoles de communication : Ils doivent être basés sur les actes de langage [Searle 69]. Leurs descriptions sont fonctions de perceptions et d'actions sur l'environnement en utilisant les senseurs et les effecteurs de communication de l'agent. Ces protocoles peuvent être considérés comme des procédures particulières du savoir-faire de l'agent,

Etant donnés un ensemble d'agents SA et une relation
de communication CR. Un système multi-agent MAS
peut être décrit comme suit :
MAS = < SA, CR > Où :

1. SA = {A / A est un agent}

2. CR = {(CE-A1, CS-A2) / (A1, A2) ∈ SA X SA.
CE-A1 est un effecteur de communication de l'agent A1
et CS-A2 est un senseur de communication de l'agent A2}

Figure 2.8: Système multi-agent

6. Protocoles de synchronisation : Leurs descriptions sont fonctions des communications. Ces protocoles permettent à l'agent de coordonner ses actions avec celles des autres agents de son environnement pour éviter des conflits éventuels,

7. Effecteurs usuels : Ils permettent à l'agent d'agir sur son environnement,

8. Senseurs usuels : Ils permettent à l'agent de percevoir son environnement,

9. Effecteurs de communication : Ils permettent à l'agent d'envoyer des messages sur son environnement à destination d'autres agents,

10. Senseurs de communication : Ils permettent à l'agent de recevoir des messages de son environnement provenant des autres agents.

2.3.3 Concept de système multi-agent

Un SMA est un système d'agents (Figure 2.8-Page 22). Ces agents coopèrent pour accomplir une tâche commune. Un SMA a alors les aspects fondamentaux d'un système. Cependant, les agents sont des systèmes autonomes et proactifs. Chaque agent dispose de son propre contrôle ainsi que de son propre fonctionnement. La seule relation directe qui peut exister entre deux agents d'un SMA est celle de communication. Les aspects d'un SMA (en particulier, son contrôle et son fonctionnement) sont alors distribués sur ses agents. En outre, ses agents ne peuvent pas partager la description de la tâche commune à accomplir. Il est alors nécessaire de distribuer cette tâche sur les agents de telle sorte que chaque agent aura une tâche indépendante de telle sorte que l'accomplissement des tâches de tous les agents soit équivalent à l'accomplissement de la tâche commune qui est celle du SMA.

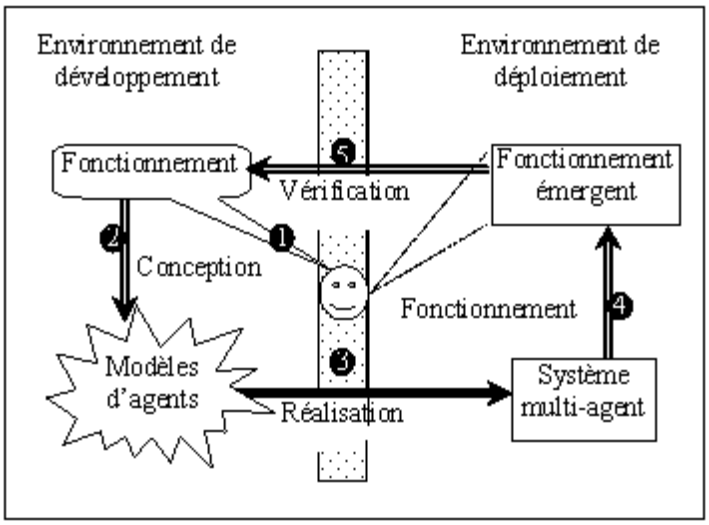

Figure 2.9: Processus méthodologique basé agent

2.4 Processus méthodologiques de développement de systèmes multi-agent

Les chercheurs dans le domaine des SMA ont proposé différentes approches pour la modélisation des systèmes. Aucun d'eux ne peut raisonnablement prétendre être universel. Nous pouvons classer ces approches en deux catégories. La première catégorie est celle des approches de développement de SMA basé agent. La deuxième est celle des approches de développement basées rôles. Cette section tente d'expliquer les deux catégories d'approches tout en se focalisant sur les approches basées rôle.

2.4.1 Développement basé agent

Le processus basé agent est dirigé par l'imagination des fonctionnements des agents. Il peut être décrit comme suit (Figure 2.9-Page 23) :

1. Développer des agents,
2. De façon ascendante, vérifier que le fonctionnement collectif des agents est compatible avec le but du développeur.

Dans un processus basé agent, le développeur doit imaginer le fonctionnement direct des agents. La conception produit des modèles d'agents du SMA qu'il est nécessaire de coordonner, ensuite, dans le SMA final. La vérification du fonctionnement du SMA est faite en comparant le fonctionnement émergent avec celui imaginé. Cette

vérification est une tâche difficile. Elle doit se faire au niveau de l'expression mentale du fonctionnement.

En principe, la compréhension du processus méthodologique basé agent est facile. Le développement des agents et la vérification de la satisfaction de leur fonctionnement collectif se font, cependant, empiriquement. Pour simplifier l'imagination du fonctionnement émergent des agents et leur intégration, les processus basés agent emploient un modèle unifié des agents. Généralement, ces processus inspirent les modèles d'agents de la nature telle que les fourmis. L'unification des modèles d'agents a plusieurs inconvénients. En particulier, cette unification ne permet pas d'avoir des agents réellement hétérogènes dans un même SMA.

2.4.2 Développement basé rôle

Les développeurs humains sont continuellement immergés dans une variété d'arrangements organisationnels. La métaphore organisationnelle est alors une métaphore naturelle pour ces développeurs. Cette métaphore ouvre la possibilité de réutiliser une variété d'études et d'expériences liées aux organisations du monde réel [Handy 76, Mintzberg 79]. De plus, la métaphore organisationnelle semble être appropriée pour des systèmes logiciels à large portée.

Dans un développement basé rôle, un SMA est considéré comme une réalisation d'une organisation informatique. Cette organisation constitue l'objectif du SMA. De façon abstraite, une organisation est constituée d'un ensemble de rôles qui doivent être assumés par des agents. Chaque agent assumera un ou plusieurs rôles. La tâche d'un agent sera alors d'assumer ses rôles. La distribution (attribution) des rôles sur les différents agents qui mettront en œuvre l'organisation aboutit à un SMA. Le rôle d'un agent définit ce qu'on s'attend à ce que cet agent fasse dans l'organisation, en concertation avec les autres agents et avec l'organisation elle-même [Zambonelli 03]. Les modèles de rôles décrivent précisément tous les rôles qui constituent l'organisation informatique. Cette description est faite en termes de leurs tâches aussi bien qu'en termes de leurs protocoles de communication.

Le développement basé rôle commence par l'adoption d'une organisation donnée pour développer, ensuite, des agents selon les rôles qu'ils assumeront. Ce développement se fait selon le processus méthodologique (Figure 2.10-Page 25) suivant :

1. Décrire les modèles des rôles de l'organisation,

2. Attribuer les rôles de l'organisation aux agents et décrire les modèles des agents à développer,

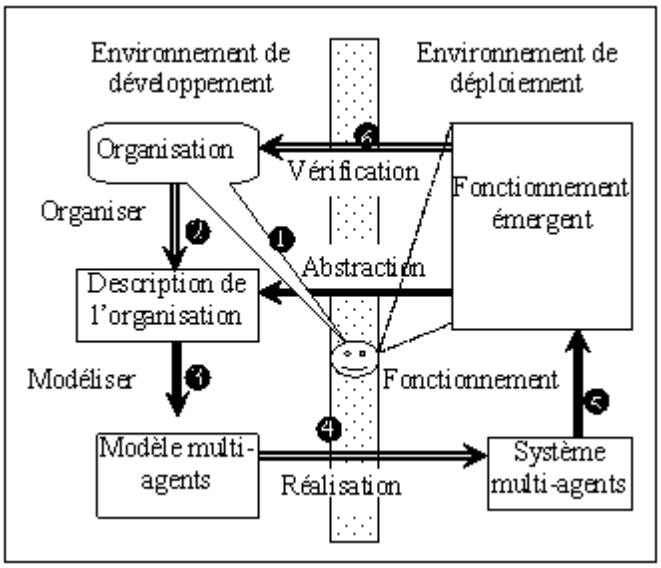

Figure 2.10: Processus méthodologique basé rôle

3. Vérifier que le fonctionnement collectif des agents est compatible avec l'organisation.

Le processus basé rôle est préférable que le processus basé agent puisque le développeur imagine une organisation des agents. Cependant, il n'est pas dirigé par l'objectif du système. Par conséquent, la modélisation permet de produire des modèles multi-agent sans pouvoir les justifier. La vérification se fait en comparant le fonctionnement émergent avec l'organisation imaginée. Cette vérification est plus facile comparé à celle du processus basé agent. La vérification peut être faite à deux niveaux : Celui de la description de l'organisation et celui de l'expression mentale. Cependant, elle reste encore difficile.

D'un coté, le problème du développement basé agent est rendu simple dans le développement organisationnel. Au lieu de penser en terme d'"agents", on raisonne plutôt en terme de rôles. Les rôles sont plus abstraits et, par conséquent, plus simples à manipuler. En particulier, on utilise des organisations bien connues tels que les organisations économiques. D'un autre coté, le problème du développement des agents des méthodologies basées agent est décalé aux rôles puisqu'il n'est toujours pas clair comment choisir les éléments d'une organisation (rôles et relations entre eux, par exemples).

2.5 Classification des méthodologies multi-agent

Dans [Iglesias 98], Iglesias et al. ont classés les méthodologies en celles étendant les méthodologies orientées objet et celles étendant les méthodologies d'ingénierie des connaissances. Cette classification bien connue et bien référencée dans la littérature est basée sur l'implémentation des SMA. Une telle implémentation est faite, généralement, soit sous forme de systèmes experts soit sous forme d'un système orienté objet. Avec l'évolution des méthodologies multi-agent, cette classification est devenue de moins en moins représentative. Plusieurs méthodologies multi-agent, telles que les méthodologies organisationnelles, ne peuvent être distinguées des autres selon cette classification.

Cette section décrit la classification proposée dans [Lahlouhi 02c]. Cette classification est basée sur un processus de développement de SMA comprenant d'autres étapes que celle d'implémentation des SMA. Ce processus est essentiellement organisationnel. Cependant, la classification inclut les méthodologies basées sur l'émergence en tant que méthodologies de modélisation. Le reste de cette section explique la classification proposée et détaille les processus des méthodologies organisationnel et complètes.

2.5.1 Critères de classification des méthodologies multi-agent

Les systèmes multi-agent sont basés sur deux aspects : individuels et collectifs. Les premiers sont ceux des systèmes classiques [Gutknecht 01, Wooldridge 00]. Les seconds constituent la nouvelle dimension introduite par les systèmes multi-agent. Se sont des aspects organisationnels. En conséquence, un processus méthodologique de développement de SMA peut être réduit aux trois phases suivantes :

1. Phase d'organisation : Description d'une organisation en partant des besoins du système,
2. Phase de modélisation : Dérivation d'un modèle multi-agent à partir de l'organisation,
3. Phase d'implémentation : Implémentation du modèle multi-agent.

Ce processus devient de plus en plus admis surtout dans des méthodologies organisationnelles telles que MaSE [DeLoach 02a, Sparkman 01], Gaia [Wooldridge 00, Zambonelli 00b, Zambonelli 03] et Alaaddin [Ferber 98, Ferber 01, Gutknecht 01].

Les méthodologies multi-agent peuvent être alors classées selon ce qu'elles considèrent du processus méthodologique ci-dessus (voir la Table 2.1-Page 28). Notons qu'une méthodologie de développement de SMA doit inclure obligatoirement l'étape de modélisation puisque c'est cette étape qui la distingue des méthodologies des autres modèles. Cette classification mène aux méthodologies qui sont :

1. D'implémentation (sous-section 2.5.2) : Cette catégorie considère un modèle de SMA bien déterminé et se focalise sur l'implémentation de ce modèle. Elle ne comporte alors que la phase d'implémentation,
2. De modélisation (sous-section 2.5.3) : Cette catégorie ne comprend que la phase de modélisation,
3. Organisationnelles (sous-section 2.5.4) : Cette catégorie comprend les deux phases d'organisation et de modélisation,
4. Complète (sous-section 2.5.5) : Cette catégorie comporte les trois phases.

Le reste de cette section décrit ces différentes classes.

2.5.2 Méthodologies d'implémentation

Ces méthodologies sont celles qui permettent l'implémentation des modèles de SMA sans se soucier du développement des modèles multi-agent. Elles sont intéressantes pour l'implémentation des modèles multi-agent, généralement, sous forme de systèmes orientés objet. Cette classe inclut les travaux sur l'adaptation du modèle orienté objet pour supporter des agents. Les travaux les plus représentatifs de cette classe sont :

1. Ceux de Odell et de Parunak sur l'extension de UML [Odell 00, Parunak 01],
2. MESSAGE/UML [Caire 01] et MESSAGE [Sanz 02].

2.5.3 Méthodologies de modélisation

Ces méthodologies sont celles qui se sont intéressés uniquement à la modélisation des SMA. Elles considèrent les aspects relatifs à la dérivation des modèles multi-agent à partir des besoins du système sans se soucier de leur implémentation. Plusieurs travaux peuvent être placés dans cette classe tels que :

1. MAS-CommonCADS [Iglesias 96],
2. ODAC [Gervais 00, Gervais 01],
3. Versions antérieures de MaSE [DeLoach 00, DeLoach 02b, DeLoach 01a, Sparkman 01, Wood 00],
4. Versions antérieures de Tropos [Bresciani 01, Giunchiglia 02],
5. [Elammari 99],

Table 2.1: Classification des méthodologies multi-agent

Classe	Sous-classe	Examples de méthodologies
Implémentation		Extension de UML [Odell 00, Parunak 01], MESSAGE/UML [Caire 01], MESSAGE [Sanz 02]
Modélisation		Tropos [Bresciani 01, Giunchiglia 02], ODAC [Gervais 01, Gervais 00], [Elammari 99], MaSE [DeLoach 01a, DeLoach 02b], [DeLoach 00, Sparkman 01, Wood 00], Zeus [Nwana 99], Prometheus [Padgham 02], SODA [Omicini 00], ADELFE [Bernon 02a, Bernon 02b, Bernon 01], MAS-CommonCADS [Iglesias 96]
Organisation-nelles	Ascendantes	Cassiopeia [Drogoul 00, Collinot 96b], [Collinot 96a], Tropos [Giunchiglia 02]
	Descendantes	Alaaddin [Ferber 98, Ferber 01, Gutknecht 01], Styx [Bush 01], MaSE [DeLoach 02a, DeLoach 01a], [DeLoach 02b, DeLoach 00, Wood 00], Gaia [Wooldridge 00] Révision de Gaia [Zambonelli 03]
	Mixtes	Aucune
Complètes	Ascendantes	Aucune
	Descendantes	Extension de MaSE [DeLoach 02a]
	Mixtes	Aucune

6. Prometheus [Padgham 02],
7. ADELFE [Bernon 02a, Bernon 02b],
8. SODA [Omicini 00],
9. Zeus [Nwana 99].

2.5.4 Méthodologies organisationnelles

Ces méthodologies sont celles qui ses ont intéressées au développement organisationnel des SMA. Elles permettent de développer des modèles multi-agent à partir d'organisations. Les méthodologies de cette classe peuvent être subdivisées en deux sous-classes, selon l'ordre dans lequel elles considèrent le développement des organisations: Méthodologies ascendantes et méthodologies descendantes.

Les méthodologies ascendantes commencent par le développement des agents, ou réutilisent des agents déjà développés, puis attribuent ces agents aux rôles de l'organisation. Cette sous-classe contient des méthodologies telles que :

1. [Bush 01],
2. Cassiopeia [Collinot 96b, Collinot 96a, Drogoul 00],
3. Tropos [Bresciani 01, Giunchiglia 02].

Les méthodologies descendantes commencent par l'identification des aspects d'organisation (rôles, en particulier) vient, ensuite, le développement des agents qui peuvent les accomplir. Cette sous-classe contient des méthodologies telles que :

1. ALAADDIN [Ferber 98, Ferber 01, Gutknecht 01],
2. [Müller 00],
3. Gaia [Wooldridge 00] et son extension [Zambonelli 00b],
4. Styx [Bush 01].

2.5.5 Méthodologies complètes

Ces méthodologies sont celles dont le processus méthodologique implique, au moins, toutes les phases du processus organisationnel minimal précédent. Puisque le processus méthodologique de ces méthodologies inclut la phase organisationnelle, elles sont également organisationnelles. Par conséquent, elles peuvent également être classées, du point de vue organisationnel, en méthodologies ascendantes et descendantes. Les méthodologies complètes sont très rares. Comme exemple de méthodologies complètes descendante, on peut citer l'extension de MaSE [DeLoach 02a].

2.6 Insuffisances communes aux méthodologies multi-agent organisationnelles actuelles

L'inconvénient commun à toutes les méthodologies non organisationnelles est qu'elles ne tiennent pas compte des aspects organisationnels dans le développement des SMA tandis que ces aspects sont fondamentaux. Malgré l'importance de ces méthodologies, elles peuvent alors être considérées comme non satisfaisantes. L'importance des aspects organisationnels est montrée dans la tendance actuelle des chercheurs des méthodologies multi-agent qui essaient d'intégrer les aspects organisationnels dans

leurs méthodologies. Cependant, pour le moment peu de méthodologies peuvent être qualifiées d'organisationnelles.

Cette section se concentre sur certaines limites des méthodologies organisationnelles. Ces limites concernent :

1. La non considération de la coopération au niveau organisationnel,
2. La non considération des agents hétérogènes,
3. L'incomplétude de certaines méthodologies.

2.6.1 Non considération de la coopération au niveau organisationnel

Les méthodologies multi-agent organisationnelles actuelles ne tiennent pas compte de la coopération au niveau des organisations mais plutôt au niveau du modèle multi-agent. Ceci concerne, au moins, les méthodologies Cassiopeia [Drogoul 00, Collinot 96b, Collinot 96a], [Elammari 99], Tropos [Giunchiglia 02], Alaaddin [Ferber 98, Ferber 01, Gutknecht 01], MaSE [DeLoach 01a, DeLoach 02b, DeLoach 00, Wood 00], Gaia [Wooldridge 00] et sa révision [Zambonelli 00b], et Styx [Bush 01]. La non prise en-charge de la coopération au niveau organisationnel peut être considéré comme étant une incohérence méthodologique [Lahlouhi 02c]. La coopération est un aspect collectif qui doit, par conséquent, être considéré au niveau de l'organisation qui incorpore de tels aspects.

Conceptuellement, la non considération de la coopération au niveau organisationnel a, au moins, trois inconvénients :

1. Il est plus difficile de justifier la structure adoptée pour l'organisation,
2. Sans savoir l'objectif global que les agents doivent travailler atteindre collectivement, il est difficile d'élaborer les divers objectifs des agents qui assumeront les rôles de l'organisation,
3. La coordination des activités des agents ne serait pas claire sans savoir l'objectif de leurs activités. Dans l'extension de Gaia [Zambonelli 00b], une étape est anticipée dans la direction de l'adoption de la coordination au niveau de l'organisation. La coordination est un aspect fondamental utilisé dans le développement de la coopération. Cependant, dans Gaia cette coordination est considérée indépendamment de la coopération.

2.6.2 Considération non uniforme des agents hétérogènes

Un autre aspect significatif est celui de l'intégration des agents humains et physiques avec les agents logiciel. Jusqu'ici, les travaux traitant ce problème regardent les humains comme utilisateurs des agents. Cependant, les agents sont autonomes et les humains ne peuvent pas les utiliser comme ils utilisent les objets. Les humains peuvent seulement coopérer avec eux. La coopération des humains avec les autres agents est une étape pour tenir compte de la coopération dans l'organisation selon un protocole qui peut comporter, évidemment, des ordres, des demandes, etc. L'approche basée coopération [Lahlouhi 02b] pour des interfaces entre les agents humains et les agents logiciels est une solution à ce problème.

2.6.3 Processus méthodologiques incomplets

Les processus méthodologiques de la majorité des méthodologies multi-agent actuelles n'incluent pas toutes les étapes du processus minimal de développement des systèmes multi-agent expliqué dans la section 2.2. La Table 2.1-Page 28 montre aussi que, parmi les nombreuses méthodologies étudiées, la seule méthodologie complète est MaSE [DeLoach 02a].

2.7 Insuffisances spécifiques à certaines méthodologies organisationnelles

Cette section considère certains aspects particuliers des méthodologies multi-agent organisationnelles considérées comme principales. Cette considération est due au fait qu'elles sont plus documentées, plus citée et dont la documentation est disponible. Ces méthodologies sont ALAADDIN, Gaia et MaSE. Dans la Table 2.2-Page 33, nous résumons cette analyse.

2.7.1 La méthodologie du projet ALAADDIN

La méthodologie du projet ALAADDIN [Ferber 98, Ferber 01, Gutknecht 01] se compose d'un méta-modèle [Ferber 98, Ferber 01] et d'un processus méthodologique. Elle est conçu dans le but de compléter des méthodologies classiques, c'est-à-dire, celles qui se concentrent sur les aspects individuels d'un agent. Le modèle d'organisations est basé sur les concepts d'agent, de groupe et de rôle. La structure de groupe est un ensemble de rôles et d'interactions entre eux offrant un contexte commun et raisonnable [Ferber 01]. Elle est également définie comme étant un ensemble atomique

de l'agrégation des agents et puis comme un ensemble de rôles, du graphe d'interaction et du langage d'interaction [Ferber 01].

La structure d'organisation est un ensemble de structures de groupes [Ferber 01]. Les auteurs de la méthodologie Alaaddin n'indiquent pas comment ils ont employé les méthodologies classiques pour compléter leur méthodologie. En outre, le processus méthodologique de Alaaddin inclut le choix de l'architecture de l'agent et non pas l'utilisation d'une méthodologie classique. A notre avis, les auteurs de Alaaddin auraient mieux fait s'ils ont décrit des règles montrant comment pouvait-on dériver des aspects d'agents et des systèmes multi-agent de l'organisation, montrer comment peut-on employer des méthodologies classiques, en général, et comment peut-on utiliser une méthodologie classique particulière sur un exemple d'application.

Dans Alaaddin, la notion de représentant associe des rôles de divers groupes. Ceci reflète l'absence du critère de groupement dans les groupes de ALAADDIN.

Enfin, la méthodologie de ALAADDIN ne tient pas compte de la coordination au niveau organisationnel.

2.7.2 La méthodologie Gaia et sa révision

Gaia [Wooldridge 00] est une méthodologie d'analyse et de conception de SMA. Elle est basée sur des définitions de différents concepts et les processus méthodologiques qui permettent de les établir. Gaia est employé par plusieurs chercheurs comme une référence pour la comparaison des méthodologies et comme base de développement de leurs propres méthodologies en l'imitant, en l'enrichissant et/ou en l'améliorant. Elle est alors la plus référencée des méthodologies organisationnelles. Cependant, Gaia ne décrit pas explicitement les modèles qu'elle utilise. Ceci ne permet pas d'avoir des spécifications précises à implémenter.

Les résultats du processus méthodologique de Gaia sont destinés d'être des spécifications qui seront implémentées en utilisant une méthodologie classique, telles qu'une méthodologie orientée objet. Cette idée liée à celle de la définition de l'agent comme système est très importante. Cependant, Gaia n'indique pas comment peut-on utiliser les méthodologies classiques pour la compléter. Sa révision [Zambonelli 00b] considère la modélisation de l'environnement. Cette révision a clarifié le besoin de considérer la coordination au niveau organisationnel. Cette considération est une étape méthodologique significative comparée aux autres méthodologies organisationnelles.

Il est significatif de noter que la fondation d'une organisation est un problème complètement conceptuel. La révision de Gaia [Zambonelli 00b] sépare les aspects d'identification des rôles, des protocoles et des règles de l'organisation de la structure organisationnelle. Nous pensons qu'une telle séparation n'est pas judicieuse. Les as-

pects précédents dont la structure de l'organisation dépend peuvent eux aussi dépendre de la structure organisationnelle elle-même. La structure à adopter pour l'organisation peut alors influencer les rôles. Si cette identification est faite au niveau d'analyse, il est nécessaire d'adapter la structure aux rôles. Celle-ci peut être une incohérence méthodologique.

Table 2.2: Table récapitulatif de l'analyse des méthodologies multi-agent organisationnelles Alaaddin, Gaia et MaSE

Aspect méthodologique	Méthodologies		
	Alaaddin	Gaia	MaSE
Processus	Organisationnnel ascendant	Organisationnel ascendant	Complet ascendant
Structures organizationnelle	Agent, groupe et rôle	Basées rôle	Basées rôle
Consideration de la coordination	Non considérée	Niveau organisationnel	Niveau conceptuel
Consideration de la cooperation	Niveau modèle multi-agent	Niveau modèle multi-agent	Niveau modèle multi-agent
Diagrammes	Empirique	Empirique	Hierarchie And/or, diagrammes UML (Relation n'est pas claire entre eux)
Agents hétérogènes	Non considérés	Non considérés	Empirique
Observations	La première incluant les aspects organisationnels	La plus publiée et référencée	Méthodologie complète (Appliquée aux robots et aux agents mobiles)

2.7.3 La méthodologie MaSE et son extension

MaSE, avec son extension, est une méthodologie organisationnelle complète descendante selon la classification présentée dans la section 2.2. L'extension de MaSE [DeLoach 02a] pour supporter les organisations est composée de méta-modèles et d'un processus méthodologique global détaillé.

Le grand reproche qu'on peut faire à la méthodologie MaSE est qu'elle est trop orientée objet. Au niveau conceptuel, elle emploie les principaux diagrammes du méta-modèle UML. Ses auteurs notent cet effet clairement en l'adoptant comme étant l'un des objectifs de MaSE. L'introduction de la description des objectifs du système dans l'étape d'analyse [DeLoach 02a, DeLoach 01a, DeLoach 02b] permet de tenir compte de la coopération d'une manière plus cohérente telle qu'elle est exprimée dans les

besoins du système. Cependant, il serait meilleur si elle était faite au niveau de l'organisation. Un autre problème est lié au fait que la relation entre cette description et les descriptions des autres parties du système (aux étapes suivant l'étape d'analyse) n'est pas claire. C'est le cas, en particulier, de leur relation avec les diagrammes des autres étapes et les modèles des agents. En outre, cette description est une hiérarchie "et/ou" qui signifie que la réalisation d'un objectif est faite par la réalisation de tous ses sous objectifs (relation "et") ou, au moins, de l'un d'entre eux (relation "ou"). Si on fait les substitutions suivantes:

- L'expression "sous objectif" sera remplacée par l'expression « accomplir le sous objectif »,
- L'opérateur "et" par la séquence et/ou le parallélisme entre les accomplissements de divers sous objectifs,
- L'opérateur "ou" par la concurrence.

Il serait alors plus facile de déduire que les diagrammes des objectifs de MaSE ne permettent pas l'expression de la synchronisation entre les réalisations des sous objectifs. La synchronisation sera alors ajoutée dans les étapes suivantes sans tenir compte de la réalité du problème à résoudre. Comme ça a été réclamé par Zambonelli dans [Zambonelli 00b], il est nécessaire de tenir compte de la coordination au niveau organisationnel.

2.8 Conclusion : Vers une méthodologie complète, mixte et dirigée par objectif

La création d'une méthodologie de développement de systèmes ne sera complet que si son processus méthodologique peut satisfaire tous ses utilisateurs potentiels dans le développement aisé de n'importe quel système. Cette aisance ne peut être complète que si un tel processus est entièrement automatique. Un tel objectif est loin d'être atteint à l'état actuel des méthodologies. La maturité du développement de systèmes selon un méta-modèle donné fait alors passer à un autre méta-modèle et une nouvelle catégorie de méthodologies. C'est ce qui à été constaté avec la programmation structurée, les méthodologies de développement de bases de données et les méthodologies orientées objet.

En ce qui concerne les méthodologies de développement de SMA, celles qui ont fait leur preuve et continuent à être développées sont à caractère organisationnel. Les aspects organisationnels sont importants dans un développement méthodologique des

SMA. Il est alors important de doter les méthodologies par des aspects organisationnels. On peut doter une méthodologie d'un développement organisationnel directement au cours de sa création tel que ça a été fait pour ALAADDIN et Gaia. On peut le faire aussi en étendant une méthodologie existante tel que ça a été fait pour MaSE. Pour cela, il faut adopter une approche basée sur l'ingénierie des organisations.

L'ingénierie des organisations doit inclure, au moins, les quatre aspects suivants :

1. Un méta-modèle organisationnel incluant les aspects organisationnels des SMA,
2. Un méta-modèle multi-agent,
3. Un processus méthodologique de développement des organisations,
4. Des règles systématiques de dérivation des modèles de SMA à partir des modèles d'organisations.

De tels aspects ne sont pas considérés de cette façon dans les méthodologies actuelles.

Les agents sont des systèmes autonomes. Leur développement peut être différent d'un modèle de systèmes à un autre. Le développement d'un agent particulier peut alors suivre une méthodologie donnée de développement de systèmes classiques. La difficulté d'utiliser les méthodologies classiques dans le développement d'agents réside dans le fait qu'elles considèrent le contrôle comme étant implicite. Ceci n'est pas le cas des systèmes agents. Les agents doivent être équipés de contrôles pouvant être différents, c'est-à-dire, ces agents peuvent être hétérogènes. Une méthodologie basée sur l'unification des modèles ne peut alors prendre en considération les SMA à agents hétérogènes. Une méthodologie multi-agent doit considérer plutôt l'intégration des agents hétérogènes.

Comme expliqué dans la section 2.5, une méthodologie descendante ne peut pas prendre en considération la réutilisation des agents existants, et une méthodologie ascendante ne peut prendre en considération le développement de nouveaux agents. Il est alors important q'une méthodologie multi-agent considère les deux sens à la fois. Une telle méthodologie sera qualifiée de "mixte".

Le développement d'une organisation est dirigé par un objectif. Dans un développement basé rôle, un tel objectif se trouve dans l'esprit du développeur, i.e., il est implicite. Il est alors important qu'une méthodologie organisationnelle permette d'expliciter un tel objectif. Ceci doit se faire sous forme d'une expression à utiliser pour dériver l'organisation permettant d'atteindre un tel objectif.

Dans le chapitre suivant (chapitre 3), nous décrivons la méthodologie multi-agent MASA-Method. Cette dernière est une méthodologie complète et mixte. Elle inclut

les trois phases du processus minimal de développement organisationnel des SMA el les quatre aspects de l'ingénierie des organisations expliqués ci-dessus. De plus, le processus méthodologique de MASA-Method est un processus mixte dirigé par objectif. Enfin, MASA-Method propose une approche d'intégration des agents hétérogènes.

Chapitre 3

Fondements de MASA-Method

Dans ce chapitre, nous décrivons la méthodologie de développement de SMA MASA-Method. Selon la classification donnée dans la section 2.5 du chapitre 2, MASA-Method est une méthodologie complète. Elle intègre les trois phase principales du développement organisationnel des SMA. Il est à noter aussi que le processus méthodologique de MASA-Method est mixte et dirigé par objectif. Un processus méthodologique mixte peut prendre en considération les aspects ascendant et descendant dans le développement d'un SMA donné. Une méthodologie mixte permet alors de diriger le développement d'agents de façon descendante en partant de l'organisation comme le font les méthodologies descendantes. Elle permet aussi de réutiliser des agents prédéveloppés en les intégrant dans ce SMA. L'intégration des agents permet à MASA-Method de développer des SMA pouvant intégrer des agents hétérogènes tels que les agents logiciels et humains. L'aspect ascendant de MASA-Method vient du fait que son processus méthodologique inclut l'intégration des agents. Le caractère mixte de MASA-Method permet de surpasser des limites des méthodologies purement descendantes et celles des méthodologies purement ascendantes.

Le processus méthodologique de MASA-Method est dirigé par objectif (Figure 3.1-Page 38). Un processus méthodologique dirigé par objectif est plus complexe qu'un processus méthodologique basé rôle. Cependant, comparé aux processus basé rôle, un processus méthodologique dirigé par objectif présente des avantages certains. Dans un développement dirigé par objectif :

1. Le développeur fixe l'objectif de l'organisation. Il n'a alors pas à prendre en considération l'organisation entière en une seule étape mais plutôt il l'a développe par raffinements successifs,

2. Le modèle de l'organisation sera dérivé de façon systématique en partant de expression formelle de l'objectif de l'organisation,

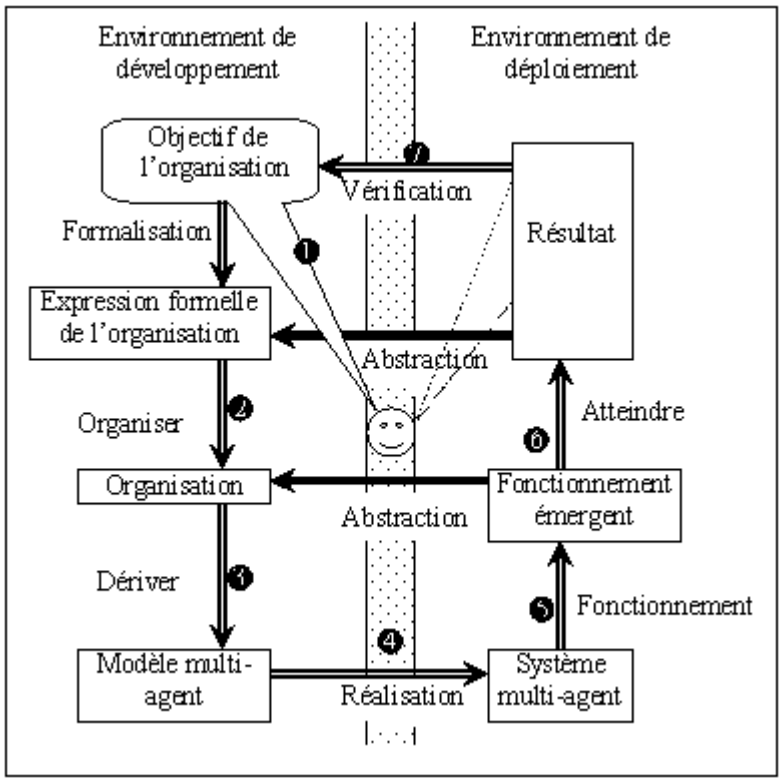

Figure 3.1: Processus méthodologique de développement dirigé par objectif des SMA

3. Le modèle multi-agent sera dérivé de façon systématique à partir du modèle de l'organisation,

4. La vérification sera faite en comparant le résultat émergeant du SMA avec l'objectif fixé. Elle est alors moins difficile que celle des processus de développement basé agent et des processus basés rôle (voir section 2.4 du chapitre 2 pour ces deux derniers processus),

5. Les systèmes développés selon ce processus seront plus sûrs puisque plusieurs de leurs détails sont développés de façon systématique,

6. La vérification peut être faite à trois niveaux différents (Figure 3.1-Page 38): Organisation, expression formelle de l'objectif de l'organisation et l'expression mentale de l'objectif. La vérification de ce dernier niveau est une vérification informelle. Ces niveaux permettent de mieux vérifier le SMA résultat.

Le reste de ce chapitre détaille les fondements de ce processus et les formalismes associés. Il est organisé en cinq sections. La section 3.1 explique le développement des

objectifs de systèmes. Elle permet de considérer une caractérisation procédurale des objectifs comme tâches de systèmes. La section 3.2 présente les notions d'organisations élémentaires et d'organisations composées. Elle décrit la dérivation de ces organisations à partir des expressions des objectifs. La section 3.3 décrit le modèle de systèmes agent. Elle se concentre sur les aspects permettant de dériver ces modèles et sur l'utilisation des agents d'interface pour l'intégration des agents hétérogènes dans un même SMA. La section 3.4 explique l'implémentation des modèles de SMA en Java. La section 3.5 conclut ce chapitre.

3.1 Développement des tâches des systèmes

L'objectif d'un système est le but qu'on lui fixe de telle sorte qu'un tel but sera atteint à l'issue du fonctionnement de ce système. Ce but sera une configuration que l'environnement aura à la fin du fonctionnement du système. Deux approches permettent de développer des systèmes qui permettent d'atteindre des objectifs. La première approche consiste à développer un système général qui peut fonctionner de telle sorte que l'objectif qu'on lui fixe soit atteint. De tels systèmes ont été l'objet de recherche en intelligence artificielle. Actuellement, on est convaincu que de tels systèmes sont soit inexistants soit ils ne peuvent être développés avec les moyens actuels. La deuxième approche consiste à développer un système dont le fonctionnement permet d'atteindre un objectif donné. C'est la deuxième approche que nous adoptons dans MASA-Method et que nous expliquons dans cette section.

3.1.1 Caractérisation des objectifs des systèmes

Un objectif est un couple (M, C) où :

1. M est un modèle d'un environnement,
2. C est une caractérisation de l'objectif en utilisant les éléments de M

Les modèles d'environnements les plus connus et les plus utilisés sont ceux basés données. Ces modèles représentent les environnements comme ensemble de variables dont les valeurs représentent l'état de l'environnement. La caractérisation de l'objectif est alors considérée comme une propriété que les valeurs des variables doivent vérifier. Pour atteindre un tel objectif, il est nécessaire d'agir sur l'environnement jusqu'à ce que ces valeurs vérifient la propriété caractérisant l'objectif. Une telle caractérisation est qualifiée de déclarative. Elle caractérise l'objectif sans indiquer comment l'atteindre.

Pour un modèle orienté objet de l'environnement, une vision semblable peut être adoptée pour la définition de la propriété qu'un état d'un objet doit vérifier. Cependant, il est plus difficile d'exprimer les définitions déclaratives inter-objets en raison de l'encapsulation des variables d'état des objets. Une autre caractérisation des objectifs est la caractérisation procédurale.

Au lieu de la description de l'objectif lui-même, il est nécessaire, dans la caractérisation procédurale, de décrire comment atteindre l'objectif en employant les éléments du modèle de l'environnement. L'expression "comment atteindre l'objectif" peut être qualifiée de tâche. L'expression procédurale d'un objectif est alors définie comme étant un couple (M, T) où :

1. M est un odèle de l'environnement,
2. T est une tâche permettant d'atteindre l'objectif

Le système met en œuvre la tâche. A la fin de cette mise en œuvre, l'objectif du système sera atteint. Le système n'a pas besoin qu'on lui indique l'objectif lui-même.

3.1.2 Description des tâches des systèmes

Dans MASA-Method, nous avons adopté la modélisation orientée objet de l'environnement. Pour la caractérisation des objectifs, nous avons adopté la caractérisation procédurale en utilisant les réseaux de Petri colorés (CPN) [Jensen 94, Jensen 96] pour la modélisation des tâches. Le modèle orienté objet a montré qu'il convient à modéliser des environnements et pour représenter des connaissances. Actuellement, il domine l'industrie logicielle. De leur coté, les CPN sont une classe de réseaux de Pétri riche et populaire, et un bon formalisme pour la modélisation de la concurrence.

Dans la modélisation des tâches des systèmes, un objet du modèle orienté objet sera une abstraction d'un objet de l'environnement. Les méthodes d'un objet donné modélisent comment peut on agir sur cet objet pour effectuer une certaine opération considérée, dans ce contexte, comme élémentaire. Nous considérons alors les tâches élémentaires comme des invocations de méthodes.

Une tâche est élémentaire ou est une combinaison de tâches plus élémentaires qu'on qualifie de sous tâches. Une tâche élémentaire est celle dont l'interprétation se fait par un processus indivisible. Une sous tâche est une tâche, pouvant être décomposable. Les tâches élémentaires peuvent être effectuées séquentiellement et/ou en parallèle, i.e., concurremment. Pour cette raison, nous avons utilisé dans MASA-Method les CPN pour la description des tâches. Les éléments d'un CPN sont illustrés dans la Figure 3.2-Page 41. Un CPN est un quintuple (T, IP, OP, IA, OA). T est un ensemble de

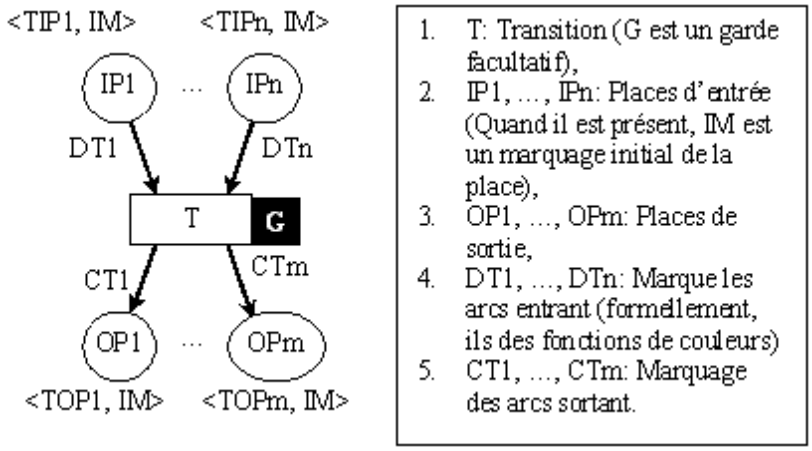

Figure 3.2: Description des éléments d'un CPN et les relations entre eux

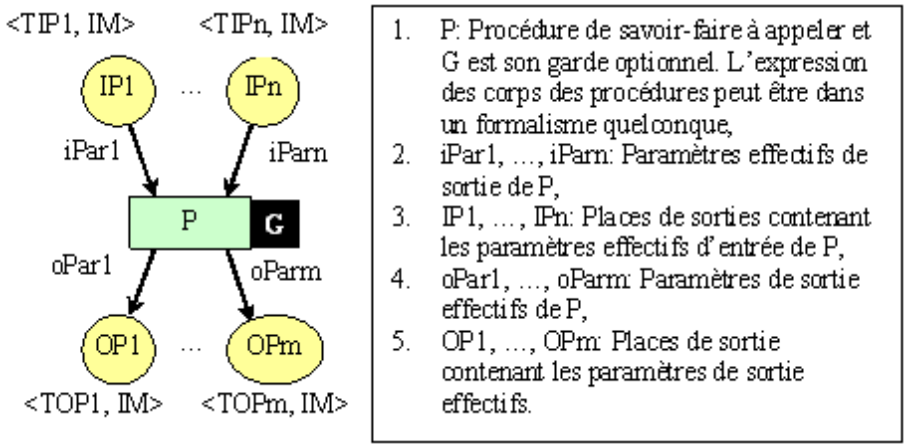

Figure 3.3: Modélisation des tâches par des CPN

transitions qui peuvent être équipées de gardes. IP est un ensemble de places d'entrée et OP est un ensemble de places de sortie. Une place contient les jetons de couleurs données. Une place peut être initialisée par des jetons représentant son marquage initial. IA est un ensemble d'arcs d'entrée et OA est un ensemble d'arcs de sortie. Les arcs sont marqués par des fonctions d'arcs. Les valeurs des paramètres variables des fonctions d'arc sont des marquages des places d'entrée. Les sorties de l'évaluation des fonctions d'arcs sont des jetons. Pour un arc de sortie, ces jetons doivent être ajoutées aux places de sortie. Pour un arc d'entrée, ces jetons doivent être utilisés par la transition.

Dans MASA-Method, la description d'une tâche se fait comme suit (Figure 3.3-Page 41) :

1. Transitions : Une transition T représente une invocation d'une tâche élémentaire. Elle est décrite par O.M. O est un objet. M est l'une des méthodes de O. Le garde de la transition est une fonction des valeurs des arcs entrants,

2. Arcs d'entrée : Les étiquettes des arcs entrants de la transition T doivent être des paramètres effectifs d'entrée iPark de la méthode M,

3. Arcs de sortie : Un arc sortant de T est marqué par un paramètre de sortie oPark de la méthode M,

4. Places d'entrée : Une place P d'entrée d'une transition T contient des paramètres effectifs d'entrée nécessaires à l'invocation de la méthode M dans la transition T,

5. Places de sortie : Un place de sortie d'une transition T contient des paramètres de sorties effectifs résultat de l'invocation de la méthode M dans la transition T.

Les résultats des évaluations des fonctions des arcs d'entrée sont utilisés en tant que paramètres effectifs d'entrée de la méthode O.M. Les évaluations des fonctions d'arcs de sortie oPar utilisent les résultats d'invocation de la méthode O.M comme paramètres variables de ces fonctions. Les résultats des évaluations sont des jetons à ajouter aux places de sortie.

Le processus de développement des objectifs est décrit dans le chapitre 4.

Quand il n'y a pas lieu de confusion, on considère les deux notations (1) O.M où O est un objet et M est une méthode, et (2) P où P est une procédure d'un savoir-faire, sont équivalentes. On suppose alors que P représente une méthode M d'un objet O. Une telle simplification est faite dans le but de simplifier la présentation des explications.

Quand une place d'entrée P d'une transition T1 (contenant la méthode O1.M1) est également une place de sortie pour une autre transition T2 (contenant la méthode O2.M2), P présentera alors une communication implicite entre ces deux méthodes (de O1.M1 à O2.M2). Si les invocations de ces deux méthodes sont distribuées et situées dans deux emplacements distincts, il est nécessaire d'établir une liaison de communication entre ces deux emplacements pour communiquer les résultats de la méthode O1.M1 à la méthode O2.M2. Une telle communication est implicite dans les réseaux de Petri.

Tel que décrit, l'interprétation d'un CPN est faite comme suit. Tant qu'il y a des transitions candidates, choisir une transition T et tirer la transition T. Une transition sera candidate si ses places d'entrée contiennent les valeurs de tous les paramètres variables des fonctions de ses arcs d'entrée. Tirer une transition se fait comme suit :

1. Des places d'entrée, retirer les valeurs des paramètres variables des fonctions des arcs d'entrée. Évaluer les fonctions d'arcs iPar1 ..., iParn et considérer les résultats de ces évaluations comme paramètres d'entrée de la méthode O.M,

2. Invoquer la méthode O.M avec ces paramètres d'entrée,

3. Récupérer les paramètres de sortie de la méthode O.M. Evaluer les fonctions d'arcs oPar1..., oParm en utilisant ces paramètres de sortie comme des paramètres variables d'entrée des fonctions d'arcs. Mettre les résultats de ces évaluations dans les places de sortie.

La terminaison de l'interprétation du CPN tel que décrit parvient dans le cas où il n'y a aucune transition candidate au tirage. Une transition T ne peut être candidate au tirage s'il y a, au moins, une fonction d'arc d'un arc entrant (P, T) qui ne peut être évaluée. L'impossibilité de cette évaluation provient de la non disponibilité des paramètres variable d'une telle fonction dans la place d'entrée P. Après l'arrêt de l'interprétation du CPN, si toutes les places sont vides, la terminaison est parfaitement normale. Tous les paramètres d'entrée sont alors utilisés dans l'invocation des méthodes. De plus, tous les jetons des places sont consommés. Autrement, il y a des problèmes dans la conception du CPN. Ces problèmes sont nombreux tels que les places puits et les inter-blocage. De tels problèmes ne sont pas abordés dans cette thèse.

3.2 Développement des organisations

Cette section explique la description des organisations dans MASA-Method et décrit les dérivations de leurs différents éléments à partir des expressions des tâches des systèmes.

3.2.1 Organisation élémentaire

Une organisation élémentaire O est un système de rôles. Une organisation élémentaire (Figure 3.4-Page 44) peut être décrite comme un couple (SR, CR) où :

1. SR est un ensemble de rôles,

2. CR est une relation de communication entre rôles

Un rôle inclut un ensemble de qualifications (en particulier, le savoir-faire) exigé de l'agent A qui est candidat pour assumer ce rôle pour que cet agent peut assumer ce rôle de façon cohérente. Le rôle inclut également un protocole de communication

Nous résumons la description d'une organisation élémentaire O
sous forme d'un couple (SR, CR) où :

1. SR : SR = { r / r est un rôle }
   où r est un triplet sous la forme (SKP, SCP, T)
   a. SKP : SKP = {kp / kp est une procédure du savoir-faire nécessaire.
      Une procédure du savoir-faire est une combinaison
      d'un ensemble d'actions}
      L'ensemble des actions SAct est défini comme suit :
      SAct = {Action / Action est une description d'une actions
      élémentaire qui sera accomplie sur l'environnement
      par l'agent qui assumera le rôle r}
   b. SCP : SCP = { cp / cp est un protocole de communication },
   c. T : Tâche est la partie de tâche de l'organisation
      que représente le rôle r dans l'organisation.
2. CR: CR = { (r1, r2) / (r1, r2) ∈ SR x SR et r1 est en relation
   de communication avec le rôle r2}

Figure 3.4: Définition d'une organisation élémentaire

indiquant à l'agent A comment il doit communiquer avec les agents assumant les autres rôles de l'organisation O, tels qu'exécuter les ordres de ses supérieurs et servir ses pairs. Finalement, le rôle doit inclure une tâche indiquant sa partie du travail dans l'organisation et comment l'agent A doit agir sur l'environnement en utilisant ses qualifications. Le savoir-faire exigé se compose d'un ensemble de procédures. Une procédure du savoir-faire est une combinaison d'actions élémentaires à effectuer sur un environnement pour accomplir une mission donnée. La description de la tâche d'une organisation est une fonction des invocations des procédures du savoir-faire des rôles.

En réalité, une organisation se crée pour un objectif donné. Méthodologiquement, on définit un objectif et on développe ensuite l'organisation permettant d'atteindre cet objectif. Certains auteurs tels que [Müller 00] intègrent l'objectif de l'organisation comme étant sa fonction dans la description de l'organisation elle-même. Une telle fonction est une définition procédurale de l'objectif de l'organisation. D'autres auteurs ne considèrent pas l'objectif d'organisation, tel que les méthodologies MaSE [DeLoach 02a], Gaia [Zambonelli 03] et Alaaddin [Ferber 01]. Dans de tels travaux, l'objectif de l'organisation est seulement implicite. Dans MASA-Method, nous adoptons une vision différente des deux précédentes. L'objectif de l'organisation est présent. Cependant, cette présence est dans le processus méthodologique du développement de l'organisation. Il ne sera pas alors partagé par les rôles en faisant partie de la description de l'organisation. La description de l'objectif du système sera alors raffinée. Les

éléments de l'organisation seront alors dérivés de l'objectif.

Initialement, deux raffinements de l'objectif du système peuvent être faits. Ces deux raffinements se feront intuitivement, c'est-à-dire, que nous ne proposons pas de méthode pour effectuer ces raffinements. Ces raffinements peuvent être décrits comme suit :

1. Les tâches élémentaires sont décrites dans l'expression de la tâche du système sans préciser les rôles qui seront responsables de leur réalisation. Le premier raffinement consiste alors à ajouter aux tâches élémentaires les rôles qui seront responsables de leur réalisation,

2. Définir les protocoles de communication des rôles ajoutés aux tâches élémentaires.

Le résultat des deux raffinements précédents de la tâche d'un système est une tâche de l'organisation OO. Celle-ci est un couple (SpR, Tâche) où :

1. SpR est un ensemble de rôles partiellement définis. Chaque rôle a un identificateur et un ensemble de protocoles de communication. SpR est défini alors comme suit :

   {< idR, SCP > / idR : Identificateur de rôle, SCP: Ensemble de protocoles de communication},

2. Tâche est une description de la tâche du système avec des rôles des agents qui accompliront les tâches élémentaires. Elle représentera alors la tâche de l'organisation

Telle que définie, la tâche d'une organisation contient les éléments nécessaires à partir desquels on peut dériver les composantes de cette organisation. L'adoption de cette organisation par un ensemble d'agents permettra d'atteindre l'objectif du système définit par sa tâche. Comme décrits dans la Figure 3.4-Page 44, les éléments comportent un ensemble de rôles et une relation de communication entre rôles. Un rôle inclut des protocoles de communication, un savoir-faire nécessaire et une tâche. Les protocoles de communication sont définis dans la tâche de l'organisation. Le savoir-faire nécessaire et la tâche d'un rôle donné ainsi que la relation de communication seront dérivés de la tâche de l'organisation. Le savoir-faire nécessaire d'un rôle R est l'ensemble de toutes les tâches élémentaires auxquelles le rôle R est associé dans la tâche de l'organisation. Une tâche élémentaire sera alors une procédure du savoir-faire nécessaire du rôle. Les tâches des rôles sont le résultat de la distribution sur les rôles de la tâche de l'organisation. La relation de communication entre rôles est le résultat

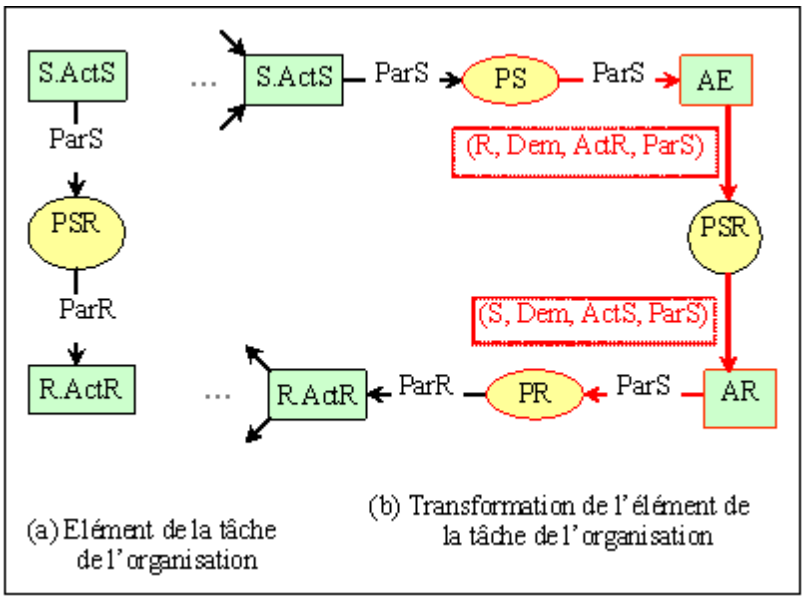

Figure 3.5: Les places comme moyens de communication

de l'explicitation de la relation implicite de communication qui existe déjà dans le CPN de la tâche de l'organisation entre les tâches élémentaires.

Dérivation des relations de communication entre les rôles

Le formalisme des CPN fait abstraction des communications qui doivent exister entre les agents qui accomplissent les différentes actions induisant les transitions du CPN. Dans le cas d'un système multi-agent, certaines actions sont accomplies par des agents différents. Il est alors important d'expliciter les communications. Dans ce qui suit, nous expliquons comment expliciter les communications implicite dans le CPN.

Soit une transition T1 contenant l'invocation de la procédure du savoir-faire R1.P1. R1 est un rôle et P1 est une procédure du savoir-faire de R1. P1 sera sous la forme O1.M1. O1 est un objet et M1 est l'une de ses méthodes. Soit une autre transition T2 contenant l'invocation de la procédure du savoir-faire R2.P2. Supposons la situation suivante :

1. P1 a un paramètre de sortie Par1 qu'elle place dans une place Q,
2. T1 est la seule transition d'entrée de Q,
3. L'invocation de P2 utilise un paramètre d'entrée Par2 de la place Q,
4. T2 est la seule transition de sortie de la place Q.

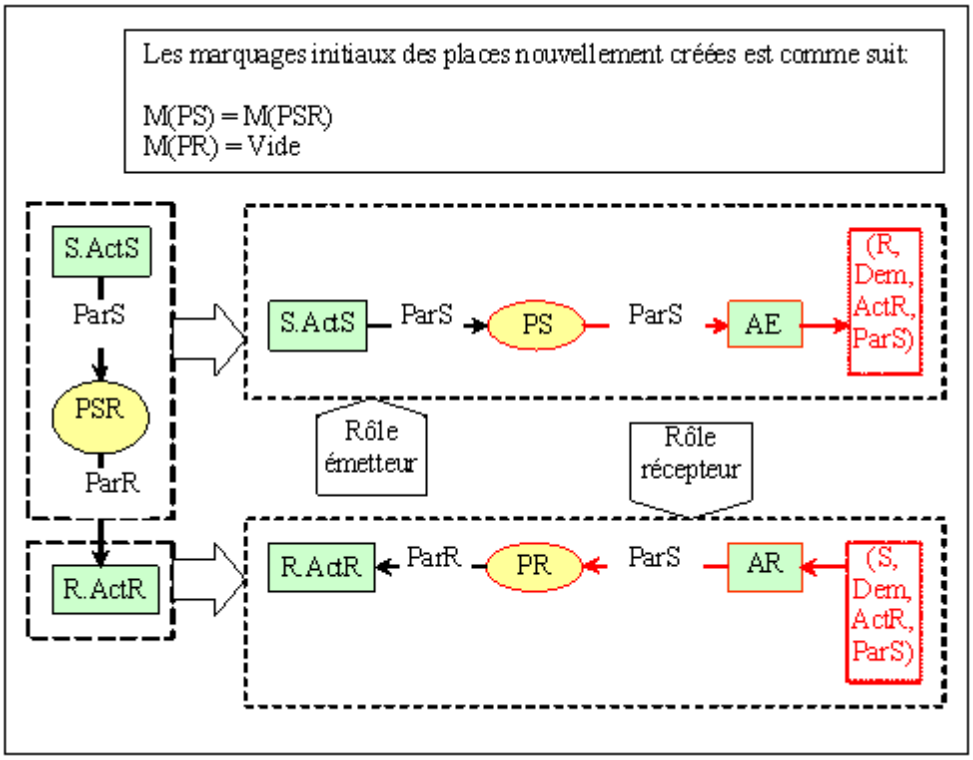

Figure 3.6: Décomposition de l'élément de la tâche de l'organisation: Le cas de communication un-à-un

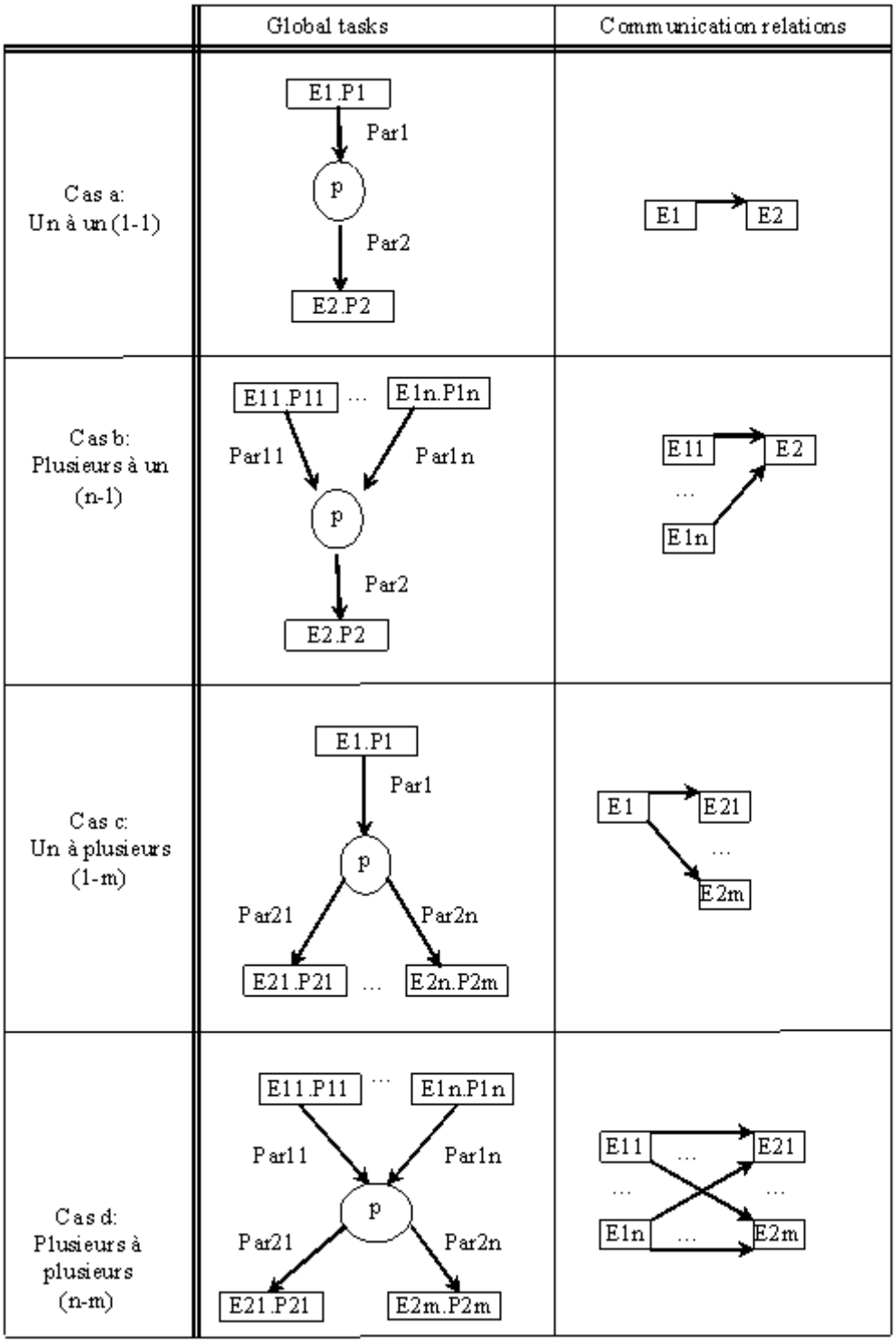

Figure 3.7: Règles de dérivation des relations de communication

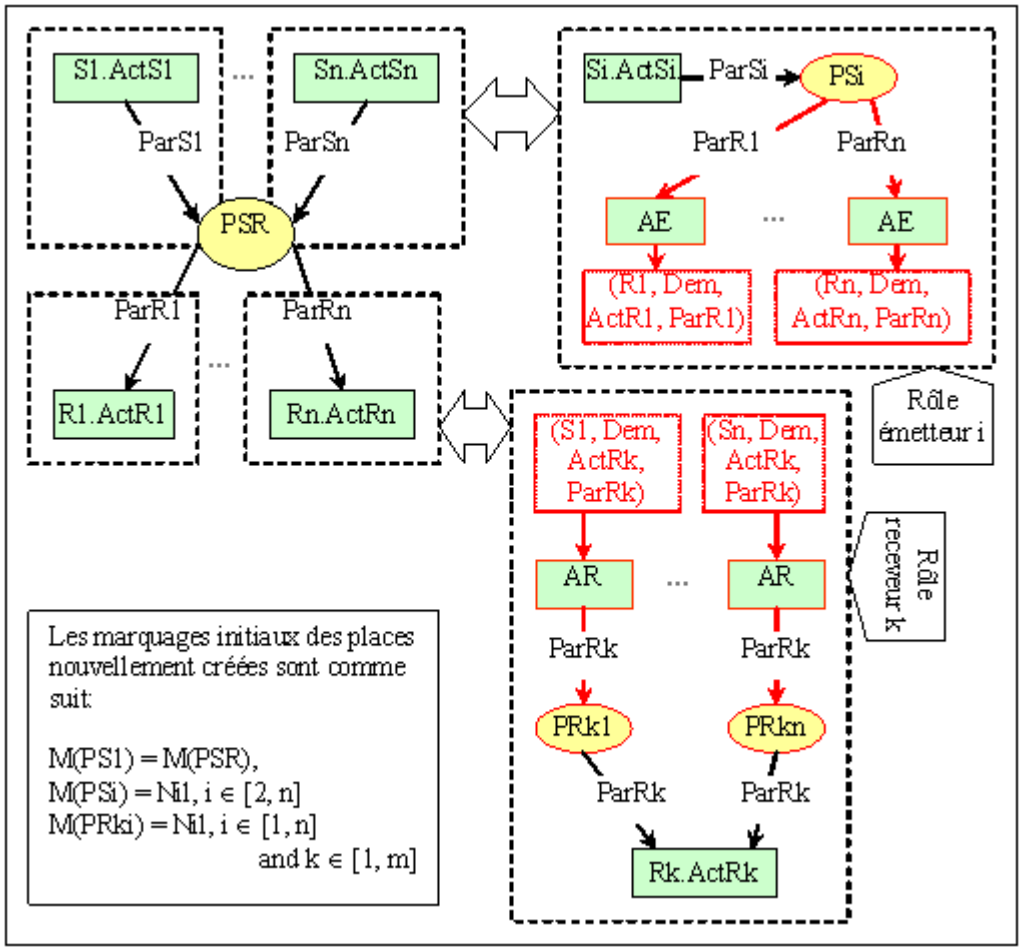

Figure 3.8: Décomposition de l'élément de la tâche de l'organisation: Le cas de communication un à plusieurs

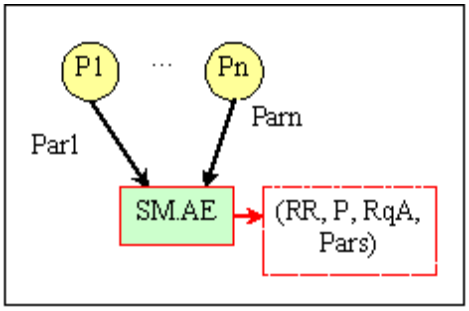

Figure 3.9: Description de l'émission d'un acte de langage

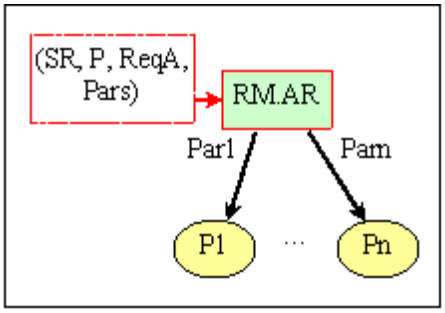

Figure 3.10: Description de la réception d'un acte de langage

Si R2 = R1, les données d'entrée et de sortie seront situés dans l'espace de données de l'agent assumant le rôle R1. Cependant, si R1 est différent de R2, le paramètre de sortie Par1 sera produit par l'invocation de P1 par le rôle R1. Le rôle R2 doit utiliser cette sortie comme entrée pour l'invocation de P2. Le rôle R1 peut être assumé par un agent A différent de l'agent B qui assume le rôle R2. Dans ce cas, nous devons établir une communication de R1 vers R2. Quand l'invocation de P1 se termine et produit Par1, R1 envoie Par1 à R2 pour le placer dans la place Q. Quand R2 invoque la procédure P2, il utilise les jetons de la place Q comme paramètre d'entrée Par2 de la procédure P2. Ces communications sont du genre "un à un" (d'un rôle à un rôle différent).

Une place peut héberger des jetons provenant de plusieurs transitions. Chaque transition peut inclure un rôle différent de ceux des autres. La relation de communication sera alors plusieurs à un. En outre, plusieurs transitions comprenant différents rôles peuvent utiliser une place. Les entrées de cette place sont produites par une seule transition. La relation de communication sera alors un à plusieurs. Finalement, plusieurs transitions, incluant des rôles différents, produisent des jetons d'une palce donnée et plusieurs autres transitions, incluant d'autres rôles différents, utilisent le

contenu de cette place. La relation de communication sera alors plusieurs à plusieurs. Ces différents cas sont récapitulés dans la Figure 3.7-Page 48.

La relation de communication d'une organisation O peut être dérivée de la description formelle de la tâche globale comme suit :

1. L'ensemble des rôles (nœuds de la relation de communication) est celui des tous les rôles référencés dans l'expression formelle de la tâche de l'organisation,

2. Considérons deux transitions R1.P1 et R2.P2. R1 et R2 sont deux rôles et P1 est une procédure du savoir-faire de R1 et P2 est une procédure du savoir-faire de R2. Soit Q une place de sortie pour la transition R1.P1. De plus, Q est une place d'entrée pour la transition R2.P2. Dans la relation de communication de l'organisation, ajouter un lien de communication du rôle R1 vers le rôle R2, s'il n'existe pas déjà.

Description des tâches des rôles

Les relations de communication entre les rôles seront identifiées en 3.2.1.1. Ces relations sont utilisées par les rôles (plus précisément, par les agents assumant les rôles) pour échanger des messages entre eu. Dans ce suit, nous expliquons la modélisation des tâches des rôles. Ces modélisations incluent la description des messages échangés entre ces rôles. Les tâches des rôles sont modélisées dans le formalisme des CPN avec certaines particularités simples.

Tout d'abord, il faut remarquer que l'attachement des rôles aux invocations des procédures du savoir-faire dans la description de la tâche d'un rôle donné R n'est pas nécessaire puisque cette tâche est celle d'un seul rôle. Toutes les procédures du savoir-faire seront invoquées par ce même rôle R. La communication entre deux rôles se fait en envoyant un message par un rôle SR et la réception de ce message par un autre rôle RR selon un protocole de communication basé sur les actes de langage. L'envoi du message se fait par une procédure du savoir-faire du rôle SR et sa réception se fait par une procédure du savoir-faire du rôle RR. Ces procédures du savoir-faire sont des méthodes d'objets. Elles peuvent être décrites comme suit :

1. L'envoi d'un message sera faite par une méthode d'envoi d'actes de langage SM.AE. SM est l'objet représentant le moyen d'envoi de messages sur l'environnement. L'objet associé à SM sera un effecteur de l'agent qui assumera le rôle SR. L'envoi d'un message sera modélisée par l'invocation de la méthode SM.AE,

2. La réception d'un message sera faite par une méthode de réception d'actes de langages RM.AR. RM est l'objet représentant le moyen de réception de messages

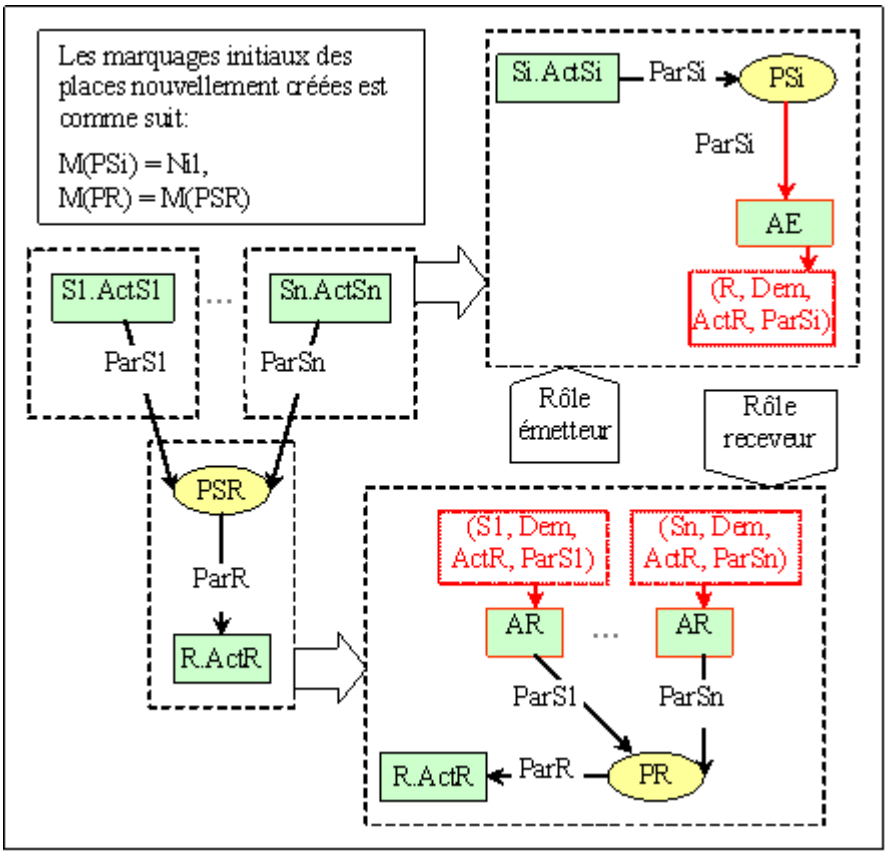

Figure 3.11: Décomposition d'un élément de la tâche de l'organisation: Le cas de communication plusieurs à un

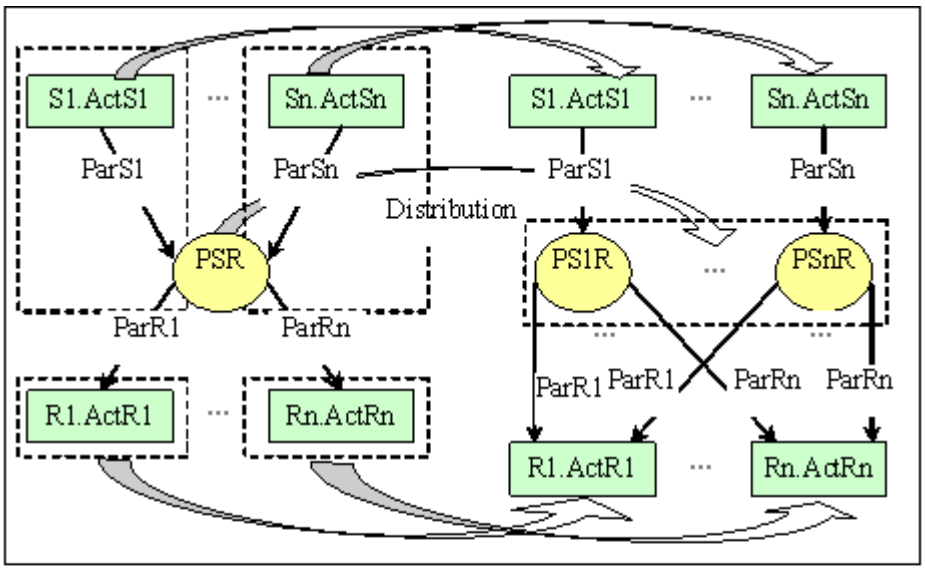

Figure 3.12: Transformation d'un cas de communication plusieurs à plusieurs d'un élément de la tâche de l'organisation en plusieurs communications un à plusieurs

de l'environnement. L'objet associé à RM sera un senseur de l'agent qui assumera le rôle RR. La réception d'un message sera modélisée par l'invocation de la méthode RM.AR.

Comme expliquée précédemment, l'invocation d'une méthode est décrite par une transition contenant la méthode à invoquer. Malgré que les formalismes des réseaux de Pétri décrivent des tâches concurrentes, les réseaux de Petri font abstraction des communications entre ceux qui causent les transitions du système. Les travaux sur la modélisation des comportements des agents utilisent les réseaux de Pétri autonomes. Ils présentent les communications entre les agents par des arcs (voir [Xu 01, Xu 03], par exemple). Ces arcs présentent des dépendances entre les comportements des agents et influencent leur autonomie qui est une des caractéristiques fondamentales des agents. Dans MASA-Method, nous utilisons les CPN non autonomes. Dans les réseaux de Pétri autonomes, le tirage des transitions est soumis aux seuls états internes des réseaux de

Pétri. En particulier, ces états concernent le marquage des places. Par contre, dans les réseaux de Pétri non autonomes, des transitions sont soumises à des événements externes en plus de leurs états internes.

Dans MASA-Method, nous considérons les messages comme événements attachés aux transitions. Un arc sortant d'une transition schématise l'envoi d'un message tandis qu'un arc entrant schématise sa réception. Ce qui se produit entre l'émission du message et sa réception n'est pas considérée ici. Ceci est considéré comme faisant partie de l'implémentation du protocole de communication.

Dans la modélisation des communications, les événements sont de deux types : envoi et réception de messages. L'envoi d'un message est modélisé en attachant un événement à une transition SM.AE sous forme d'un arc sortant (Figure 3.9-Page 50). Le message à envoyer est un quadruplet (RR, P, RqA, Pars) où :

1. RR : Identification du rôle auquel le message sera envoyé,
2. P : Performatif (ordre, souhait. . .),
3. RqA : L'action demandée,
4. Pars : Paramètres de l'action.

Une transition SM.AE a des arcs entrants contenant les paramètres d'entrée de l'action demandée RqA. Si les places de ces arcs entrants contiennent les paramètres d'entrée nécessaire à l'action demandée, la transition SM.AE peut être tirée en invoquant la méthode SM.AE par le rôle SR. Cette invocation permettra d'envoyer au rôle RR le message (SR, P, RqA, pairs).

La réception d'un message est modélisée en attachant un événement à une transition RM.AR sous forme un arc entrant (Figure 3.10-Page 50). Ce message est un quadruplet (SR, P, RqA, Pars) où:

1. SR : Une identification du rôle duquel le message sera reçu,
2. P : Un performatif,
3. RqA : L'action demandée,
4. Pars : Les paramètres de l'action.

Une transition RM.AR a des arcs sortants contenant les paramètres de sortie de l'action demandée RqA. Tirer une transition RM.AR se fait s'il y a un message venant d'un rôle SR et, ce message contient l'action demandée RqA et les paramètres Pars. Ceci permettra de recevoir du rôle SR le message (SR, P, RqA, Pars) et de mettre les paramètres Pars dans les places appropriées pour que l'action demandée RqA qui utilise ces paramètres puisse être invoquée.

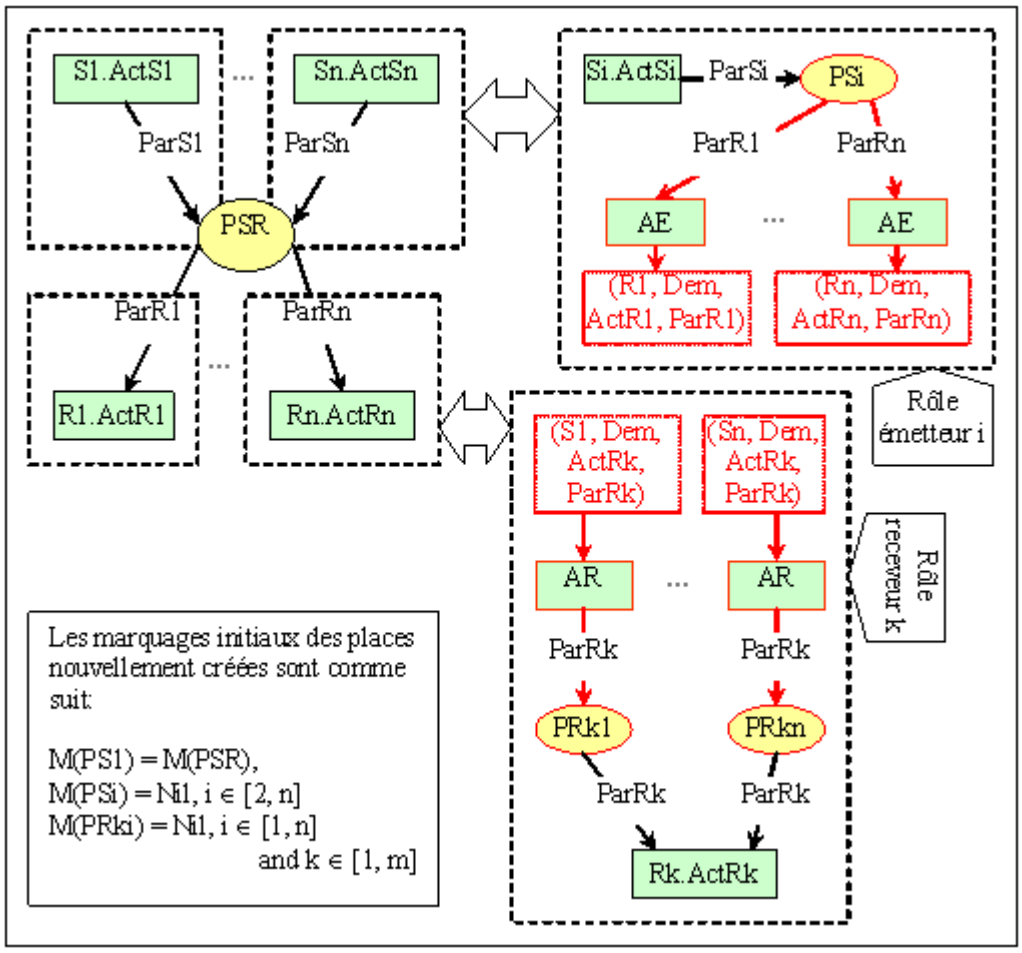

Figure 3.13: Décomposition d'un élément de la tâche de l'organisation: Le cas de communication plusieurs à plusieurs (en utilisant la transformation de la figure 3.12)

Dérivation des tâches des rôles

Soit une transition T de la forme R1.P1 indiquant l'invocation de la procédure P1 du savoir-faire par le rôle R1. Dans la description des tâches des organisations, le marquage d'une place Q à partir de laquelle est issue un arc entrant une transition T peut être :

1. Un marquage initial par lequel la place Q est marquée initialement,
2. Issue d'une autre transition R.P2 comme résultat de l'invocation de la procédure P2 qui sera accomplie par le rôle R.

Dans le premier cas, le rôle R1 initialisera la place Q. Il utilisera, ensuite, le résultat de cette initialisation pour invoquer la méthode P1. Les opérations se passe alors dans

l'espace de l'agent assumant le rôle R1. Dans le deuxième cas, le rôle R peut être R1 lui-même ou un autre rôle R2. Si R est le même que R1 alors le rôle R1 mettra en œuvre la procédure P2, place le résultat dans la place Q et invoque la procédure P1. Dans ce cas aussi, les opérations se passe dans l'espace de l'agent assumant le rôle R1. Cependant, le cas où le rôle R est un autre rôle R2 différent du rôle R1 exige une communication entre les rôles R1 et R2 comme discuté en 3.2.1.1. Cette communication permet de transmettre les résultats de la mise en œuvre de la procédure P2 du rôle R2 vers le rôle R1 pour que R1 utilise ces résultats dans l'invocation de la procédure P1. Nous considérons la place Q comme représentant d'un lien de communication de R2 vers R1 (Figure 3.5-Page 46, la place Q est représenté par la place PSR).

Quatre cas de communications entre rôles peuvent être identifiés (voir section 3.2.1.1) :

1. Un à un (Un rôle soumet et un rôle reçoit) : La règle de distribution de ce cas est illustré dans la Figure 3.6-Page 47,

2. Un à plusieurs (Un rôle soumet et plusieurs autres reçoivent) : La règle de distribution de ce cas est illustré dans la Figure 3.8-Page 49,

3. Plusieurs à un (Plusieurs rôles soumettent et un reçoit) : La règle de distribution de ce cas est illustré dans la Figure 3.11-Page 52,

4. Plusieurs à plusieurs (Plusieurs rôles soumettent et plusieurs autres rôles reçoivent) : La règle de distribution de ce cas est illustré dans la Figure 3.13-Page 55.

Le premier sous cas est illustré dans la Figure 3.5-Page 46. Dans la partie a de cette figure, la place PSR sera considéré comme représentant un lien de communication entre les deux rôles S (expéditeur) et R (récepteur). La réalisation de la communication entre les rôles S et R sera faite par l'émission de messages par S et leur réception par R. Des places et des transitions additionnelles nécessaires pour cette communication peuvent être introduite comme le montre la Figure 3.10-Page 50. Si on fait abstraction des liens de communication représentés par la place PSR, la place PSR sera supprimée. La communication sera considérée alors comme un événement sortant pour la transition SM.AE et comme un événement entrant pour la transition RM.AR. La place PSR conserve son marquage initial.

La règle de distribution pour le cas un à un est donnée dans la Figure 3.6-Page 47. Sur la Figure 3.6-Page 47, le marquage de PSR sera attribué à PS tandis que le marquage de la place PR sera initialisé à vide. Un raisonnement similaire peut être fait pour les trois autres cas. Leurs règles de décompositions sont illustrées dans les Figures

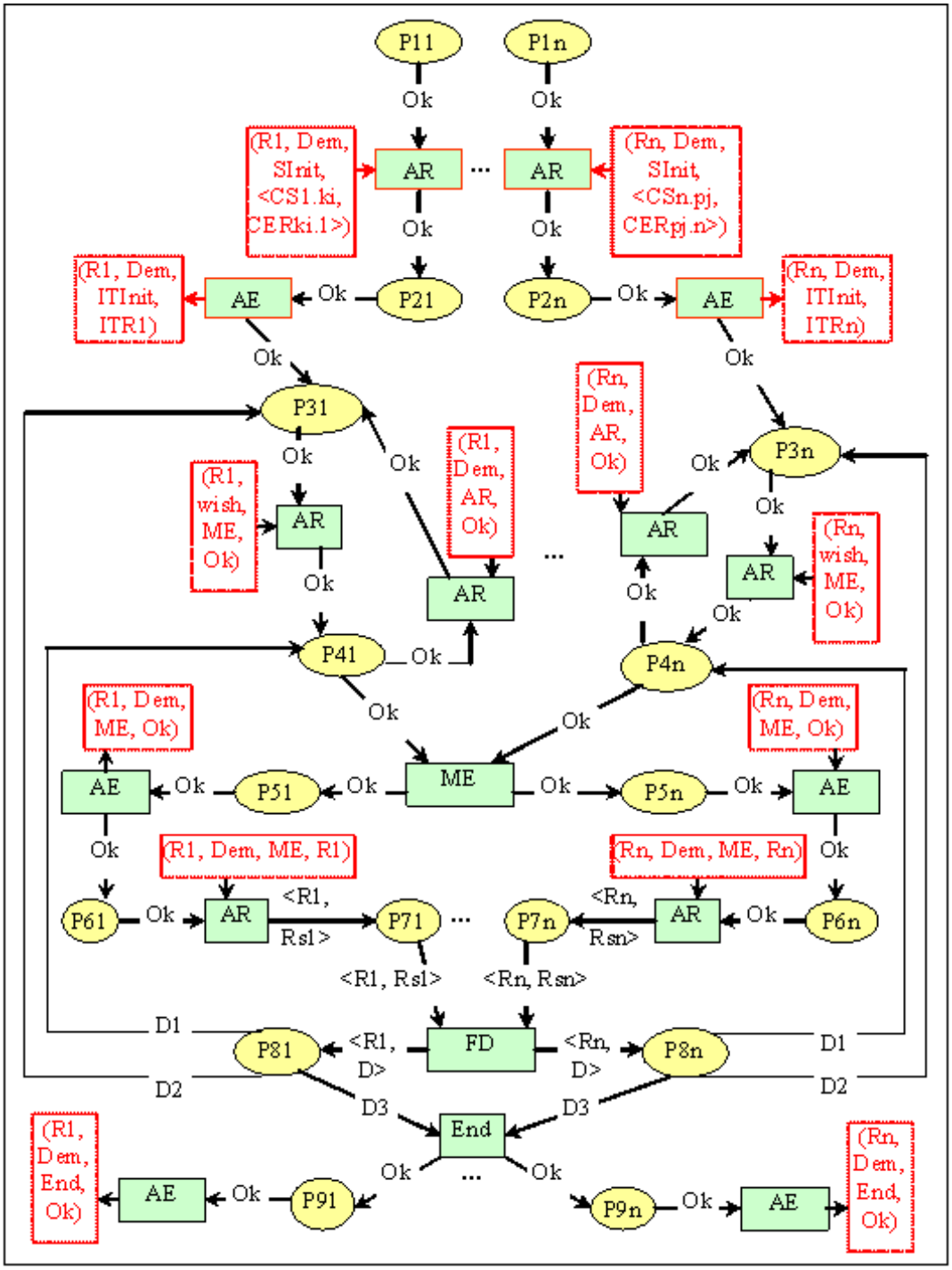

Figure 3.14: Le rôle controllerRole. Nous avons précisé seulement les procédures invoquées sans les objets auxquels appartiennent ces méthodes.

3.8-Page 49, 3.11-Page 52 et 3.13-Page 55. Dans le cas plusieurs à plusieurs, comme illustré sur la Figure 3.12-Page 53, la place de communication PSR est partagée par plusieurs rôles émetteurs et plusieurs rôles récepteurs. La communication se fait entre deux cotés : coté émission et coté réception. Pour les cas un à un, un à plusieurs et plusieurs à plusieurs, nous avons établis des communications entre un rôle, d'un coté (émission ou réception), et un ou plusieurs rôles, de l'autre coté (émission ou réception). Nous avons alors attribué la place de communication PSR à un coté ou à l'autre mais qui comporte un seul rôle. Dans le cas plusieurs à plusieurs, nous avons plusieurs rôles des cotés. L'attribution de la place de communication PSR à n'importe quel coté fait qu'elle soit aussi partagée par plusieurs rôles. Le cas plusieurs à plusieurs signifie que plusieurs rôles déposent des jetons dans une place destinée à ce que plusieurs autres rôles retirent ces jetons. Ceci peut être fait différemment. Chaque rôle émetteur dépose ses jetons dans une place propre à lui. Les rôles récepteurs retirent les jetons de cette place. De cette façon, tous les rôles émetteurs soumettent à tous les rôles récepteurs. Cette transformation du CPN est illustrée sur la Figure 3.12-Page 53.

Terminaison de la mise en œuvre des tâches des rôles

Pour la tâche d'un système telle décrite dans la sous-section 3.2.2, toutes les places se trouvent dans un même emplacement. La condition de terminaison consistait à vérifier qu'il n'y a aucune transition candidate. Cependant, les tâches des rôles assumées par des agents différents se trouvent dans des emplacements différents. Ces tâches seront alors mise en œuvre de façon distribuée. La condition de terminaison devient alors celle de la détection de la terminaison des mises en œuvre des tâches de manière distribuée. La détection de la terminaison d'un système réparti est une tâche distribuée. Les paragraphes suivants expliquent le raffinement de la terminaison de l'interprétation du CPN comme détection de la terminaison des mises en œuvre distribuée des tâches des rôles. Une telle condition doit, cependant, hériter la condition de terminaison de l'interprétation du CPN de la tâche du système. Cette dernière condition est exprimée ainsi "Il n'y a aucune transition candidate dans les tâches de tous les rôles".

La condition de la détection de la terminaison de la mise en oeuvre des tâches des rôles doit hériter aussi les éléments de la détection de la terminaison de la mise en œuvre de tâches distribuées. Cette dernière condition est celle d'aucun messages n'est en transit. Ce qui signifie qu'il n'y a aucun message soumis par un agent assumant un rôle donné qui n'est pas encore reçus par l'agent assumant une rôle qui doit recevoir ce message. Une telle condition est la même que celle exprimée dans [Dijkstra 83] pour des processus distribués. Cette condition peut être énoncée pour les agents comme suit : "À tout moment, le nombre de messages en transit est égal à la somme de tous

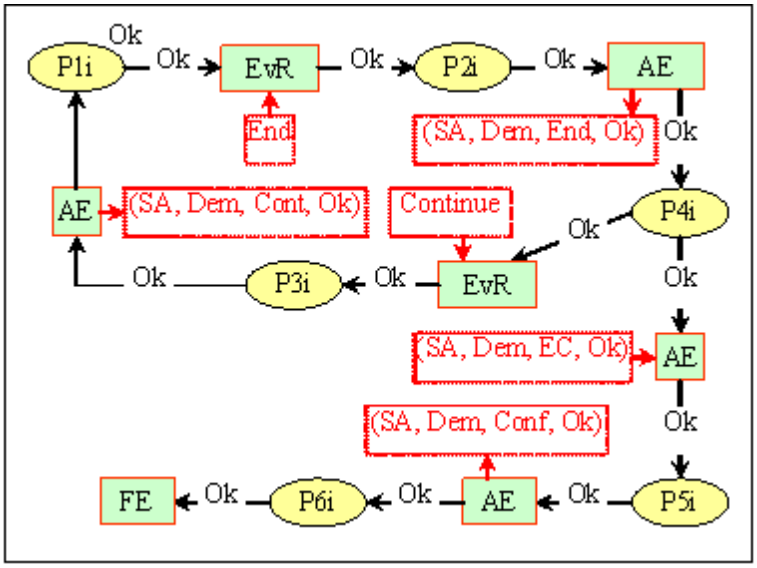

Figure 3.15: Rôle AnyRole

les déficits en messages. Le déficit en messages pour un agent A est le nombre de messages que l'agent "A" a envoyés jusqu'ici moins le nombre de messages que cet agent a reçus. L'absence de message en transit signifie la somme nulle des déficits de tous les agents."

Les actions nécessaires à la détection de la terminaison doivent être intégrées dans les comportements des agents pour que ces agents puissent les mettre en œuvre pendant leurs fonctionnements. Dans le cas d'un développement d'une organisation de SMA, SMA développés en utilisant MASA-Method, les actions nécessaires à la détection de la terminaison peuvent être assumées comme des rôles. La comptabilisation des messages en transit sera assurée par un agent assumant le rôle d'un contrôleur appelé "controllerRole" (Figure 3.14-Page 57). Chaque agent doit maintenir un déficit de messages en assumant un rôle que nous appelons "anyRole" (Figure 3.15-Page 59). Il y aurait alors plusieurs rôles de type "anyRole" et un seul rôle de type "controllerRole". La tâche du rôle controllerRole est complexe. Il peut être décomposé en plusieurs rôles organisés. Ces rôles seront intégrés en tant qu'élément de l'organisation du SMA ou comme une sous organisation d'une organisation composée (voir section 3.2.2 pour les sous-organisations et les organisations composées).

Sur la Figure 3.14-Page 57 de la tâche du rôle controllerRole, la transition ME peut être la fin de la mise en œuvre des tâches des rôles si tous les agents ont émet un Ok. Rsi est la réponse du rôle Ri à la demande, par le rôle controllerRole, de la confirmation de la fin de la mise en œuvre de sa tâche. La transition FD est la

décision finale. Si tous les rôles confirment la fin de la mise en œuvre de leurs tâches, la décision finale sera la fin de l'exécution du SMA. Autrement, nous aurons deux cas. Ceux qui ont confirmé la fin seront considérés comme les premières sollicitations de la fin. Ceux qui n'ont pas confirmé la fin seront considérés comme étant à leur états initiaux. L'état initial est celui d'attendre la première sollicitation. La transition End est la fin effective de l'exécution du SMA.

3.2.2 Organisations composées

Certaines organisations sont larges et leur développement est difficile. Il est alors nécessaire de décomposer de telles organisations en sous organisations pour pouvoir les maîtriser. L'organisation résultat est dite organisation composée. Une organisation composée est appelée mère relativement à ses sous organisations. Par conséquent, une sous-organisation est appelée fille d'une organisation composée. Deux sous organisations d'une même organisation composée sont appelées sœurs.

Une sous organisation est, elle aussi, une organisation (élémentaire ou composée). Elle a la particularité d'être un élément d'une organisation composée. En tant qu'organisation, une sous organisation possède une structure et une tâche. En plus, une sous organisation possède un ensemble de rôles dits représentants. De tels rôles représentent la sous organisation dans ses communications avec ses sœurs.

Une organisation composée possède une structure et une tâche. Comme expliqué dans la sous section 3.1.2, une tâche est une combinaison de tâches plus élémentaires. La tâche d'une sous-organisation est vue comme étant plus élémentaire que la tâche de sa mère. La tâche de l'organisation composée est une combinaison des invocations des tâches de ses sous organisations. La relation de communication d'une organisation composée est une relation de communication entre les représentants de ses sous organisations.

Fondamentalement, les organisations sont basées sur des rôles. Leur structuration en sous organisations est un aspect méthodologique dans le but de les rendre modulaire. Les sous organisations d'une organisation composée doivent alors être substituées pour rendre l'organisation sous sa forme élémentaire. Cette substitution va permettre d'identifier les éléments des différents rôles. L'algorithme de substitution est donné sur la Figure 3.16-Page 61. Une telle substitution sera montrée sur un exemple au chapitre 5.

Etant donnée une définition d'une organisation O
comme un quadruplet (O-SE, O-CR, O-GT)
Soit une organisation composée CO :
CO = (CO-SE, CO-CR, CO-GT)
L'algorithme de substitution permet de produire
une organisation plus élémentaire S-CO qui est un triplet :
(S-SE, S-CR) où :

1. S-SE est l'ensemble des éléments de S-CO.
   Il est défini comme suit : $U_{o \in CO-SE}$ (o-SE)
2. S-CR est la relation de communication de S-CO.
   Elle est défini comme suit :
   CO-CR $\cup$ ($\cup_{o \in CO-SE}$ (o-CR))
3. S-GT est la tâche de l'organisation S-CO.
   Elle est définie comme suit :
   S-GT = Combinaison (
   { Substitution (o-GT) / o $\in$ CO-SE })

Figure 3.16: Substitution d'une organisation composée

3.3 Développement des modèles multi-agent

Un SMA est constitué d'un ensemble d'agents et de relations de communication entre agents de cet ensemble. Les relations physiques de communication entre les agents sont assurées par l'environnement dans lequel le SMA est destiné à fonctionner. Nous considérons les relations physiques de communication comme une implémentation des relations de communications logiques. Les relations logiques de communication seront intégrées dans les agents. Chaque agent doit connaître les agents de l'organisation avec lesquels il est en relation de communication. Un modèle de SMA sera alors constitué d'un ensemble de modèles d'agents intégrant les relations logiques de communication. De telles relations seront dérivées de la relation de communication de l'organisation. Elles seront incluses explicitement dans les senseurs et effecteurs de communication des agents.

Dans cette section, nous considérons le développement des modèles de SMA permettront de mettre en oeuvre des organisations. Ces modèles sont basés sur les aspects présentés au chapitre 2.

3.3.1 Agents

Au chapitre 2, nous avons définit un agent comme un dix-uplet (C, E, SF, T, PC, PS, SH, EH, SC, EC) où :

1. C: Contrôle,
2. E: Etat,
3. SF: Savoir-faire,
4. T: Tâche,
5. PC: Protocoles de communication,
6. PS: Protocoles de synchronisation,
7. SH: Senseurs habituels,
8. EH: Effecteurs habituels,
9. SC: Senseurs de communication,
10. EC: Effecteurs de communication

Dans un développement organisationnel des agents, un agent assume un ou plusieurs rôles. Son contrôle, son savoir-faire, son état et sa tâche dépendent de la nature de l'organisation et des rôles qu'il va assumer. En considérant que chaque organisation a ses propres protocoles de communication, les protocoles de communication et de synchronisation d'un agent peuvent être dépendant de l'organisation. Ces types de communication font partie des savoirs faire des agents sur la façon de communiquer avec les agents assurant les autres rôles. Les protocoles de communication et de synchronisation sont basés sur des modèles généraux. Les protocoles de communication sont basés sur les actes de langage et les la synchronisation est basé sur l'utilisation des aspects des réseaux de Petri.

Les senseurs et les effecteurs habituels d'un agent dépendent de la nature de l'environnement et des tâches qu'il est sensé accomplir pour assumer les rôles auxquels il est assigné. Ses senseurs et ses effecteurs de communication dépendent également de la nature de l'environnement et des communications qui doivent exister entre des rôles que les agents assumeront. Finalement, sa tâche est constituée de l'ensemble des tâches des rôles qu'il assumera dans une distribution de rôles sur des agents.

3.3.2 Agent d'interface

Habituellement, un humain emploie les systèmes logiciels par l'invocation d'outils. Pour cela, il détermine son objectif et planifie les actions lui permettant d'atteindre un tel objectif. Ces actions peuvent être subdivisées en deux classes. Les actions qui doivent être faites par l'humain et les actions qui doivent être faites par le système

logiciel. Puisque l'humain est intelligent son plan est imprécis. Il met à jour ce plan selon la situation qui surgit en reconsidérant des actions, en ajoutant de nouvelles actions, ... L'humain détient alors son plan et assure son exécution. Ce plan inclut les tâches suivantes :

1. Séquencer les actions du plan,
2. Exécuter des actions destinées à lui,
3. A travers une interface utilisateur, demander au système logiciel d'exécuter les actions qu'il a prévu à ce système.

Les inconvénients de cette approche sont nombreux tels que :

1. Le développement des interfaces est basé sur une recherche interminable d'un modèle universel valide pour l'utilisation de n'importe quel logiciel par n'importe quel rôle humain,
2. Plusieurs incohérences peuvent se produire durant l'utilisation et/ou du séquencement des outils,
3. Cette approche ne permet pas aux concepteurs d'utiliser des capacités intellectuelles des humains pour servir l'objectif du système à développer,
4. La détention du plan et son exécution par l'humain est embarrassante, improductive, elle cause des erreurs dans l'exécution du plan et dans son développement,
5. etc.

Les travaux de recherches entrepris sur des agents d'interface ([Middleton 01]) sont basés sur le concept traditionnel du client/serveur. Ils voient l'humain comme client le qualifiant d'utilisateur tandis qu'ils voient l'agent d'interface comme serveur d'un humain dans l'utilisation des outils. En conséquence, l'agent d'interface doit avoir certaines caractéristiques d'intelligence pour mieux servir les humains. C'est ce qui a fait que des aspects de l'intelligence artificielle classique (interfaces en langage naturel, assistance intelligente, compréhension, apprentissage, ...) dominent la recherche dans le domaine des agents d'interface. D'un autre coté, le développement des systèmes intégrant l'intervention des humains comme partie de ces systèmes n'est pas basé sur une approche cohérente. La vision précédente des agents d'interface ne permet pas de fonder une coopération entre les humains et les systèmes artificiels de manière cohérente.

Interfaces utilisateur et agents

Nous pouvons classer les solutions combinant des agents et des interfaces comme suit :

1. Agent d'interface pour des applications classiques,
2. Interface classiques pour des SMA,
3. Agents d'interface pour des SMA.

Dans le cas des agents d'interface pour des applications classiques, l'humain interagit avec le système logiciel tel qu'il le fait habituellement. L'agent d'interface observe toutes ses actions tout en le conseillant et en lui proposant des solutions s'il rencontre des difficultés particulières. Il se comporte en tant qu'assistant qui doit être intelligent, convivial, comprenant, apprenant, communiquant en langage naturel, etc. Cette approche est celle proposée par Maes [Metral 93, Maes 93], améliorée par la collaboration de plusieurs agents d'interface pour apprendre de plusieurs utilisateurs par Lashkari [Lashkari 94] et en introduisant l'autonomie de l'agent d'interface par Lieberman [Lieberman 98, Lieberman 97, Lieberman 01] et continuée, ensuite, par Rich et ses collègues [Rich 96, Rich 97a, Rich 98, Rich 97b, Lesh 98]. Dans cette vision, le système logiciel est hétérogène, c'est-à-dire, il comporte des composants basés sur la technologie agent et des composants classiques. L'adaptation des agents d'interface aux composants classiques fait qu'ils ne satisfassent pas les caractéristiques minimales des agents.

Dans le cas des interfaces classiques pour des SMA, l'humain interagit directement avec les agents du SMA à travers une interface classique. C'est l'approche suivie par les développeurs des SMA dont les interfaces ne constituent pas leur souci. Comme dans l'approche précédente, le rôle de ces interfaces est d'aider l'humain dans l'utilisation des divers agents. Généralement, ces approches pour les interfaces sont orientées objet. Dans cette dernière, on présente aux humains des objets matérialisés par des icônes représentant les objets du système logiciel accessibles aux humains. L'humain choisit des icônes et invoque des méthodes des objets qui lui y sont accessibles en utilisant un menu global ou contextuel. Les agents sont alors vus comme des objets. Ce qui n'est pas cohérent avec le concept d'agent. De plus, le système logiciel ne peut pas profiter explicitement et de façon cohérente des capacités intellectuelles des humains et/ou de leurs actions sur l'environnement physique. En outre, on ne peut pas contrôler la coopération entre les humains puisqu'on les considère comme des utilisateurs. Ces deux aspects sont des besoins de beaucoup d'application et en particulier des processus logiciels tel que expliqué dans l'introduction.

Le cas des agents d'interface pour les SMA est une intégration de l'humain dans le SMA. L'humain est considéré comme étant un composant du système (un agent). Dans ce cas, nous distinguons deux environnements :

1. Logiciel,
2. Physique.

L'agent humain et un logiciel particulier F sont associés pour constituer un agent. F enveloppe l'humain et le représente dans l'environnement logiciel de sorte qu'il emploie les capacités de cet humain pour servir des agents logiciels. De son coté, l'humain enveloppe F et le représente dans l'environnement physique. Cette approche est qualifiée d'approche basée coopération.

Approche basée coopération pour les agents d'interface

L'approche basée coopération considère abstraitement un humain comme un agent H. Conceptuellement, cette approche associe à H un logiciel F qui détient l'objectif de H. Le logiciel F est qualifié d'agent d'interface. Cependant, l'agent d'interface F n'a ni les capacités intellectuelles de l'humain ni celles de son exploitation des ressources de l'environnement physique. L'approche proposée complète cette insuffisance par une coopération entre le logiciel F et l'humain. L'agent H sera alors composé du logiciel F compléter par les capacités de l'humain.

L'agent H est considéré comme un élément d'un certain SMA où les caractéristiques des agents de ce SMA seront développées selon l'approche utilisée pour le développement des autres agents du SMA. Une telle approche peut être une utilisation d'une méthodologie de développement multi-agent telle que MASA-Method. Une telle méthodologie permettra le développement des agents du SMA comprenant les agents du genre H. La dérivation des caractéristiques exigées de H et puis de ceux de F sera alors faite en suivant une méthodologie de développement multi-agent.

Dans l'approche basée coopération, le plan d'action doit être explicite. Ce plan constitue la tâche T que l'agent H doit accomplir en tant qu'élément d'un SMA. Il est alors déterminé pendant la conception du SMA de sorte que le fonctionnement de ce SMA sera cohérent. La tâche T sera détenue par l'agent d'interface F qui assurera sa mise en œuvre. F doit interpréter le plan d'action. Une action de ce plan peut être :

1. Répondre aux requêtes des autres agents logiciels (y compris d'autres agents d'interface) demandant l'exécution d'une certaine procédure du savoir-faire de l'humain. Les réponses à ces requêtes sont faites en soumettant ces requêtes à l'humain, la récupération de la réponse de cet humain et son envoi à l'agent de la

requête. Ces requêtes seront reformulées par l'agent d'interface sous une forme compréhensible par l'humain,

2. Demander à l'humain d'exécuter des procédures de son savoir-faire destinées à lui,

3. Demander à d'autres agents logiciels (y compris d'autres agents d'interface représentant des humains) d'exécuter les actions qui sont prévues pour eux.

Les avantages de cette approche sont nombreux tels que :

1. L'utilisation des techniques de coopération des SMA pour dériver les caractéristiques nécessaires de l'agent H et, par conséquent, ceux des agents d'interface F,

2. Faire profiter les SMA des capacités intellectuelles des humains et de leurs capacités d'agir sur des environnements physiques,

3. La gestion de la coopération entre des humains à travers un réseau.

Son inconvénient réside dans le besoin d'une planification préalable. Cependant, d'une certaine manière, cet inconvénient peut être considéré comme étant également un avantage puisqu'il assure un développement rigoureux des SMA.

Mise en œuvre des agents d'interface

Les éléments d'un agent d'interface seront identifiés comme suit :

1. Les senseurs et les effecteurs de l'environnement : Ils sont les moyens de l'interface d'interaction de l'agent d'interface avec l'humain tels que le clavier et l'écran. L'agent d'interface utilise ces moyens d'interaction, non pas pour percevoir et agir directement sur l'environnement mais, pour être informé par l'humain de l'état de environnement de ce dernier (l'humain) et pour transmettre à l'humain la nature des actions que le système a planifié pour que l'humain agit sur cet environnement,

2. Les senseurs, les effecteurs et les protocoles de communication : Ils sont identiques à ceux des autres agents utilisés pour communiquer entre les agents logiciels d'un SMA,

3. L'état : C'est une abstraction de l'état mental de l'humain. Nous donnons plus de détails sur l'état de l'agent d'interface dans les paragraphes suivants,

4. Savoir-faire : Il contient toutes les procédures nécessaires de l'humain pour accomplir sa mission. Cependant, ces procédures ne sont pas exactement ceux que l'humain emploie. Nous donnons plus de détails sur le savoir-faire de l'agent d'interface dans les paragraphes suivants,

5. La tâche : Elle est celle qui sera attribuée à l'agent H comme celles de n'importe quel autre agent,

6. Le contrôle : C'est celui d'un agent logiciel quelconque.

Deux solutions peuvent être considérées pour l'état d'un agent d'interface :

1. Mémorisation de l'état par l'agent d'interface : La difficulté de cette solution réside dans le fait que l'agent d'interface ne peut pas mémoriser tous les types d'informations qu'un agent humain en aura besoin,

2. Détention de l'état par l'humain : La difficulté de cette solution réside dans le fait que les agents d'interface ne peuvent pas aider suffisamment l'humain dans la mémorisation et la présentation de certaines données et résultats.

Une troisième solution consiste en une combinaison des deux solutions précédentes. Dans ce cas, l'information dont la représentation est acceptable (du point de vu coût, faisabilité, ...) sera mémorisée par l'agent d'interface. Toute autre information sera mémorisée par l'humain. La décision sur choix des informations à attribuer à l'agent d'interface est laissée au concepteur des systèmes multi-agent particuliers.

Une procédure du savoir-faire d'un agent d'interface doit permettre de :

1. Communiquer le nom de la procédure à l'humain,

2. Si la procédure du savoir-faire de l'humain nécessite des données, la procédure du savoir-faire de l'agent d'interface doit présenter ces données à l'humain,

3. Recevoir de l'humain les résultats de l'exécution de cette procédure.

Si les moyens de communication avec l'humain sont les moyens habituels (clavier, écran, souris, ...) l'affichage des données d'entrée et des résultats peut être complexe. De telles données peuvent être des textes, des graphes et/ou des images. La procédure de visualisation de telles données doit utiliser les outils appropriés permettant d'afficher ces données et/ou d'aider l'humain pour entrer les résultats. Dans MASA-Method, une procédure du savoir-faire d'un agent d'interface sera implémentée comme une méthode d'un objet comme expliqué à la section 3.1. Elle peut être décrite comme suit :

1. Afficher le nom de la procédure du savoir-faire que l'agent d'interface demande à l'humain de mettre en œuvre,

2. Afficher ses paramètres (données) s'il y en a. Ces données seront précisées par l'agent d'interface,

3. Récupérer les résultats (données) de sa mise en œuvre. Ces résultats seront donnés par l'humain.

3.3.3 Intégration des agents hétérogènes dans un système multi-agent

Les systèmes logiciels et les humains peuvent être considérés comme des agents. La tâche qui est difficile à accomplir est celle de l'intégration de ces agents hétérogènes de sorte qu'ils puissent coopérer de façon cohérente. Dans MASA-Method, ceci a été surmonté en exploitant l'approche basée coopération pour le développement des agents d'interface. Un des buts de l'agent d'interface est d'établir une relation communication saine entre l'environnement logiciel et les humains. En outre, ces agents d'interface sont des systèmes logiciels qui peuvent communiquer avec les autres agents logiciels incluant d'autres agents d'interface. La communication entre les agents d'interface implique des humains ou chaque agent d'interface est associé à un rôle humain.

En utilisant des agents d'interface, une communication peut être établie entre les agents logiciel et les humains ainsi qu'entre les humains eux-mêmes à travers les agents d'interface. Puisque les relations de communication sont la seule relation directe qui peut exister entre les agents d'un système multi-agent, le résultat de l'association des agents par de telles communications peut former un système multi-agent intégrant des agents hétérogènes. Ces relations de communication sont très utiles entre les humains qui ne peuvent pas communiquer directement tels que ceux qui sont situés dans des espaces distants. Parfois, ces relations sont aussi utiles pour des humains proches, tel qu'entre les joueurs d'un jeu vidéo et la communication de volumes importants d'informations. Cependant, dans beaucoup de situations, les humains communiquent beaucoup mieux de façon directement. Il est alors de la responsabilité du développeur du SMA de choisir la manière la plus appropriée de faire communiquer deux agents humains qui font partie d'un SMA.

Dans un SMA intégrant des agents hétérogènes, on peut charger les agents logiciel de :

1. L'exécution des tâches de traitement automatique de données tels les calculs, le stockage de données, la recherche d'informations,

2. Prendre certaines décisions.

Les humains seront alors chargés de :

1. L'accomplissement de traitements de tâches intelligentes,
2. L'accomplissement de certaines actions sur l'environnement physique décidées par eux-mêmes (les humains) et/ou par des agents logiciel.

3.4 Implémentation des modèles multi-agent

Les développeurs de SMA ne disposent pas de langages de programmation multi-agent bien acceptés pour implémenter les concepts des modèles de SMA. En conséquence, plusieurs développeurs implémentent les SMA empiriquement dans des paradigmes classiques. Il y a alors un changement du paradigme des SMA vers des paradigmes classiques. Une telle pratique peut amener à des systèmes incohérents. Il est alors nécessaire d'assurer ce changement de paradigme méthodologiquement. Un tel changement doit permettre, au moins, d'aboutir à des systèmes cohérents.

L'implémentation des systèmes multi-agent est possible dans différents paradigmes. C'est le cas, par exemple, de l'utilisation de modèles de systèmes à base de connaissance tel que décrit dans la méthodologie MAS-CommonKADS [Iglesias 96]. Cependant, la plupart des développeurs préfèrent des implémentations orientées objet. Cette préférence est la conséquence de la bonne maîtrise des concepts des systèmes orientés objet. De plus, les paradigmes des systèmes orientés objet sont bien admis et largement utilisés dans l'industrie logicielle. Leurs langages de programmation sont disponibles et supportés par des outils et des environnements de développement largement diffusés. Il est alors intéressant d'implémenter les modèles multi-agent méthodologiquement en utilisant le paradigme orienté objet.

Certains travaux, tels que [Odell 00], se concentrent sur l'implémentation méthodologique de certains concepts multi-agent dans des paradigmes orientés objet sans considérer l'identification méthodologique de ces concepts. D'autres proposent des plateformes, tel que DIMA [Guessoum 99], qui perettent aux développeurs d'implémenter leurs conceptions. Par contre, d'autres travaux, tel que la méthodologie Gaia [Zambonelli 03], considèrent l'identification méthodologique d'abstractions sans considérer leurs implémentations. En l'absence d'un consensus sur des concepts multi-agent, l'implémentation des abstractions identifiées par les travaux de modélisation par des travaux d'implémentation des concepts est problématique. Il est alors important de considérer l'implémentation méthodologique dans le paradigme orienté objet dans la méthodologie permettant la description de ces abstractions. C'est ce que nous avons fait dans MASA-Method.

Comme discuté au chapitre 2, un système classique peut être considéré comme un agent sans autonomie fonctionnelle. Il est réduit à un savoir-faire qui peut être utilisé par un agent utilisateur. Ces utilisations sert à réaliser des buts non spécifiés pour ce système classique. Elles ne dépendent que de l'agent utilisateur. Cependant, les systèmes agents sont fonctionnellement autonomes. Cette autonomie fonctionnelle exige, principalement, la définition d'une tâche pour ce système classique que son contrôle peut interpréter pour le rendre un agent. Ce qui amène à la modification aussi des protocoles de communication pour qu'ils soient basés sur les actes de langage. Un agent peut alors être considéré comme un système orienté-objet et, par conséquent, un SMA comme un ensemble de systèmes orientés objet communicants.

L'approche décrite dans cette section demeure valide pour des langages de programmation orientés objet, autres que Java, avec des caractéristiques spécifiques. Dans MASA-Method, nous avons utilisé le langage de programmation Java parce qu'il a certaines caractéristiques intéressantes. Java permet la déclaration d'objets publics à l'intérieur d'une classe. Cette permission est contraire au principe d'encapsulation du modèle orienté objet. Cependant, elle nous permet de considérer un objet comme système orienté objet. Sans considérer ses méthodes, un objet O comprenant des objets publics est constitué des objets en interaction. Ces objets peuvent être accédés de l'extérieur de l'objet O. Si O est un objet thread, il est alors doté d'un flot de contrôle indépendant. De tels objets peuvent alors être considérés comme des systèmes orientés objet.

Un objet thread en Java avec des objets publics peut être considéré comme système orienté objet. Le thread constitue son contrôle qui est un contrôle logiciel. Les thread de Java sont associés à une méthode spéciale appelée Run. Cette méthode sera utilisée en tant que tâche de l'agent. La communication entre les agents sera faite par les actes de langage. Ces protocoles seront implémentés en utilisant le protocole d'invocation de méthodes à distance (RMI). Les agents peuvent alors être situés dans des nœuds d'un réseau et le SMA sera un système orienté objet distribué. Dans ces conditions, un objet thread peut être considéré comme un agent logiciel. L'association d'objets thread avec les caractéristiques d'un agent peut être faite comme suivent :

1. Le contrôle est procédural et logiciel. Il est représenté par un thread,

2. Un senseur (usuel ou de communication) est un objet S qui peut être modifié de l'extérieur de l'agent. Le contrôle de l'agent peut consulter S pour s'informer de l'état d'une région spéciale de l'environnement. L'objet S comporte un état qui dépend de la nature du senseur utilisé. Il comporte aussi deux méthodes. La première permet de modifier l'objet S de l'extérieur. La seconde permet au contrôle de l'agent de consulter l'état de S. Dans le cas d'un senseur de

communication, c'est un autre agent qui modifiera cet objet S,

3. Un effecteur (usuel ou de communication) est un objet E que le contrôle de l'agent peut utiliser pour modifier l'environnement. Il comporte un état qui dépend de la nature de l'effecteur utilisé. Il comporte aussi deux méthodes. La première permet au contrôle de l'agent de modifier E. La seconde permet aux éléments de l'environnement de consulter l'état de E. Dans le cas d'un effecteur de communication, c'est un autre agent qui le consultera,

4. Le savoir-faire est vu comme dépendant des objets à manipuler. En effet, ce qui est qualifié de savoir-faire est l'ensemble des méthodes de ces objets,

5. L'état de l'agent est constitué des objets du thread,

6. Une procédure du protocole de communication est une méthode d'un objet effecteur (pour l'envoi de messages) ou d'un objet senseur (pour la réception de messages),

7. La tâche individuelle sera implémentée comme un constructeur d'un objet virtuel, c'est-à-dire, ne comportant que ce constructeur,

8. Protocole de synchronisation: La synchronisation dans MASA-Method est implicitement incluse dans le CPN de la tâche de l'organisation. La dérivation systématique des tâches individuelles des rôles et leur attribution aux agents préserve la synchronisation.

L'implémentation d'un modèle multi-agent se compose d'une implémentation des modèles des agents (y compris les agents logiciel et d'interface) et d'un module démarrage des différents agents. Ce système de démarrage est décrit dans le chapitre 4.

3.5 Conclusion

Selon la classification des méthodologies multi-agent décrite dans la section 2.5 du chapitre 2, MASA-Method est une méthodologie complète mixte. Son expérimentation a commencé il y a cinq ans. Elle a été appliquée dans neuf applications incluant :

1. Le processus de développement de logiciels (six applications),

2. Simulation multi-agent de la coopération de robots mobiles autonomes (une application),

3. Système multi-agent avec des agents hétérogènes (robot, humain et logiciel) (une application),

4. Système multi-agent comprenant des agents mobiles (une application).

L'utilisation de MASA-Method dans le développement de ces applications a permis de proposer de nouvelles approches. De telles approches concernent, particulièrement, les agents d'interface [Lahlouhi 02b]. Ceci concerne aussi les agents mobiles, les sociétés hétérogènes et le processus logiciel. Ces applications ont permis également de réviser, d'améliorer et d'enrichir MASA-Method.

En plus des rôles et des relations de communication entre eux, les modèles des organisations de MASA-Method incluent les aspects de leurs démarrages et de la détection de la fin de leurs mises en œuvre. L'utilisation des CPN a permis de valider notre proposition et à rendre opérationnel les aspects organisationnels. Par rapport aux travaux sur les organisations des SMA, les aspects organisationnels dans les organisations sont nouveaux. Ils présentent bien notre vision du développement des organisations comme une ingénierie.

La dérivation à partir des organisations des modèles multi-agent est faite de façon systématique. De tels modèles permettent la mise en œuvre de ces organisations. L'intégration des agents réellement hétérogènes a été faite en utilisant des agents d'interface qui sont développés en utilisant l'approche basée coopération. Enfin, l'implémentation des modèles de SMA est faite elle aussi de façon systématique. La considération de l'agent comme un système, son association à un objet thread java et l'implémentation des modèles de SMA en tant que multi-système distribué orienté objet est faite aussi pour la première fois.

Enfin, les aspects de l'utilisation pratiques de MASA-Method sont décrits dans le chapitre 4. Ces aspects incluent des tables permettant la description des différents modèles et les processus permettant de les dériver.

Le chapitre 5 montrera l'utilisation de MASA-Method dans le développement d'un modèle multi-agent d'un environnement de développement de logiciels.

Chapitre 4

Outils pratiques pour l'utilisation de MASA-Method

Une méthodologie de développement de systèmes se base, fondamentalement, sur deux composantes : un méta-modèle et un processus méthodologique. Ces deux composantes sont nécessaires pour expliquer le bien fondé de la méthodologie. Sans ces deux composantes, on ne peut pas qualifier un travail de méthodologie de développement de systèmes. Dans les chapitres 2 et 3, nous avons détaillé les aspects relatifs au méta-modèle et au processus méthodologique de la méthodologie MASA-Method. Cependant, beaucoup de leurs détails ne sont pas d'une grande utilité pour un développeur de SMA.

Le formalisme est une autre composante importante d'une méthodologie. Au chapitre 3, nous avons expliqué l'utilité des CPN et des CPN synchronisés qui sont utilisés comme formalisme pour MASA-Method. Nous avons aussi explicité certains détails de leur utilisation dans le développement des SMA. Un formalisme bien fondé d'une méthodologie est d'une grande utilité pour les développeurs. Cependant, il est insuffisant, lui seul, d'assurer le succès d'une méthodologie. Une fois qu'une méthodologie est choisie, le développeur préfère de ne pas se préoccuper des justifications du bien fondé de cette méthodologie. Il préfère se concentrer sur les détails des conceptions. Les développeurs s'intéressent, plus particulièrement, aux outils d'utilisation pratique de la méthodologie. Ce chapitre sera consacré aux outils d'utilisation pratique de MASA-Method. Ces outils ont été classés en de deux parties :

1. Ceux décrivant les aspects de modélisation,
2. Ceux décrivant les processus méthodologiques.

Les outils de développement associent des moyens de description et des processus de développement de ces descriptions. Les moyens de description montrent les aspects

qu'il est nécessaire de décrire pour représenter les différents raffinements qui aboutiront aux modèles de SMA. Ils sont présentés sous forme de tables. Chaque table comporte les éléments à décrire.

Les processus de développement des descriptions comportent les activités et leurs successions permettant de développer les éléments de description représentés dans les tables. Ils proposent aussi des processus de dérivation de certaines descriptions.

Chacune des sections suivantes représente une étape de développement dans le processus global de MASA-Method pour le développement des SMA. Ce processus est le suivant :

1. Développement des tâches de systèmes,
2. Développement des modèles d'organisations,
3. Développement des modèles de SMA,
4. Implémentation des modèles de SMA.

Chacune des étapes (sections) comporte deux parties :

1. Moyens de description,
2. Processus méthodologique de développement.

L'étape d'implémentation comporte, en plus, une section décrivant l'implémentation de certains éléments des agents tels que les senseurs, les effecteurs, etc.

Les moyens de description sont décrits sous forme de tables. Chaque table comporte les éléments à inclure dans une description donnée. Elle est composée d'au moins six parties : Numéro, objet, identification, corps, légende et commentaires. Le numéro permet une identification pratique des tables. L'objet permet d'indiquer la nature de la table (Par exemple, la partie "System's task" de la Table 4.1-Page 75). L'identification comporte un identificateur. L'identificateur identifie l'objet en question (Par exemple, la partie "Identifier" de la Table 4.1-Page 75). Elle peut comporter aussi des relations avec d'autres descriptions (tables) (Par exemple, "Mother organization" de la Table 4.2-Page 80). Le corps contient la description de l'objet. Il peut comporter lui aussi plusieurs parties selon la description de l'objet. La légende comporte une description des symboles utilisés dans les différentes descriptions de la table. Les commentaires sont destinés à commenter les différent éléments de la description.

Un processus méthodologique comporte quatre parties : Entrées, Sorties, Remarques et Processus de dérivation des sorties à partir des entrées.

Table 4.1: Description d'un modèle de tâches d'un système

1	Modèle de la tâche d'un système		
Identificateur			
Description Informelle			
Description Formelle	Modèle de l'environnement		
	Modèle de la tâche		
	Symbole	Description	
Places			
Types			
Transitions			
Fonctions d'arcs			
Gardes			
Tâches élémentaires	Identificateur	Description	
Légende	Acronyme	Description	
Commentaires			

Le reste de ce chapitre est organisé en cinq sections. La section 4.1 décrit les aspects relatifs à la description des tâches des systèmes. La section 4.2 décrit les aspects relatifs au développement des organisations. La section 4.3 décrit les aspects relatifs au développement des modèles de SMA. La section 4.4 décrit les aspects relatifs à l'implémentation des SMA. La section 4.5 conclut ce chapitre.

4.1 Développement des tâches de systèmes

4.1.1 Moyens de description

La description des tâches de systèmes comporte une seule table (Table 4.1-Page 75). Cette table comporte :

1. Une description informelle (en langage naturel) de la tâche du système. Le plus souvent, la description informelle n'est pas précise. Elle permet de donner une première idée sur la description formelle,

2. Une description formelle de la tâche du système sous forme d'un CPN sous forme d'une synchronisation de tâches (plus) élémentaire,

3. Une description des éléments du CPN. Pour les places marquées initialement, nous avons inclu ce marquage dans leur description,

4. Une description des tâches élémentaires.

4.1.2 Processus de développement

Le développement de la tâche d'un système comporte (Table 4.1-Page 75) les éléments suivants :

Entrées: Objectif Obj du système à développer (Non spécifiée ici)

Sorties: Expression informelle de la tâche ST du système permettant d'atteindre l'objectif Obj et expression formelle de ST sous forme d'un CPN (Table 4.1-Page 75)

Processus:

1. **Décrire informellement la tâche du système ST**

   Entrées: Objectif Obj du système à développer
   Sorties: Expression informelle de la tâche du système ST,
   Remarque : Se baser les notions expliquées dans la section 3.1 du chapitre 3.
   Processus: Empirique.

2 **Formaliser la tâche du système ST**

   Entrées: Expression informelle de la tâche du système ST,
   Sorties: Expression formelle de la tâche du système ST sous forme d'un CPN,
   Remarque : Tâche empirique. Voir la définition des tâches de système section 3.1 du chapitre 3,
   Processus: Empirique.

3 **Vérifier la consistance de l'expression formelle de la tâche du système ST**

   Entrées: Expression formelle de la tâche du système ST,
   Sorties: Confirmation de la consistance de l'expression formelle de la tâche du système ST,
   Remarque : Utiliser les outils d'analyse des CPN
   Processus: Empirique.

4 **Décrire les tâches élémentaires référencées dans la tâche du système ST**

Entrées: Expression formelle de la tâche du système ST,
Sorties: Ensemble de tâches élémentaires avec leurs descriptions associées,
Remarque :
Processus: Empirique.

4.2 Développement des modèles d'organisations

4.2.1 Moyens de description

Le développement des modèles d'organisations comporte les table suivantes :

1. Description d'un modèle de tâches d'une organisation élémentaire (Table 4.2-Page 80),
2. Description d'un modèle de tâches d'une organisation compose (Table 4.3-Page 80),
3. Description d'un modèle d'organisations élémentaires (Table 4.4-Page 81),
4. Description d'un modèle de rôles (Table 4.5-Page 81),
5. Description d'un modèle d'une organisation composée (Table 4.6-Page 82).

La table de description de la tâche d'une organisation élémentaire (Table 4.2-Page 80) comporte :

1. Un identificateur du modèle de la tâche de l'organisation élémentaire,
2. Une référence au modèle de tâches du système que ce modèle de tâches d'organisation implémente,
3. Un modèle de l'environnement qui est le même que celui de la tâche du système,
4. Une référence vers la description (formelle) de la tâche de cette organisation,
5. Une description informelle des rôles impliqués dans la tâche de l'organisation.

La table de description de la tâche d'une organisation composée (Table 4.3-Page 80) comporte :

1. Un identificateur du modèle de la tâche de l'organisation élémentaire,

2. Une référence vers le modèle de tâches du système que ce modèle de tâches d'organisation implémente,
3. Un modèle de l'environnement qui est le même que celui de la tâche du système,
4. Une référence vers la description (formelle) de la tâche de cette organisation,
5. Une descriptions informelles des différentes sous-organisations impliquées dans l'organisation composée.

La table de description d'un modèle d'une organisation élémentaire (Table 4.4-Page 81) comporte :

1. Un identificateur du modèle d'organisations,
2. Une organisation mère si cette organisation élémentaire est une sous-organisation,
3. Une référence vers le modèle de la description (formelle) de la tâche de cette organisation,
4. Une référence vers le modèle de l'organisation substituée dans le cas où cette organisation est une substitution d'une organisation composée,
5. Une description des rôles sous forme de couples (Identificateur de rôle, Référence vers le modèle de ce rôle),
6. Une description de la relation de communication. La relation de communication est décrite de deux façons différentes. Représentation graphique qui est une représentation visuelle. L'expression analytique est décrite sous forme de couples (Rôle émetteur, Rôle récepteur). Elle permet des traitements et des descriptions des implémentations.

La table de description d'un modèle de rôles (Table 4.5-Page 81) comporte :

1. Un identificateur du modèle de rôles,
2. Une organisation de ce rôle,
3. Un modèle de l'environnement du rôle. Ce modèle sera dérivé de celui de la tâche de l'organisation,
4. Une description (formelle) de la tâche du rôle. Cette tâche sera dérivée de la tâche de l'organisation,

5. Un savoir-faire du rôle : Les identificateurs des procédures du savoir-faire ainsi que leurs paramètres seront dérivés de la tâche du rôle. Cependant, leurs descriptions seront fait par le développeur.

La table de description d'une organisation composée (Table 4.6-Page 82) comporte :

1. Un identificateur de l'organisation composée,
2. Une référence ver l'organisation mère, si cette organisation est une sous-organisation,
3. Une référence vers la tâche de cette organisation,
4. Une référence vers l'organisation substituée, si cette organisation est une substitution d'une organisation composée,
5. Des représentants : Ils ne seront présents que si cette organisation composée est une sous organisations d'une autre organisation composée,
6. Une description des sous organisations sous forme de triplets (Identificateur de la sous organisation, Les représentants de la sous organisation, Référence vers le modèle de cette sous organisation),
7. Une description de la relation de communication. Cette relation est décrite de deux façons différentes. La première est une représentation graphique visuelle. La seconde est une expression analytique qui décrite sous forme de couples (Représentant de la sous-organisation émettrice, Représentant de la sous-organisation réceptrice). La représentation analytique permet des traitements et des descriptions des implémentations.

4.2.2 Processus de développement

Le processus de développement des organisations peut être décrit comme suit :

Entrées : Expression formelle de la tâche du système ST (Table 4.1-Page 75)

Sorties : Un modèle d'organisations élémentaires O (Table 4.4-Page 81) permettant d'accomplir la tâche du système ST, sous forme d'un couple (ER, RC) où :

1. ER: Ensemble de rôles,
2. RC: Relation de communication entres rôles

Processus :

Table 4.2: Description d'un modèle de tâches d'une organisation élémentaire ou de la substitution de la tâche d'une organisation composée

2	Modèle de la tâche d'une organisation élémentaire		
Identificateur			
Tâche du système			
Tâche substituée			
Description de la tâche	Modèle de l'environnement		
	Modèle de la tâche		
	Rôles	Identificateur	Description
Légende	Acronyme	Description	
Commentaires			

Table 4.3: Description d'un modèle de tâches d'une organisation composée

3	Modèle de la tâche d'une organisation composée		
Identificateur de la tâche			
Tâche			
Tâche substituée			
Description de la tâche	Modèle de l'environnement		
	Modèle de la tâche		
	Sous-organisations	Identificateur	Description
Légende	Acronyme	Description	
Commentaires			

Table 4.4: Description d'un modèle d'une organisation élémentaire ou d'une substitution d'une organisation composée

4	Modèle d'une organisation élémentaire		
Identificateur			
Organisation mère			
Tâche			
Organisation substituée			
Rôles	Identificateur	Modèle	
Relation de communication	Graphique	Analytique	
		Emetteur	Récepteur
Légende	Identificateur	Description	
Commentaires			

Table 4.5: Description d'un modèle de rôles

5	Modèle de rôles		
Identificateur			
Organisation			
Modèle de l'environnement			
Tâche			
Savoir-faire	Identificateur	Paramètres	Description
Légende	Symbole	Description	
Commentaires			

Table 4.6: Description d'un modèle d'une organisation composée

6	Modèle d'une organisation composée		
Identificateur			
Organisation mère			
Tâche			
Organisation substituée			
Sous-organisations	Identifier	Représentants	Modèle
Communication relation	Graphique	Analytique	
		Emetteur	Récepteur
Légende	Symbole	Description	
Commentaires			

1. **Traiter les organisations composées**

   Entrées : Modèle de tâches de systèmes (Table 4.1-Page 75),

   Sorties : Organisation élémentaire permettant d'accomplir la tâche du système (Table 4.4-Page 81),

   Remarque:

   Processus : Utiliser le processus décrit dans la sous sous-section 4.2.2.

2. **Traiter les organisations élémentaires**

   Entrées : Modèle de tâches de systèmes (Table 4.1-Page 75),

   Sorties : Organisation élémentaire permettant d'accomplir la tâche du système (Table 4.4-Page 81),

   Remarque: ,

   Processus : Utiliser le processus décrit dans la sous sous-section 4.2.2.

3. **Traiter les modèles de rôles**

   Entrées : Organisation composée (incluant des sous-organisations),

   Sorties : Organisation élémentaire,

   Remarque: ,

   Processus : Utiliser le processus décrit dans la sous sous-section 4.2.2.

Processus de traitement des organisations composées

Entrées : Modèle de tâches de systèmes (Table 4.1-Page 75),

Sorties : Organisation élémentaire permettant d'accomplir la tâche du système (Table 4.4-Page 81),

Remarque: Si les sous-organisations sont aussi des organisations composées, il est nécessaire d'appliquer à chacune d'elles le processus de développement des organisations composées.

Processus:

1. **Raffiner le modèle de tâches de systèmes ST en un modèle de tâches d'organisations composées COT**

   Entrées : Expression formelle de la tâche du système ST (Table 4.1-Page 75),
   Sorties : Tâche de l'organisation COT (Table 4.3-Page 80),
   Remarques :
   Processus :

   1. (a) Ajouter les sous-organisations qui accompliront les tâches élémentaires de l'expression formelle de la tâche du système ST,
      (b) Décrire informellement chacune des sous-organisations.

2. **Spécifier les représentants de chacune des sous-organisations de cette organisation composée**

   Entrées : Expression formelle de la tâche de l'organisation COT (Table 4.2-Page 80),

   Sorties : Les représentants des sous-organisations de l'organisation composée (Table 4.6-Page 82),

   Remarques : Le choix des représentants dépend de l'objectif de l'organisation. Ce choix reste empirique.

   Processus : Empirique.

3. **Dériver les relations de communication entre les représentants des différentes sous-organisations**

   Entrées : Expression formelle de la tâche de l'organisation COT (Table 4.2-Page 80),

   Sorties : Relations de communication entre les représentants des sous-organisations (Table 4.6-Page 82),

Remarques : Les sous-organisations nécessitant des communications entre elles seront déterminées à partir de la tâche de l'organisation. Cependant, le choix des représentants des sous-organisations à mettre en relations de communication sera à la charge du développeur.

Processus : Empirique.

4. **Développer les sous organisations référencées dans la tâche de l'organisation composée**

Entrées : Les sous organisations de l'organisation sous développement,

Sorties : Description des sous organisations,

Remarques: La création d'une sous organisation suit, récursivement, le processus de la création des organisations,

Processus :

(a) **Dans le cas des sous-organisations élémentaires,** développer chacune des sous-organisations selon le processus de traitement des organisations élémentaires de la sous sous-section 4.2.2.

(b) **Dans le cas des sous-organisations composées,** pour chacune des sous-organisations :

i. Considérer sa tâche comme un objectif d'un système,

ii. De façon récursive, recommencer le processus de développement des SMA à partir de son début, i.e., à partir du développement de la tâche du système.

5. **Substituer les sous organisations**

Entrées : Modèle d'une organisation composée CO (Table 4.6-Page 82), Ensemble d'organisations élémentaires représentant les sous-organisations de l'organisation composée,

Sorties : Organisation élémentaire,

Remarques:

Processus : Appliquer l'algorithme donné sur la Figure 3.16-Page 61 du chapitre 3.

Processus de traitement des organisations élémentaires

Entrées : Organisation composée (incluant des sous-organisations),

Sorties : Organisation élémentaire,

Remarque: Si les sous-organisations sont elles aussi des organisations composées, il est nécessaire d'appliquer à chacune d'elles le processus de développement des organisations composées,

Processus : .

1. **Raffiner la tâche du système ST en une tâche de l'organisation OT**

   Entrées : Expression formelle de la tâche du système ST (Table 4.1-Page 75),
   Sorties : Tâche de l'organisation OT (Table 4.2-Page 80),
   Processus :

   (a) Ajouter les rôles qui accompliront les tâches élémentaires de l'expression formelle de la tâche du système ST,

   (b) Décrire informellement chacun des rôles.

2. **Dériver les relations de communications de l'organisation à partir de la tâche de l'organisation OT**

Entrées : Expression formelle de la tâche de l'organisation OT (Table 4.2-Page 80),

Sorties : Les relations de communications de l'organisation (Table 4.4-Page 81),
Remarques :
Processus : Utiliser la méthode de dérivation des relations de communication à partir de la tâche de l'organisation. Cette méthode est décrite dans la section 3.2.1 du chapitre 3.

Processus de traitement des modèles de rôles

1. **Dériver les tâches des rôles**

   Entrées : Tâche de l'organisation OT,
   Sorties : Association {(R, T) / R : Rôle et T : Tâche} (Table 4.5-Page 81)
   Remarques :
   Processus : Utiliser la méthode décrite dans la sous sous-section 3.2.1 du chapitre 3.

2. **Déterminer les savoirs faire nécessaires des rôles**

Entrées : Tâche de l'organisation OT,
Sorties : RNK = {rNK / rNK = (r, NK) où r est un rôle et NK son savoir-faire nécessaire}

Table 4.7: Description d'un modèle de systèmes multi-agent

7	Modèle d'un système multi-agent		
Identificateur			
Organisation			
Attribution de rôles	Identificateur de l'agent	Rôles	
Liens de communication	Graphique	Analytique	
		Emetteur	Récepteur
Légende	Acronyme	Description	
Commentaires			

Table 4.8: Description d'un modèle d'agents

8	Modèle d'agents	
Identificateur		
Modèle de systèmes Multi-agent		
Contrôle		
Rôles		
Communications	Senseurs	
	Effecteurs	
	Protocoles	
Relation avec l'environnement	Objets	
	Savoir-faire	
	Senseurs	
	Effecteurs	
Légende	Acronyme	Description
Commentaires		

Remarques :

Processus :

Pour chaque rôle :

(a) Explorer l'expression formelle de sa tâche. Les identificateurs des procédures du savoir-faire de ce rôle sont ceux attachés aux transitions du CPN de la tâche du rôle. Collecter les identificateurs des procédures du savoir-faire et leurs entrées/sorties associées,

(b) Dériver le modèle de l'environnement du rôle. Celui-ci inclut les modèles d'objets référencés dans la tâche du rôle,

(c) Les entrées (resp. les sorties) d'une procédure P donnée du savoir-faire sont les fonctions des arcs d'entrées (resp. de sorties) attachés aux transitions de l'invocation de cette procédure P,

(d) Décrire chaque procédure du savoir-faire avec plus de détails de façon informelle.

4.3 Développement des modèles de systèmes multi-agent

4.3.1 Moyens de description

La table de description d'un modèle de SMA (Table 4.7-Page 86) comporte, en plus de la légende et des commentaires :

1. Un identificateur du modèle de SMA,
2. Une référence vers le modèle de l'organisation que ce modèle de SMA va implémenter.
3. Une attribution des rôles : C'est un ensemble d'association de couple (Agent A, Ensemble de rôles SR). SR est l'ensemble de rôles que l'agent A va assumer,
4. Des liens de communication : Précise les liens de communication entre les agents à établir dans une implémentation du modèle de SMA. Ces liens sont présentés de deux façons différentes : La forme graphique et la forme analytique.

La table de description d'un modèle d'agents (Table 4.8-Page 86) comporte :

1. Un identificateur du modèle d'agents,
2. Le modèle multi-agent auquel ce modèle d'agents appartient,
3. Le contrôle de l'agent qui peut être intelligent, déclaratif ou procédural,

4. Les rôles que cet agent va assumer. L'ensemble des tâches de ces rôles constitue la tâche du modèle d'agents. Les rôles doivent être énumérés selon leur ordre de priorité. Cet ordre sera utilisé par l'agent pour interpréter les tâches des rôles. L'agent ne passera de l'interprétation de la tâche d'un rôle R1 à celle d'un autre rôle R2 moins prioritaire que s'il n'a rien à faire dans la tâche de R1,

5. Les aspects de communication incluant un ensemble de senseurs, un ensemble d'effecteurs et un ensemble de protocoles de communication,

6. La relation avec l'environnement incluant le modèle de l'environnement, le savoir-faire de l'agent, un ensemble de senseurs et un ensemble d'effecteurs. Le modèle de l'environnement est constitué d'un ensemble de modèles d'objets. Le savoir-faire est un ensemble de procédure utilisant les senseurs et les effecteurs de l'agent relatifs à l'environnement pour accomplir des objectifs particuliers.

4.3.2 Processus de développement

Le processus de développement de SMA peut être décrit comme suit :

Entrées: Modèle d'une organisation, Agents à réutiliser,
Sorties: Un ensemble de modèles d'agents communicants,
Remarques :
Processus :
Le processus de développement des SMA consiste en deux sous processus :

1. Déterminer la relation du modèle de SMA avec l'organisation,

   Entrées: Modèle d'une organisation,
   Sorties: Un modèle de SMA,
   Remarques :
   Processus : Utiliser le processus de la sous sous-section 4.3.2

2. Dériver les modèles des agents.

   Entrées: Modèle de SMA,
   Sorties: Un ensemble de modèles d'agents communicants,
   Remarques :
   Processus : Utiliser le processus de la sous sous-section 4.3.2

Détermination de la relation du modèle de SMA avec l'organisation

Le processus de détermination de la relation du modèle de SMA avec l'organisation peut décrit comme suit :

Entrées : Modèle d'une organisation, Agents à réutiliser
Sorties : Modèle de SMA,
Processus :

1. **Identification des agents**

   Entrées : Modèle d'une organisation,
   Sorties : Agents à utiliser (OA), agents à développer (NA),
   Remarques : Tâche créatives,
   Processus : Identifier les agents qui accompliront l'organisation.

2 **Attribution des rôles**

   Entrées : Rôles de l'organisation, agents à réutiliser et agents à développer,
   Sorties : Association {(A, SR) / A: Agent et SR: Ensemble de rôles}
   Remarques: Tâche très intuitive,
   Processus :

3 **Dérivation de la relation de communication entre les agents**

   Entrées : Relation de communication de l'organisation et attribution rôles,
   Sorties : Relation de communication entre les gents du SMA,
   Remarques :
   Processus : Remplacer chaque rôle dans la relation de communication de l'organisation par l'identificateur de l'agent qui va l'assumer et fusionner les communications entre le même agent (Dérivation des liens de communication entre agents)

Dérivation des models d'agents du SMA

La dérivation des models d'agents du SMA peut être faite selon le processus suivant :

Entrées : Modèle de SMA,
Sorties : Un ensemble de modèles d'agents communicants,
Remarques :
Processus :

1. **Détermination des savoir-faire des agents**

   – Pour chaque agent, déterminer son savoir-faire nécessaire

Entrées : Ensemble de tous les rôles de l'organisation

Sorties : Savoirs faire nécessaires ANK des agents, ANK = {aNK / aNK = (a, NK) "a" est un agent et NK est son savoir-faire nécessaire}

Remarques:

Processus :

Etant donné l'ensemble de rôles ASR que l'agent A va assumer et l'ensemble des procédures RNK du savoir-faire des rôles, l'ensemble des procédures du savoir-faire ANK de l'agent A est définit comme suit :

ANK = U(r ∈ ASR) & ((r, NK) ∈ RNK) π ((r, NK)/NK)

où π est une opérateur de projection d'un vecteur sur une de ses composantes (NK, dans ce cas)

– Vérifier que le savoir-faire nécessaire des agents réutilisés est complet

Entrées : Savoir-faire nécessaire ANK de l'agent A, ANK = {(A, NK) / A est un agent et NK : Savoir-faire nécessaire}

Sorties : Ensemble NSKP = {(RA, NNK) / RA est un agent réutilisé et NNK est un ensemble de procédure nécessaires du savoir-faire nécessaire que RA n'a pas}

Remarques :

(a) Cette vérification peut être faite en comparant uniquement les identificateurs des procédures du savoir-faire de l'agent avec celles du savoir-faire nécessaire dérivé pour cet agent. Il peut aussi être fait par une analyse profonde des objectifs de ces procédures,

(a) Si l'ensemble NSKP n'est pas vide, nous devons recommencer tout le processus de développement du modèle de SMA en prenant en considération les agents à réutiliser qui n'ont pas le savoir-faire nécessaire

Processus :

Pour chaque agent réutilisé, vérifier que son savoir-faire inclut les procédures du savoir-faire nécessaire de tous les rôles qu'il va assumer.

2. **Identifier les senseurs et les effecteurs pour les communications de chaque agent**

Entrées : Relation de communication entre les agents du SMA,

Sorties : Association {(A, SS, SE)/ A : Agent, SS : Ensemble de senseurs de communication et SE : Ensemble des effecteurs communication},

Remarques:

Peocessus:

Table 4.9: Description d'un agent logiciel

9	Agent logiciel	
Identificateur		
Modèle		
Langage		
Senseurs et effecteurs	Communication	
	Environnement	
Tâche de l'agent		
Savoir-faire		
Programme de lancement		
Légende	Acronyme	Description
Commentaires		

3. **Attribuer les tâches aux agents**

Entrées : Modèles de rôles de l'organisation

Sorties : Association {(A, IT)/ A: Agent et IT: Tâche de l'agent A qui est l'ensemble des tâches des rôles que l'agent A va assumer}

4. Classer les modèles dérivés des agents en modèles d'agents similaires sur les senseurs, effecteurs, protocoles de communications, savoir-faire, tâches, états et/ou contrôles.

Entrées : Ensemble des modèles d'agents du SMA,

Sorties : Ensemble de modèles d'agents similaires.

4.4 Implémentation des modèles de systèmes multi-agent

4.4.1 Moyens de description

La table de description d'un d'agent logiciel (Table 4.9-Page 91) comporte :

Table 4.10: Description d'un système multi-agent

10	Programme de lancement d'un système multi-agent	
Identificateur		
Modèle		
Langage		
Etablissement des liens de communication et affectation des tâches		
Légende	Acronyme	Description
Commentaires		

1. L'identificateur de l'agent,
2. Le modèle d'agents que cet agent logiciel implémente,
3. Le langage de programmation dans lequel cet agent logiciel a été implémenté,
4. Les classes des objets senseurs et effecteurs (de communication et usuels),
5. Les classe d'un objet fictif représentant une implémentation de la tâche de l'agent logiciel,
6. Le savoir-faire de l'agent logiciel,
7. La classe de l'agent incluant les classes précédentes.

La table de description d'un SMA (Table 4.10-Page 92) comporte :

1. L'identificateur du SMA,
2. Le modèle du SMA associé,
3. Le langage de programmation dans lequel ce SMA a été implémenté,
4. L'établissement des liens de communication entre les différents agents du SMA et attribution de tâches à ces agents selon l'attribution de rôles

4.4.2 Processus d'implémentation

Le processus d'implémentation d'un modèle multi-agent peut être décrit comme suit :

Entrées : Modèle de SMA, Ensemble de modèles d'agents communicants

Sorties : Un ensemble d'agents logiciels communicants sous forme de systèmes orientés objet communicants,

Remarques :

Processus :

1. Implémentation des agents d'interface pour des agents non logiciels avec les procédures de leurs savoir-faire

   Entrées : Modèle de SMA, Ensemble des non logiciels,

   Sorties : Un ensemble d'agents d'interface,

   Remarques : L'implémentation des procédures du savoir-faire d'un agent d'interface est simple et systématique.

   Processus :

2. Implémentation des modèles des agents logiciels et des procédures de leurs savoir-faire

   Entrées : Modèle de SMA, Ensemble des non logiciels,

   Sorties : Un ensemble d'agents d'interface,

   Remarques : L'implémentation des procédures des agents logiciels est un problème de programmation classique.

   Processus : Aucun

3. Implémentation des protocoles de communication basés sur les actes de langage

   Entrées : ,

   Sorties : ,

   Remarques : Généralement, ces protocoles sont prédéfinis.

   Processus : Des suggestions pour l'implémentation de ces protocoles sont données dans la sous-section 4.4.3

4. Implémentation du module de lancement du système multi-agent

   Entrées : Relation de communication entre les agents du modèle de SMA,

   Sorties : Module de lancement du système multi-agent,

   Remarques : Des suggestions pour l'implémentation de ce module sont données dans la sous-section 4.4.3

   Processus : Aucun

4.4.3 Squelettes pour implémenter des éléments d'agents

Dans cette sous-section, nous décrivons l'implémentation de certains éléments des agents. Nous décrivons alors l'implémentation des senseurs et des effecteurs (usuels ou de communication), savoir-faire, la tâche d'un agent et le système agent.

Senseurs et effecteurs de communication

Un senseur de communication est un objet dont l'état dépend du protocole de communication de base utilisé. Mais il doit comporter :

1. Une file d'attente des messages reçus,
2. Une méthode "Receive" qui permet de recevoir les messages et de les mettre dans la file d'attente du senseur de communication. Cette méthode est destinée d'être invoquée par l'effecteur de l'agent qui veut communiquer avec cet agent,
3. Une méthode "FinalReceive" qui sera invoqué par l'agent pour consommer (utiliser) le premier message (de la file d'attente) reçu.

```
Class CS /* CS: Senseur de communication
{
  ...
      Queue Q; /* Q : File d'attente des messages reçus
 ...

      /** La méthode Receive sera invoquée par l'objet "Effecteur" de l'agent qui
voudra communiquer avec l'agent propriétaire de ce senseur */
      Public Void Receive(Sender S; Performative P; Act A, Parameters Pars);
       {
          ...
            This.Q.In(S, P, A);
        ...
       }
     /** FinalReceive sera invoquée par l'agent propriétaire de ce senseur */
   Public Void FinalReceive(Sender S; Perf P; Act A, Parameters Par);
   {
    ...
      This.Q.Out(S, P, A);
   ...
  }
```

```
Void CS ()
{
  ...
 }
...
}
```

Un effecteur de communication est un objet dont l'état dépend du protocole de communication de base utilisé. Mais il doit comporter :

1. Une méthode "Send" qui sera invoqué par l'agent pour envoyer un message. Une fois invoquée, la méthode "Send" invoquera la méthode "Receive" du senseur de communication de l'objet récepteur de ce message,

2. Le constructeur d'un objet effecteur de communication doit être initialisé par une référence à l'objet "effecteur de communication" de l'agent récepteur. Ceci constitue un établissement logiciel d'un lien de communication avec l'agent récepteur.

```
Class CE /* CE: Effecteur de communication
{
  ...
       SC R;
       /* La méthode Send sera invoquée par l'agent propriétaire de cet effecteur
*/
      Public Void Send(Receiver R; Sender S; Performative P; Act A, Parameters
Pars);
      {
        ...
           R.Receive(S, P, A, Par);
        ...
      }
     Void CE (SC R)
      {
        ...
           This.R=R;
       ...
      }
    ...
}
```

Senseurs et effecteurs de l'environnement

L'état d'un senseur de l'environnement dépend de la nature de l'environnement et du senseur particulier. Il doit comporter une méthode "Percept" qui établit l'état de l'objet senseur à partir de l'état de l'environnement.

```
Class ES /* ES: Senseur de l'environnement
{
  ...
    Public Void Percept(State St);
   {
      ...
         St.In(...) ;
      ...
    }
  ...
}
```

L'état d'un effecteur de l'environnement dépend de la nature de l'environnement et de l'effecteur particulier. Il doit comporter une méthode "Affect" qui permet à l'agent d'agir sur l'environnement selon une des actions A1, ..., As.

```
Class EE /* EE: Effecteur de l'environnement
{
   ...
        Void A1(Parameters P1)
        {
          ...
        }
   ...
        Void As(Parameters Ps)
        {
          ...
        }
    /* A est l'action a effectuer. Elle représente une invocation d'une des procédures
A1, ..., As du savoir-faire de l'agent
        Public Void Affect(Act A; Parameters Par);
        {
          ...
            This.A(Par) ;
          ...
```

```
    }
  ...
}
```

Tâche d'un agent

Pour systématiser le développement des agents, la tâche de l'agent a été considérée comme étant un objet particulier comportant la tâche de l'agent. Cet objet comporte une méthode d'initialisation "ReInitialize" et une méthode "Interpret" qui invoquera l'interpréteur du CPN de la tâche de l'agent.

```
Class Task
{
  ...
     Public Void ReInitialize(Task T)
    {
      ...
     }
  /* Si le CPN est compilé, la procédure Interpret sera remplacée par le programme
résultat de la transformation du CPN en un programme Java
   Public Void Interpret (T)
   {
     ...
    }
 ...
}
```

Savoir-faire d'un agent

Le savoir-faire d'un agent se présente sous forme d'un ensemble de méthodes d'objets (du modèle orienté objet) représentant le modèle de l'environnement. Chaque objet comporte un sous ensemble de ces méthodes M1, ..., Mn.

```
Class ...
{
  ...
     Public Void M1 (...)
    {
      ...
     }
  ...
```

```
    Public Void Mn (...)
    {
      ...
    }
  ...
}
```

Agent logiciel

L'agent logiciel se présente sou forme d'un système orienté-objet implémenté par un objet thread Java avec des objets "Public" représentant les senseurs pour qu'ils soient modifiables par les autres agents ou l'environnement. Il comporte aussi l'objet "Tâche de l'agent" et la méthode "Run" qui permettra de lancer le thread de l'agent.

```
Class AgT Extends Thread
{
  ...
    //Senseurs, effecteurs et protocoles de communication
    Public CS S1, ..., Sn ;
           CE E1, ..., Em;
    //Tâche de l'agent
    AgentTask AgT       /* AgT: Tâche de l'agent
    //Senseurs et effecteurs de l'environnement et savoir-faire
  ...
    Void Run /* Run est la méthode "Run" du thread
    {
      ...
        AgT.Interpret(TT)
      ...
    }
  ...
}
```

Si certains agents doivent être situés dans différents ordinateurs d'un réseau, nous pouvons expliciter ceci dans la méthode Run de telle sorte que l'agent se déplace à la machine désirée comme un agent mobile ou son initialisation sera faite avec une adresse réseau appropriée.

Etablissement des liens de communication, attribution des tâches et lancement des agents

La classe suivante peut être vue comme un SMA. Elle permet :

1. La déclaration des différents agents,
2. Etablissement des liens de communication entre les agents en initialisant leurs objets effecteurs par les références aux objets senseurs des agents avec lesquels ils vont communiquer selon le modèle de SMA associé,
3. Attribution des tâches en précisant les CPN des tâches des rôles que l'agent va assume,
4. L'activation des différents thread.

```
Class ...
{
    Void MASid; /* MASid: identificateur du SMA
    {
        /* Agents declarations
       AM1 A11, A12,..., A1n1; /* AM1: Modèle d'agents 1
      // A11: Agent 1 du modèle 1
      // A12: Agent 2 du modèle 1
      // A1n1: Agent n1 du modèle 1
            ...
    AMm Am1, Am2..., Amnm; /* AMm: Modèle d'agents m
     // Am1: Agent 1 de modèle m
     // Am2: Agent 2 de modèle m
     // Amnm: Agent mn de modèle m
           ...
   /* Etablissement des liens de communication: Pour chaque lien de communication
dans les liens de communication du modèle multi-agents entre deux agents Aij et Akl
faire in initialisation comme suit */
    Aij.CSAijAkl.Init(Akl.CEAklAij) /* Aij: Agent j du modèle i
     // Akl: Agent k du modèle l
     // CSAijAkl: Senseur de communication de l'agent Aij avec l'agent Akl
     // CEAklAij: Effecteur de communication de l'agent Aij avec l'agent Akl
          ...
    // Attribution de tâches (Pour chaque agent Aij)
```

```
        ...
  A11.Init(A11IT) /* AijIT: Tâche de l'agent A11
        ...
  Amn.Init(AmnIT) /* AijIT: Tâche de l'agent Amn
        ...
   // Activation des agents (Lancement des threads des agents Aij)
        ...
  A11.Start();
       ...
  Amn.Start();
 }
}
```

4.5 Conclusion

L'autosuffisance des tables permettra d'y inclure tous les aspects relatifs à ce développement. De cette façon, le développeur n'aura pas besoins d'utiliser d'autres formes pour expliquer les aspects relatifs au développement du SMA. Cependant, beaucoup reste à faire pour rendre les outils pratiques de MASA-Method plus lisibles, plus conviviaux et plus complets.

Les formes des moyens de description des SMA avec les processus associés ont été implémentées dans un environnement de développement de SMA selon MASA-Method. Nous avons implémenter beaucoup d'aspects des moyens pratiques d'utilisation de MASA-Method. De plus, les différents processus associés aux moyens de description ont été implémentés dans différentes versions de l'environnement de développement de SMA selon MASA-Method (centralisé et distribué).

Une représentation informatique des aspects de modélisation constitue un langage de programmation de très haut niveau. Les processus méthodologiques permettent de montrer comment identifier des éléments des modèles de SMA et comment dériver d'autres. L'automatisation de ces différents processus permettra de produire les différents systèmes agents. Elle permettra aussi de distribuer ces agents sur un réseau pour qu'ils fonctionnent et constituer un SMA. Cependant, beaucoup d'aspects de ces processus ne sont pas automatisables pour le moment. Ils sont destinés à être mis en œuvre par des développeurs.

Chapitre 5

Modélisation multi-agent du processus logiciel

Le mot "processus logiciel"[1] a été inventé en 1969 par Lehman [Lehman 69] qui l'a qualifié de "processus de programmation"[2]. Lehman l'a défini comme étant "toute la collection de technologies et activités qui transforment le germe d'une idée en robinet binaire e de programme"[3]. Le processus logiciel a été inspiré de la chaîne de production industrielle. Il a été modélisé pour le comprendre, pour l'améliorer, pour l'automatiser partiellement, et pour guider les personnes impliquées. Pendant la décennie entre 1985 et 1995, plusieurs travaux ont essayé de proposer des approches au problème de modélisation du processus logiciel. Osterweil a publié en 1987 un papier intitulé "Software processes are software too" [Osterweil 87] dans lequel il stipule que le processus logiciel est un système logiciel complexe qui peut être développé en lui appliquant des méthodes et des techniques qu'on utilise dans le développement de logiciels. Nous, comme plusieurs autres chercheurs, avons suivis la caractérisation du processus logiciel faite par Osterweil.

Le logiciel "processus logiciel" exige l'accomplissement d'activités de diverses natures (divers domaines d'application, gestion du développement, conception, programmation, ...). Un certain nombre de ces activités sont automatisables (compilation, débogage, aide, etc..). Dans ce qui suit, ces activités seront qualifiées de moyens automatiques. D'autres activités sont des activités exclusivement humaines. Les développeurs (humains) impliqués dans un développement de logiciels raisonnent de différentes manières, emploient des connaissances diversifiées (non unifiées), ont des comportements hétérogènes et peuvent être placés dans des endroits éloignés. Dans

[1] Traduction française du terme "Software process"

[2] Traduction française du terme "programming process"

[3] Traduction par l'auteur du text "the total collection of technologies and activities that transform the germ of an idea into a binary program tape"

ces conditions, ces développeurs doivent coopérer pour un développement cohérent de logiciels. Il est alors important qu'un modèle du processus logiciel :

1. Intègre ces intervenants, qu'ils soient automatiques ou humains,
2. Supporte la coopération des moyens de développement des logiciels. Les moyens automatiques doivent coopérer entre eux, et les développeurs doivent coopérer entre eux mais les développeurs et les moyens automatiques doivent aussi coopérer,
3. Supporte l'évolution de ces moyens : L'évolution des moyens automatiques, le recrutement et le départ des employés, l'évolution de la coopération des moyens de développement.

Actuellement, ces trois problématiques ne sont pas prises en considération d'une manière cohérente dans les travaux traitant les processus logiciels. Dans leur papier [Cugola 98], Cugola et Ghezzi déclarent que "Les processus centrés humain sont caractérisés par deux aspects cruciaux qui ont été en grande partie ignorés par la plupart de la recherche sur processus de logiciel : Ils doivent supporter la coopération des personnes, et ils doivent être fortement flexibles."[4]. Dans son papier [Balzer 00], Robert Balzer déclare également que les deux principaux problèmes de la technologie existante du processus logiciel est qu'elle ne donne pas aux développeurs de logiciels le support du processus coopératif adapté et elle ne fait pas face aux changements du processus logiciel. Ces deux visions étaient encore soutenues par Conradi, Fugetta et Jaccheri dans [Conradi 98] qui proclament que le processus logiciel a échoué de s'adapter aux parties du développement de logiciel impliquant des humains.

Un modèle est une abstraction, c'est-à-dire, une représentation simplifiée, d'un système plus complexe réel ou conceptuel [Kellner 99]. L'abstraction fait qu'un modèle ne comporte de la réalité que certains éléments considérés comme pertinents pour un objectif donné. Un tel objectif peut appartenir à l'une des classes suivantes :

1. Analyse d'une réalité existante : Le modèle sert comme une représentation de certains éléments d'une réalité complexe pour simplifier un traitement (généralement manuel) difficile à effectuer directement sur une telle réalité. On parle simplement de "modélisation",
2. Simulation d'une réalité existante : Le modèle sert pour une réalisation simplifiée reproduisant seulement les aspects nécessaires d'une certaine réalité complexe pour un objectif donné. Dans ce cas, on parle de "simulation",

[4] Traduction par l'auteur du text "Human-centred processes are characterised by two crucial aspects that were largely ignored by most software process research: They must support cooperation among people, and they must be highly flexible."

3. Réalisation d'une nouvelle réalité : Le modèle sert comme moyens de description d'une conception afin de réaliser un nouveau système. Le nouveau système est destiné à remplacer le système existant. Ce nouveau système est vu comme étant mieux que l'ancien. On parle de "conception".

La modélisation du processus logiciel peut concerner l'un des trois types de modélisation précédents. Elle peut être destinée à la recherche de la meilleure organisation des activités de développement. Une telle modélisation est celle destinée à l'amélioration du processus logiciel (tel que [Paulk 93]). Elle peut aussi être destinée à une simulation permettant de tester une organisation donnée des activités, telle que [Kellner 99]. Cependant, les travaux de modélisation du processus logiciel les plus importants sont ceux destinés à l'automatisation totale ou partielle du processus logiciel. De tels travaux sont destinés au développement d'environnements de développement de logiciels, tels que [Perry 91].

Dans ce chapitre, nous nous intéressons à l'automatisation du processus logiciel sous forme d'un environnement de développement de logiciels. Le reste de ce chapitre est organisé en trois sections. La section 5.1 présente quelques approches d'automatisation (complète ou partielle) du processus logiciel. Elle décrit les approches d'intégration, de séquencement, de coordination et de coopération des outils. La section 5.2 présente la modélisation multi-agent du processus de développement de logiciel en utilisant MASA-Method. Enfin, le chapitre se termine par une conclusion à la section 5.3.

5.1 Environnements de développement de logiciels

Le développement de logiciels est resté longtemps entièrement à la charge des développeurs. Ces développeurs sont les contrôleurs de la cohérence de leurs activités [Lahlouhi 97]. Cette pratique a amené à divers problèmes tels que :

1. Une augmentation considérable des coûts des logiciels,
2. Un allongement des délais de livraison des logiciels et leurs dépassements répétés,
3. Des pertes énormes des entreprises à cause des erreurs dans des logiciels dont certains ont été retirés du marché,

Ces problèmes ont imposé le recours à l'assistance de l'ordinateur. Celle-ci est faite à travers des outils (correcteurs, analyseurs de flot de données, ...) qui sont mis à la disposition des développeurs pour les aider dans leurs activités de développement. Ces outils sont des prescriptions de méthodes à suivre pour accomplir une activité donnée

de développement de logiciels. Pour bénéficier des services de ces outils, il faut tout d'abord comprendre leur méthode d'utilisation. Chacun de ces outils est dédié à une activité déterminée de développement de logiciels et, plus particulièrement, à celles relatives à la programmation. Ces outils ont été réunis, par la ensuite, dans des boites à outils, puis intégrés pour devenir des environnements. Un environnement couvre une partie du cycle de vie logiciel que peut couvrir une combinaison de plusieurs outils.

5.1.1 Environnements comme automatisation de méthodes de développement spécifiques de logiciels

Le développement d'environnements couvrant tout le cycle de vie logiciel est difficile à cause de l'absence d'outils concernant certaines activités de développement. Les méthodes utilisées par les développeurs pour ces activités sont empiriques (diversifiées, informelles et mal comprises). Par conséquent, ces activités ne sont pas formalisées dans des outils. De plus, les limites entre de telles activités sont variables et floues. Ce qui ne permet pas de distinguer ce qui appartient à un outil de ce qui doit appartenir à un autre. Ceci a imposé à la majorité des environnements d'être des environnements de programmation où les activités sont, généralement, bien formalisées.

Les premiers environnements imposent des méthodes de développement particulières et même un langage de programmation particulier. Cette obligation a posé des difficultés aux développeurs qui sont contraints tous d'utiliser la même méthode malgré les différences méthodologiques et pragmatiques qui peuvent exister entre les différentes organisations et aussi entre les différents individus. Ces difficultés ont amené à l'introduction d'un niveau d'abstraction en considérant le développement des environnements comme développement d'un logiciel et, ils ont été qualifiés de processus logiciels [Osterweil 87]. Cette abstraction permet aux différentes organisations de développer leurs propres environnements. Cependant, les paradigmes classiques ne suffisent pas de prendre en charge plusieurs besoins de modélisation du processus logiciel. Différentes approches ont été alors proposées ou réadaptées pour supporter le développement de processus logiciels. Par manque de généralité, chaque approche offre une vision différente de développement de logiciels. Cette vision est celle de l'auteur de l'approche. Ce qui a fait qu'un processus à base d'un paradigme donné rentre dans une catégorie de méthodologies. L'objectif final de ces approches est l'allégement de l'automatisation des différentes méthodologies de développement de logiciels.

5.1.2 Automatisation du processus logiciel

L'automatisation d'une tâche donnée nécessite la détermination de combinaisons cohérentes d'activités "accomplissables" permettant de dériver une situation à partir

d'une autre situation donnée. En intelligence artificielle, ceci est appelé génération de plans [Hayes-Roth 85]. D'habitude, cette tâche est réservée aux humains qui prescrivent à la machine ce qu'elle doit faire sous forme d'un programme. Automatiser une telle tâche dans le cas général était un but inachevé de l'intelligence artificielle.

Pour automatiser le processus logiciel, deux approches ont été utilisées: l'approche formelle et l'approche cognitive. L'approche formelle consiste en la spécification du logiciel dans une théorie bien fondée. La spécification peut être considérée comme un ensemble de théorèmes qui doivent être prouvées. Elle est mise en oeuvre, ensuite, par prototypage ou par application de règles de passage. La correction, la consistance et la complétude de ces règles doivent être prouvées aussi. Le résultat est un programme dont sa correction est prouvée et sa maintenance est plus facile, puisqu'elle se fait au niveau des spécifications.

Le problème de l'approche formelle est qu'elle nécessite, préalablement, la maîtrise de certaines théories. Cette maîtrise doit permettre le développement de spécifications et faire les démonstrations nécessaires. Généralement, les développeurs des applications trouvent des difficultés d'apprendre de nouvelles théories. De plus, le développement des spécifications formelles est long et exige un très large consensus sur les définitions des concepts du domaine du problème. Ceci veut dire que les travaux sur l'approche formelle se font, principalement, sur des domaines stables et, par conséquent, relativement anciens. Ceci ne convient pas une discipline telle que les processus logiciels dont les changements de technologies sont très fréquents (changement des moyens de communication, changements des modèles, etc.). Ce qui change continuellement les besoins et les terminologies utilisés pour satisfaire ces besoins. L'approche formelle à l'automatisation du processus logiciel prend plus en considération les aspects de performance (complétude, correction, consistance, ...). Elle prend avec moins de considération les aspects humains. L'approche formelle défend la vision du "comment doit-on construire des logiciels". A l'inverse, l'approche cognitive qui considère "comment sont réellement construits les logiciels".

5.1.3 Approches méthodologiques au développement de processus logiciels

Dans les environnements actuels, le développement de logiciels est basé sur l'invocation d'outils. Les outils manipulent des objets logiciels, tels que les spécifications, les documents de conception, les programmes, ... Ces objets sont sauvegardés sous forme d'un ensemble de fichiers ou d'une base de données. L'approche base de données est une amélioration de celle des fichiers où la modélisation, l'accès, l'évolution, la gestion de contraintes d'intégrité et les relations entre objets sont facilités par l'utilisation

d'un système de gestion de bases de données. Dans des systèmes à commandes seulement (DOS, UNIX, ...), le développeur choisit selon ses besoins, à chaque étape du développement du logiciel, l'outil nécessaire lui permettant d'atteindre son objectif. Une telle utilisation des outils est qualifiée d'invocation manuelle. De tels systèmes (DOS, UNIX, ...) ne sachent pas ce que fait le développeur. Ils ne savent que l'outil que le développeur a invoqué. Il appartient au développeur de savoir les outils, les conditions et le plan qui lui y sont nécessaires pour atteindre son objectif. Il doit aussi préparer ce plan et vérifier la validité de telles conditions. Une telle pratique a posé beaucoup de difficultés aux développeurs et elle leur fait perdre du temps, d'énergie et d'attention. En plus, elle est provocatrice d'erreurs.

La méthodologie de développement de tels systèmes est simple. Elle se réduit à celle de développement des outils où chacun est développé selon une méthodologie différente ou de façon complètement empirique. La méthodologie de développement de logiciels, avec ces systèmes, est soumise à une prescription de méthodes d'utilisation (manuels d'utilisation) des outils dont le développeur doit respecter les moindres détails. La gestion du plan (son développement et sa cohérence) d'invocation des outils est à la charge du développeur.

Les améliorations sont faite par l'introduction des environnements selon trois axes : intégration des outils [Arnold 92, Dawson 87, Dixon 88, Perrin 92, Thomas 89], leur enchaînement automatique [Belkhatir 91b, Kaiser 87, Kaiser 88, Kaiser 90, Khammaci 91, Khammaci 92, Sutton 91] et leur coordination et coopération [Alloui 95, Belkhatir 91a, Ciancarini 93, Meslati 96]. Ces axes seront détaillés dans les sous sections suivantes.

5.1.4 Intégration des outils

L'intégration des outils permet de donner une vision globale unifiée à l'ensemble des outils indépendamment de leurs aspects spécifiques tels que les langages de leurs implémentations, les interfaces de leurs utilisations et la forme de stockage des données. Il y a trois principaux types d'intégration : intégration au niveau des interfaces, intégration au niveau des données et intégration au niveau du contrôle. Dans une intégration au niveau des interfaces [Perrin 92], le développeur interagit avec un outil à travers le système global. Les améliorations apportées par cette intégration sont au niveau d'apprentissage, de l'utilisation de l'environnement, où le développeur utilise tous les outils "presque" de la même façon. Dans une intégration de données [Perrin 92], l'utilisateur ne gère pas les mises à jour de ses données redondantes, n'a pas besoin de transformer les résultats d'un outil pour qu'un autre puisse les utiliser comme entrée, ... Enfin, dans l'intégration de contrôle [Arnold 92, Perrin 92], un autre aspect

d'amélioration est celui d'inclusion des outils dans des plans automatiques. Cependant, ces plans sont prédéfinis et soumis à des règles très simples, telles que le séquencement. Par exemple, à la fin de l'édition il y a lancement automatique de la compilation. L'environnement, dit intégré, dispose alors de certaines connaissances sur les outils tels qu'un ordre de leur invocation. Ce type de connaissances est élémentaire, "très" difficilement modifiable, et son utilisation est fixe et n'est pas intelligente. Dans les types d'intégration, celle des données est la plus utilisée et la plus maîtrisée.

Dans ce qui suit, nous détaillons l'approche d'intégration des données utilisée dans [Perrin 92]. Dans [Perrin 92], l'intégration de données est basée sur des connaissances, dites représentation, des données manipulées par les outils. Ces connaissances comprennent, pour chaque donnée :

1. Son type (sa forme de stockage dans l'environnement),
2. Son rôle (sa signification lorsqu'elle est manipulée par l'outil),
3. Son format (l'unité d'une valeur numérique).

Ces connaissances permettent d'informer sur les transformations à effectuer pour permettre à un outil de gérer les données d'un autre. Pour effectuer ces transformations, trois niveaux de compatibilité, entre les attributs des représentations de deux outils, ont été définis :

1. Partiel s'il existe dans deux représentations différentes des attributs communs dont leurs types et formats ne sont pas identiques,
2. Simple s'il existe des attributs communs dont leurs types et formats sont identiques,
3. Total s'il y a une correspondance totale entre attributs, types et formats.

Le convertisseur est constitué d'un ensemble d'opérateurs chargés d'analyser les représentations. Si les représentations sont compatibles alors les données seront transformées suivant le niveau de compatibilité détecté et des règles préétablies entre rôles, types et formats.

La méthodologie de développement d'un environnement se trouve modifiée, par rapport à la précédente. Un autre niveau d'abstraction a été ajouté à celui de développement des outils, c'est celui de développement de l'environnement qui va les intégrer. Cette intégration est une imposition d'une certaine méthodologie d'utilisation des outils pour le développement de logiciels. Elle n'est plus libre comme avant, mais elle est préétablie (imposé) : structure modulaire, hiérarchie de contrats [Dixon 88, Thomas 89], ...

5.1.5 Enchaînement automatique des outils

Pour mieux automatiser le processus logiciel, il doit avoir un savoir-faire équivalent à celui des développeurs, pour la manipulation de ces outils, et qu'il soit apte à utiliser ces connaissances. Parmi ces connaissances, on trouve les conditions initiales d'invocation des outils. Doter un environnement de telles connaissances et de la manière de les utiliser permet d'invoquer automatiquement les outils. L'invocation d'un outil, dont les conditions initiales sont vérifiées, modifiera les objets logiciels. Ceux-ci vérifieront les conditions initiales d'un autre outil, qui sera invoqué. Celui-ci modifiera, à sont tour, les objets logiciels et ainsi de suite ; ce qui enchaînera les outils de façon automatique.

Les conditions initiales d'invocation des outils sont évaluées à partir des connaissances sur les outils et les objets logiciels. Les approches les plus utilisées pour représenter ces connaissances sont celles de règles (Par exemples, MARVEL [Kaiser 87, Kaiser 88, Kaiser 90] et UPSSA [Khammaci 91, Khammaci 92]) et celles de déclencheurs (Par exemples, APPL/A [Sutton 91] et ADELE [Belkhatir 91a, Belkhatir 91b]). Dans les paragraphes suivants, nous décrivons ces deux approches telles qu'elles sont utilisées. Nous commencerons par l'approche à base de règles.

Une règle de production est un couple (Condition, Action) où l'action sera exécutée si la condition est vérifiée. Dans une approche à base de règles, à la MARVEL, les conditions initiales d'invocation des outils sont considérées comme étant des conditions des règles, dites pré-conditions à l'action. L'action est subdivisée en deux parties : L'invocation de l'outil et les post-conditions. Celles-ci sont des opérations modifiant les objets logiciels. L'enchaînement automatique commence par l'invocation d'un outil t0 par un utilisateur selon ses objectifs. Deux cas peuvent alors se présenter :

1. Les pré-conditions d'invocation de l'outil t0 sont satisfaites alors t0 et ses post-conditions seront exécutées. Selon le nouvel état des objets logiciels, résultat de l'exécution des post-conditions, les pré-conditions d'un autre outil t1 peuvent être vérifiées. Par conséquent, il sera invoqué, et l'enchaînement continu par chaînage avant. Dans le cas où les pré-conditions de tous les outils ne seraient pas satisfaites, le contrôle sera rendu à l'utilisateur pour qu'il choisisse de nouveau un autre outil, pour recommencer le même processus,

2. Les pré-conditions d'invocation de de l'outil t0 ne sont pas satisfaites, elles seront alors considérées comme un but à satisfaire par chaînage arrière. Pour cela, les post-conditions des autres outils seront examinées pour déterminer laquelle d'entre elles permet de rendre satisfaites, par modification des objets logiciels, les pré-conditions de l'outil t0. Si les post-conditions de l'un des outils (t2) permettent de satisfaire les pré-conditions de t0 alors il sera invoqué automa-

tiquement, ce qui le soumettra au processus d'invocation des outils (vérification de ses conditions initiales, ...).

Les règles de MARVEL ne permettent pas une programmation événementielle, ce qui ne permet pas d'avoir des réactions instantanées pour une meilleure interaction avec l'utilisateur. Les déclencheurs permettent de mieux prendre en considération cette réactivité. Un déclencheur est une connaissance opératoire sous forme d'un couple (Evénement, Action). Une condition peut être attachée à l'action ou à l'événement.

Dans ce dernier cas, l'événement sera considéré comme étant l'atteinte du système sous développement d'un certain état déterminé. Ce qui rend le déclencheur, dans ce cas, semblable à une règle de production [Belkhatir 91b]. Le déclencheur à une caractéristique de réactivité, i.e., il réagit instantanément à l'apparition de l'événement auquel il est associé. Les déclencheurs seront alors attachés aux objets logiciels et réagiront à des opérations sur eux pour lancer les actions (ou outils). Ces outils modifieront des objets logiciels et d'autres déclencheurs réagiront en conséquence. Ce qui enchaînera les outils de façon automatique.

La méthodologie de développement des environnements se base sur l'utilisation d'un moteur d'inférence. Il faut alors, en plus du développement des outils, choisir (et adapter) ou développer un moteur d'inférence, et développer une base de connaissances sur les objets logiciels et les outils. Les connaissances sur les outils incluent, principalement, les conditions initiales de leurs invocations, leurs paramètres et la syntaxe des commandes de leurs invocations.

Du point de vu de l'utilisation de ces environnements, l'utilisateur se contente de choisir, à chaque fois, l'outil qui lui permettra d'atteindre son objectif sans se préoccuper des outils intermédiaires à invoquer. Ceci lui permettra d'ignorer, au moins théoriquement, les connaissances qui les concernent. Par exemple, il peut lancer l'exécution de son programme sans se préoccuper de sa compilation ni son édition de liens. Il n'a pas à savoir comment lancer le compilateur, ses paramètres, etc.

La méthodologie de développement de logiciels, libre en apparence, est soumise à cet enchaînement des outils. Celui-ci (qui n'est autre qu'un séquencement des outils) permet de diriger le travail du développeur sur les étapes à suivre. La méthodologie est, par conséquent, imposée par les outils sur lesquels est basé l'environnement. Cependant, elle peut être modifiée par remplacement des outils ou leur modification.

5.1.6 Coordination et coopération des outils

L'enchaînement automatique des outils permet de déterminer des séquencements consistants des outils. Cependant, les développeurs, utilisateurs des outils, ne travaillent pas de manière complètement séquentielle. Ils peuvent travailler en parallèle, leurs

activités peuvent être dépendantes et ils interagissent pour échanger des informations ou pour se rendre des services.

La prise en considération de ces aspects nécessite plus de connaissances sur les outils que les conditions de leurs invocations. C'est la considération des connaissances sur les dépendances entre outils, des utilisateurs multiples et des buts à atteindre au lieu des outils à exécuter. Ceci ne peut être pris en considération convenablement par les approches de l'intelligence artificielle classique. Alors des approches à base d'intelligence artificielle distribuée ont été utilisées. Les techniques, d'intelligence artificielle distribuée, utilisées sont le tableau noir (blackboard) (Exemple, ESP [Ciancarini 93]), et les systèmes multi-agent (Exemple, Peace+ [Alloui 95]).

Le modèle de tableau noir est constitué d'une structure de données appelée tableau noir et d'un ensemble de sources de connaissances séparées. Les sources de connaissances ne communiquent qu'à travers le tableau noir, elles peuvent le modifier et, elles réagissent sur ces modifications. Le contrôle, dans les systèmes à tableaux noirs, peut être attaché aux sources de connaissances. Dans ce cas, une source de connaissance sera considérée comme un système expert avec son propre moteur d'inférence. Le tableau noir sert alors, en plus d'autres tâches qui peuvent lui être associées, comme moyen de communication entre les différents systèmes experts pour former un système multi-expert. Le contrôle peut être aussi centralisé en l'attachant au tableau noir ou comme un module indépendant. Les accès au tableau noir doivent être synchronisés, les actions des différentes sources de connaissances doivent être mutuellement consistantes, ...etc. On parle alors de coordination des agents (ou d'experts). Dans ce qui suit, nous décrivons comment est prise en considération la coordination dans ESP.

ESP (Extended Chared Prolog) utilise la technique du tableau noir (appelé, dans ESP, espace multiple de tuples: PoliS). Un tuple dans ESP est un terme PROLOG, ou un but, dans la terminologie de PROLOG. Il peut être considéré comme un objectif à atteindre. Un espace de tuples est un multi-ensemble nommé de tuples. PoliS peut comprendre plusieurs espaces de tuples. Les opérations sur de telles structures incluent la création d'espaces de tuples et, création, teste d'appartenance et suppression de tuples. La suppression d'un espace de tuples, dite terminaison, est automatiquement liée à un invariant de sa création. Celui-ci peut être considéré comme un but à atteindre.

Les tâches sont regroupées en modules appelés théories. Chacune d'elles comprend une interface et un programme PROLOG, dit implémentation de la théorie. L'interface comprend des règles qui peuvent être invoquées. Une règle est constituée d'un garde (qui peut être considéré comme une pré-condition), un but et une production. Elle a la forme suivante : Garde ! But Production. Le garde est constitué:

1. D'un test, sur les tuples que doit contenir l'espace de tuple,
2. D'une consommation de tuples, qui doivent exister dans l'espace de tuples avant leur suppression.

Le succès de l'évaluation du garde autorise l'évaluation du but selon l'implémentation de la théorie (programme PROLOG). Ceci peut être considéré comme une invocation d'un outil, écrit en PROLOG. Ce programme peut contenir aussi des outils externes à PROLOG. Pour prendre en considération l'échec de l'évaluation du but PROLOG, la production est subdivisée en deux opérations exclusives: création de certains tuples en cas de succès et création d'autres en cas d'échec. Ceci permet de recréer les tuples supprimés avant l'évaluation du but en cas d'échec, par exemple. Ce qui offre la possibilité de programmer les retours arrière, et une programmation indéterministe.

Le garde permet de savoir si les conditions d'invocation de l'outil sont réunies. Ces conditions sont basées sur des tuples qui peuvent être fournis par des mises à jour de l'espace de tuples par des agents. Dans ESP, le contrôle est centralisé. Une théorie peut-être considérée comme une base de règles. La base de fait est contenue dans l'espace de tuples. L'association du contrôle, d'une théorie et, d'une base de fait constitue un système expert (ou agent). La suppression permet de formaliser la prise en charge, par un agent, de l'objectif représenté par le tuple supprimé. La création de tuples permet de spécifier des résultats et des objectifs à considérer ultérieurement. Ceci pourrait être utilisé pour coordonner et faire coopérer des agents. Cependant, les agents de ESP ne sont pas complètement autonomes comme c'est le cas des agents réels puisqu'ils sont soumis à un contrôle centralisé.

Dans ce qui suit, on décrit comment est pris en considération l'interaction entre les agents dans Peace+. Peace+ [Alloui 95] est basé sur un modèle multi-agent à agents intentionnels. Chaque agent possède, par conséquent, une base de connaissances sur laquelle il peut raisonner et un objectif qu'il essaye d'atteindre. Le processus logiciel est considéré, dans Peace+, comme étant un ensemble d'agents qui coopèrent pour mettre en oeuvre un ensemble de tâches représentant des activités de développement de logiciels. Une tâche comprend un but à satisfaire par l'agent qui l'exécutera. Les agents coopèrent par interaction pour satisfaire leurs objectifs.

Dans Peace+, l'interaction est basée sur les actes du langage. Pour toute interaction entre deux agents, on associe deux diagrammes de transition d'états, un pour chaque agent. Les transitions, dans ces diagrammes, sont faites à travers l'émission ou la réception d'un des actes de communication (requête, acceptation, refus, ...). Ces actes de communication sont formalisés dans la logique multi-modale à l'aide des opérateurs de croyance, de possibilité, de nécessité, etc. L'activation du système per-

(1) R.A: Analyser les besoins pour produire une spécification du logiciel,
(2) M.D: Concevoir in modèle imitant la spécification du logiciel,
(3) S.P: Implémenter le modèle conceptuel.

Figure 5.1: Description informelle de la tâche du système de développement de logiciels

met aux agents de travailler de façon autonome pour atteindre leurs buts. Si un agent peut accomplir son but alors il le fait sinon, il fait appel à ces accointances qui ont en les compétences de le faire. Après le choix d'un agent, parmi ceux-ci, les deux agents (l'agent sélectionneur et l'agent choisi) interagissent pour se mettre d'accord sur l'objectif à réaliser. Pour cela, l'agent client (demandeur de service) poursuit un diagramme de transition d'état d'interaction avec l'agent serveur. Ce diagramme peut être qualifié de plan préétabli qui dirige l'agent sur l'interaction. A la réception de l'acte de communication, l'agent serveur poursuit le diagramme de transition d'état associé (relatif à l'agent client) pour pouvoir interagir avec le client.

5.1.7 Conclusion: Vers une modélisation multi-agent d'un environnement de développement de logiciels

L'intégration d'outils est une combinaison préétablie non modifiable facilement. Chaque intégration modélise une vision particulière de l'utilisation des outils. Cette vision est celle du réalisateur de l'environnement. Une intégration donnée ne permet pas de satisfaire des besoins d'utilisations spécifiques autres que celle de la vision sur laquelle cette intégration est basée. L'enchaînement automatique des outils permet la programmation de plusieurs utilisations possibles. Finalement, la coordination et la coopération considèrent les aspects concernant le travail collectif.

La prise en charge par l'environnement de tous les aspects humains concernant le développement de logiciels est une automation complète du processus logiciel. Cependant, ceci n'est pas possible, au moins, pour le moment. La section suivante (la section 5.2) décrit une approche de modélisation complète du processus logiciel comprenant ses aspects humains. Une telle approche n'est pas une automation complète des activités du processus logiciel. Plutôt, une telle automation complète concerne le processus lui-même. Certaines activités du processus logiciel seront effectuées par des humains mais elles seront dirigées par le processus automatisé.

Table 5.1: Description du modèle de tâches d'un système de développement de logiciels

1	Modèle de tâches d'un système	
Identificateur	SDT	
Description informelle	Figure 5.1	
Description formelle	Modèle de l'environnement	{R, M, S}
	Modèle de la tâche	Figure 5.2
Tâches élémentaires	Identificateur	Description
	A	Analyser les besoins pour produire une spécification du logiciel
	D	Concevoir un modèle imitant la spécification du logiciel
	P	Implémenter la conception
Légende	Acronyme	Description
	R	Requirements
	M	Model
	S	Software
	A	Analyze
	D	Design
	P	Program
	SDT	Software development task
Commentaires		

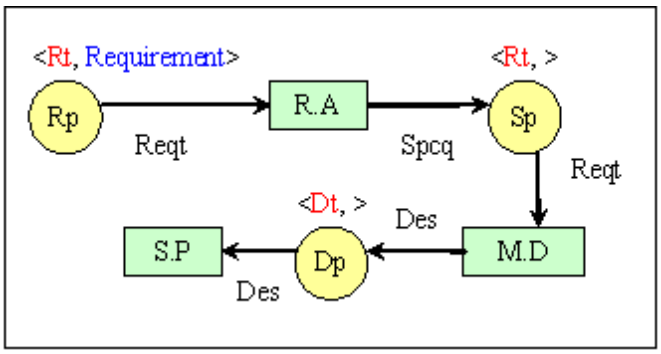

Figure 5.2: Description formelle de la tâche du système de développement de logiciels. La description des éléments du CPN est donnée dans la table 5.2

Table 5.2: Description des éléments du CPN de la figure 5.2

	Symbole	Description
Places	Rp	Requirement place (initialisée aux besoins de du logiciel à développer)
	Sp	Specification Place
	Dp	Design place
Types	Rt	Requirement type
	St	Specification type
	Dt	Design type
Transitions	R.A	Analyse des besoins et production d'une spécification
	M.D	Développer un modèle conceptuel imitant la spécification des besoins
	S.P	Programmer le modèle conceptual et produire un logiciel
Fonctions d'arcs	Reqt	Requirements
	Spcq	Specification
	Des	Design
Gardes		

(1) AGO.R.A: Analyser les besoins par la sous organisation AGO pour produire une spécification du logiciel,
(2) DGO.M.D: Concevoir in modèle imitant la spécification du logiciel par la sous organisation DGO,
(3) PGO.S.P: Implémenter le modèle conceptuel par la sous organisation PGO.

Figure 5.3: Description informelle de la tâche de l'organisation de développement de logiciels

Table 5.3: Description d'un modèle de tâches de l'organisation de développement de logiciels

2	Modèle de tâches d'une organisation composée		
Identificateur de la tâche	SDOT		
Tâche du système	SDT (Table 5.1)		
Organisation	SDO (Table 5.4)		
Tâche substituée	Aucune		
Description de la tâche	Modèle de l'environnement	{R, M, S}	
	Modèle de la tâche	Description informelle	Figure 5.3
		Description formelle	Figure 5.4
	Sous-organisations	Identificateur	Description
		AGO	Sous organisation d'analyse des besoins
		DGO	Sous organisation de conception de modèles conceptuels
		PGO	Sous organisation de programmation des modèles conceptuels
Légende	Acronyme	Description	
	AGO	Analysis group organization	
	DGO	Design group organization	
	PGO	Programming group organization	
Commentaires			

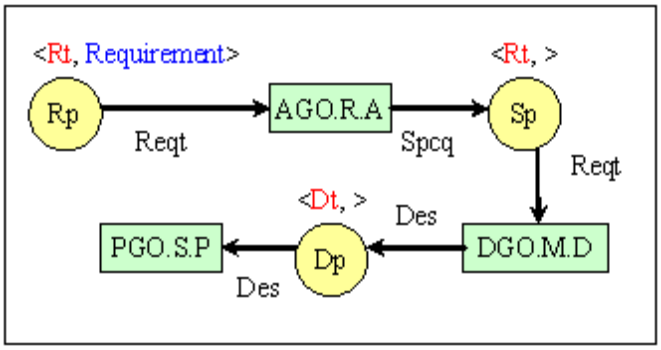

Figure 5.4: Description formelle de la tâche de l'organisation de développement de logiciels. La description des éléments de ce CPN est similaire à celle de la figure 5.2 donnée dans la table 5.2

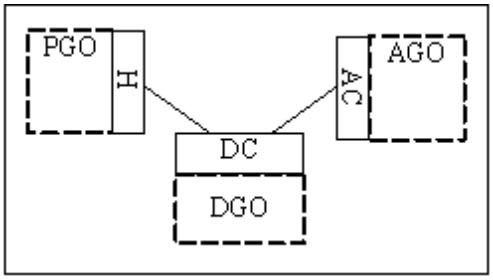

Figure 5.5: Représentation graphique de la relation de communication entre les représentants des sous organisations de l'organisation de développement de logiciels SDO de la table 5.14

(a) Ds.Br: Décomposer un modèle conceptuel en composants,
(b) Sc.Ex : Extraire les composants conceptuels du modèle de conception,
(c) Ci.Pg, i ∈ [1, n]: Programmer un composant pour produire un module,
(d) Md.Mem: Mémoriser un module Mi du composant Ci,
(e) Sm.Intg: Intégrer les modules résultats selon le modèle conceptuel.

Figure 5.6: Description informelle de la tâche d'un système de programmation

Table 5.4: Description d'un modèle d'organisations de développement de logiciels

3	Modèle d'une organisation composée		
Identificateur	SDO		
Organisation mère	Aucune		
Tâche	SDOT (Figure 5.3)		
Organisation substituée	Aucune		
Sous-organisations	Identificateur	Représentant	Modèle
	AGO	AC	Not given
	DGO	DC	Not given
	PGO	H	Table 5.10
Relation de communication	Graphique	Analytique	
		Emetteur	Récepteur
	Figure 5.5	H	DC
		DC	AC
Légende	Symbole	Description	
	AGO	Analysis group organization	
	DGO	Design group organization	
	PGO	Programming group organization	
	SDO	Software development organization	
	AC	Analyzer chief	
	DC	Designer chief	
	H	Programming chief	
Comment	Cette organisation décrit une organisation d'un processus de développement de logiciels		

Table 5.5: Description d'un modèle de tâches d'un système de programmation

4	Modèle de tâches d'un système	
Identificateur	PT	
Description informelle	Figure 5.6	
Description formelle	Modèle de l'environnement	{Ds , Md, Sm, Sc} ∪ {Ci / i ∈ [1, n]}
	Modèle de tâches	Figure 5.7
Tâches élémentaires	Identificateur	Description
	Ds.Br	Décomposer un modèle conceptuel en composants
	Sc.Ex	Extraire les composants conceptuels du modèle de conception
	Ci.Pg	Programmer un composant pour produire un module
	Md.Mem	Mémoriser un module Mi du composant Ci
	Sm.Intg	Intégrer les modules résultats selon le modèle conceptuel
Légende	Acronyme	Description
	Ds	Design
	Sc	Set of component
	Ci	Component i
	Md	Module
	Sm	Set of modules
	Br	Break
	Ex	Extract
	Pg	Program
	Mem	Memorize
	Intg	Integrate
Commentaires		

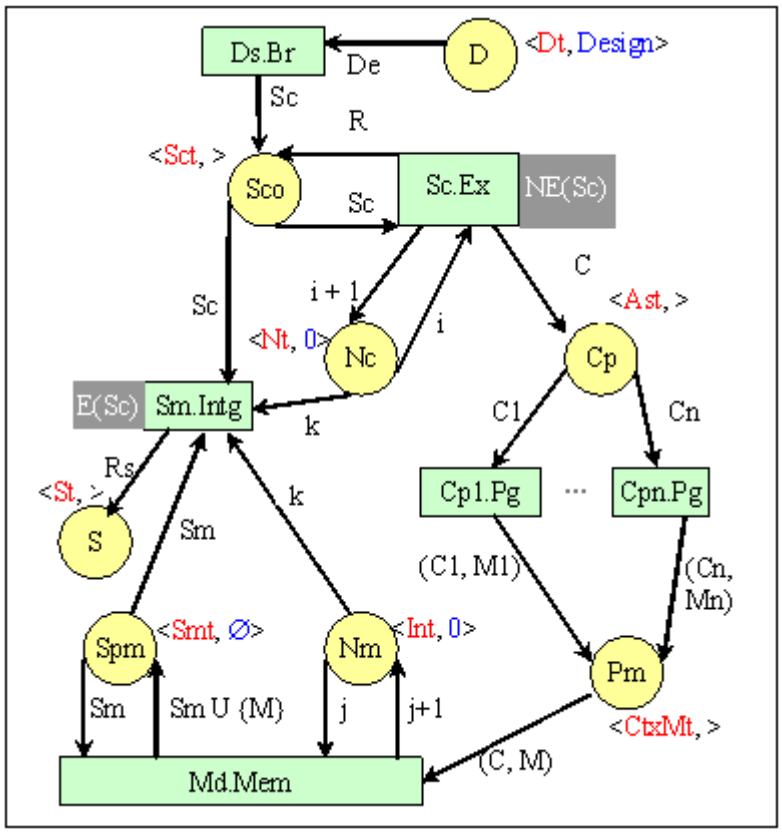

Figure 5.7: Description formelle de la tâche du système de programmation. La description des éléments du CPN est donnée dans la table 5.6

(a) H.Ds.Br: Décomposer un modèle conceptuel en composants,
(b) H.Sc.Ex : Extraire les composants conceptuels du modèle de conception,
(c) Pi.Ci.Pg, i ∈ [1, n]: Programmer un composant pour produire un module,
(d) H.Md.Mem: Mémoriser un module Mi du composant Ci,
(e) H.Sm.Intg: Intégrer les modules résultats selon le modèle conceptuel.

Figure 5.8: Description informelle de la tâche l'organisation de programmation

Table 5.6: Description des éléments du CPN de la Figure 5.7

	Symbole	Description
Places	D	Design (Initialisée avec la conception)
	Sco	Set of components
	Nc	Natural numbers counter (Initialisée à 0, le nombre de composants est initialement zéro)
	Cp	Component
	Pm	Programs
	Nm	Integer numbers (Initialisée à 0, le nombre de modules de modules programmés est initialement zéro)
	Spm	Set of programmed modules (Initialisée à ϕ, l'ensemble de modules programmés est initialement vide)
	S	Software
Types	Dt	Design type
	Sct	Set of components type
	Nt	Natural numbers type
	Ast	Components type
	CtxMt	Component type x Module type
	Int	Integer numbers type
	Smt	Set of modules type
	St	Software type
Fonctions d'arcs	De	Design
	Sc	Set of components
	R	Rest of the set of components
	C	Component
	M	Module
	Sm	Set of modules
	Rs	Resulted software
Gardes	V(Ss)	Sc is empty
	VN(Ss)	Sc is not empty

Table 5.7: Description d'un modèle de tâches d'organisations du groupe de programmation

5	Modèle de tâches d'une organisation élémentaire		
Identificateur		PGOT	
Tâche du système		PT (Table 5.5)	
Organisation		PGO (Table 5.8)	
Tâche substituée		Aucune	
Description de la tâche	Modèle de l'environnement	{Ds , Md, Sm, Sc} U {Ci / i ∈ [1, n]}	
	Modèle de tâches	Informelle	Figure 5.8
		Formelle	Figure 5.9
	Rôles	Identificateur	Description
		H	Head of the programming team
		P1	Programmer 1
		...	...
		Pn	Programmer n
Légende	Acronyme	Description	
Commentaires			

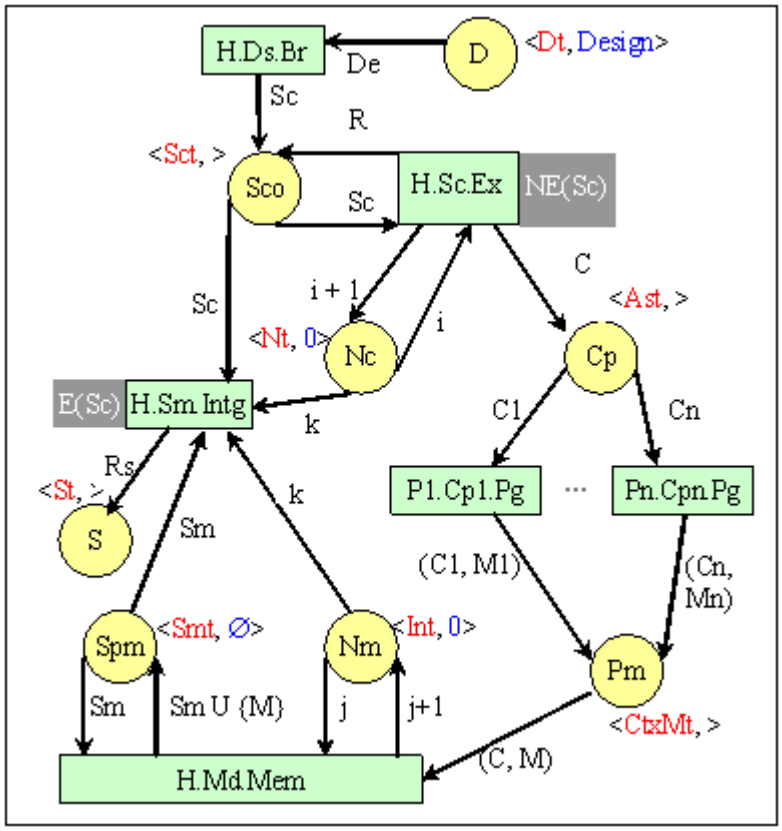

Figure 5.9: Description formelle de la tâche de l'organisation de programmation. La description des éléments de ce CPN est similaire à celle du CPN de la figure 5.7 donnée dans la table 5.6

5.2 Modélisation multi-agent d'un environnement de développement de logiciels

Le développement de logiciel est une tâche complexe qui exige la maîtrise de beaucoup de concepts scientifiques et techniques. Pour cette raison, la majorité des difficultés qui ont amené à la crise de logiciel des années soixante persistent toujours. Un tel développement est fait selon un processus de développement de logiciel qui est un système de comment doit on développer des logiciels. Sa modélisation pour son amélioration, simulation ou automatisation complète ou partielle est une étape très significative dans le sens de la maîtrise de cette complexité. Cette modélisation doit tenir compte de la participation de plusieurs spécialistes. Ces spécialistes incluent ceux du domaine d'application du logiciel, les gestionnaires du développement, les analystes, les concepteurs, les programmeurs, ... Ces spécialistes travaillent sous des contraintes telles qu'ils :

1. Raisonnent de manières différentes,
2. Utilisent des connaissances variées,
3. Ont des comportements hétérogènes,
4. Sont spatialement distribués

Ces spécialistes doivent coopérer sous les contraintes précédentes pour profiter mutuellement de leurs compétences. Les aspects précédents sont ceux des modèles multi-agent. Les modèles traditionnels (fonctionnel, basé données et orientés objet) ne peuvent pas prendre ces conditions d'une manière naturelle. Dans une modélisation multi-agent, les humains et les moyens automatiques seront considérés comme des agents. Ceci amène à un modèle de SMA avec des agents hétérogènes. L'intégration des agents humains et des agents logiciels est basée sur la coopération. Une telle coopération est basée sur la communication. L'évolution des moyens automatiques et du processus logiciel devient une évolution du SMA. Une modélisation multi-agent du processus logiciel permet alors de mieux résoudre ses difficultés.

Dans cette section, le logiciel "processus logiciel" est considéré comme un système multi-agent. Son développement est systématisé en employant la méthodologie de développement de SMA MASA-Method. Cette section a alors un triple objectif :

1. Modélisation du processus logiciel :
2. Illustrer les aspects de MASA-Method,

Table 5.8: Description d'un modèle d'une organisation du groupe de programmation

6	Modèle d'une organisation élémentaire		
Identificateur	PGO		
Organisation mère	SDO		
Tâche	PGOT (Table 5.7)		
Organisation substituée	Aucune		
Rôles	Identificateur	Modèle	
	H	HM (Table 5.10)	
	P1	PM (Table 5.9)	
	...		
	Pn	PM (Table 5.9)	
Relation de communication	Graphique	Analytique	
		Emetteur	Récepteur
	Figure 5.10	H	P1
		...	...
		H	Pn
		P1	H
		...	...
		Pn	H
Légende	Identificateur	Description	
	PGO	Programming group organization	
	SDO	Software development organization	
	PGO-GT	Programming group organization-Global task	
	H	Head of the programming team	
	P_i, $i \in [1, n]$	Programmers roles from 1 to n	
	HM	Programmer chief model	
	PM	Programmer model	
Commentaires	1. Cette sous-organisation est également une organisation qui sera utilisée en tant qu'exemple d'une organisation de laquelle nous dérivons les modèles multi-agent et son implémentation dans les phases suivantes de la méthodologie de MASA-Method. 2. Dans cette illustration, le nombre de programmeurs n'a pas de signification particulière.		

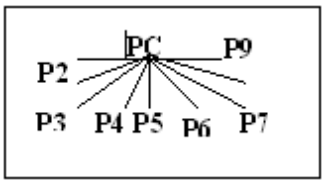

Figure 5.10: Représentation graphique de la relation de communication entre les rôles de l'organisation de l'équipe de programmation

3. Montrer comment pourra-on employer MASA-Method.

Cette section décrit principalement l'approche utilisée pour la modélisation du processus logiciel. Elle présentera les organisations composées et considèrent les développeurs comme des agents assumant des rôles dans l'organisation de développement de logiciels. L'intégration des humains dans des SMA incluant des agents logiciels a été faite selon l'approche basée coopération de la méthodologie MASA-Method [Lahlouhi 02b].

Les sous-sections suivantes montrent que ces aspects peuvent être pris en considération convenablement par un modèle de SMA. Un processus de développement de logiciel est vu alors comme un ensemble de communautés de diverses natures. Ces communautés constituent des sous organisations différentes d'une organisation globale qui est celle de développement de logiciels. Pour notre illustration, on se limitera à l'organisation technique du développement de logiciels. Une telle organisation peut être vue comme étant composée de trois sous organisations comprenant celles d'analyse, de conception et d'implémentation.

La proposition d'un modèle de SMA pour le développement de logiciel comprenant toutes les étapes de développement de logiciels est une tâche qui est, non seulement, difficile à réaliser mais, dans l'état actuel des choses, elle est impossible. Une telle impossibilité est la conséquence du fait que les activités de développement n'ont pas reçues le consensus minimal nécessaire dans la communauté du génie logiciel. En plus, avec l'évolution continue des concepts dans le domaine du génie logiciel, ces activités changent fréquemment. Nous sommes convaincus alors qu'un modèle de processus de développement de logiciels ne peut qu'être adaptatif. Cependant, la modélisation du processus logiciel par un SMA adaptatif est hors de portée de cette thèse. Cependant, une modélisation du processus logiciel par un modèle de SMA classique constitue déjà une avancée dans ce domaine. En particulier, la prise en considération de façon cohérente des interventions des développeurs est une originalité.

Le reste de cette section explique l'approche de modélisation multi-agent du processus logiciel en utilisant MASA-Method. Il est organisé en trois sous-sections. La

sous-section 5.2.1 présente l'organisation composée du développement de logiciel et détaillera, ensuite, l'organisation élémentaire de l'implémentation de logiciels et, la sous-section 5.2.2 décrit la dérivation d'un modèle multi-agent à partir de cette organisation élémentaire.

5.2.1 Organisation de développement de logiciels

Le développement de logiciels peut être vu comme un système dont la description de la tâche est donnée dans la table 5.1-Page 113. Les tâches élémentaires de la tâche de développement de logiciels seront des tâches de trois sous organisations. Le modèle de tâches pour cette organisation est décrite dans la Table 5.3-Page 115. La description d'un modèle d'organisations de développement de logiciels est donnée dans la table 5.4-Page 117. Il est nécessaire de détailler, ensuite, chaque sous organisation. La sous-organisation de programmation est choisie pour être détaillées. La sous-organisation de programmation est une organisation élémentaire. Cette organisation sera utilisée, par la suite, pour illustrer les prochaines étapes de développement d'un modèle de SMA. Ce modèle est celui d'implémentation de modèles conceptuels. Le modèle de tâches du système de programmation est fourni dans la Table 5.5-Page 118, le modèle de tâches de l'organisation de programmation est donné dans la table 5.7-Page 121 et, enfin, le modèle d'organisations de programmation est donné dans la Table 5.8-Page 124. Les modèles de rôles sont décrits dans les Tables 5.9-Page 127 et 5.10-Page 129. La substitution de la tâche de l'organisation composée est décrite dans la Table 5.11-Page 130 et la substitution de l'organisation composée de développement de logiciels est présenté dans la Table 5.12-Page 131.

Notons que les formalismes utilisés pour exprimer les différentes tables de MASA-Method sont autosuffisantes. La majorité des explications sont contenues dans ces tables qui n'exigent pas d'explications complémentaires. Pour cette raison, le reste de cette section ne comporte pas du texte. Il est constitué principalement des tables décrivant le développement d'un modèle de SMA.

5.2.2 Développement d'un modèle multi-agent de développement de logiciels

Cette étape commence par la dérivation d'un modèle multi-agent. Ceci concerne la description du modèle multi-agent décrit dans la Table 5.13-Page 133. Les descriptions des modèles d'agents seront présentés dans les Tables 5.14-Page 134 et 5.15-Page 135.

Table 5.9: Description d'un modèle de rôles PM

7	Modèle de rôles		
Identificateur	PM		
Organisation	PGO		
Modèle de l'environnement	{C, M}		
Tâche	PMT (Figure 5.11)		
Savoir-faire	Identificateur	Paramètres	Description
	Cp.Pg	Input: C, L, Output: M	Transforme un composant C d'une conception en module M de programme dans un langage L de programmation
Légende	Symbole	Description	
	C	Component	
	M	Module	
	Cp	Component	
	L	Language	
Commentaires			

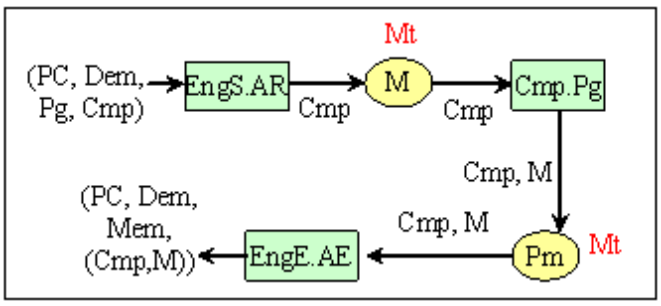

Figure 5.11: Description formelle de la tâche du modèle de rôles PM

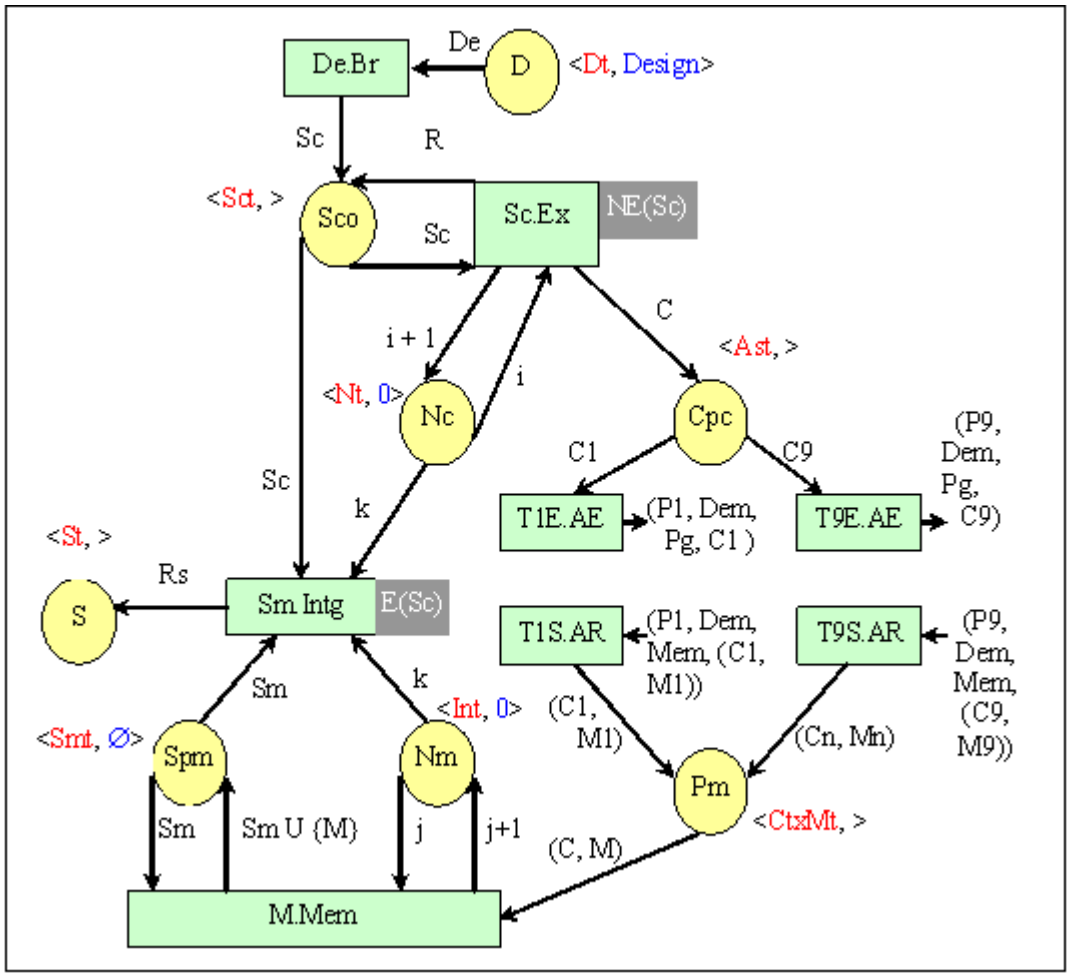

Figure 5.12: La description formelle de la tâche du modèle de rôles HM

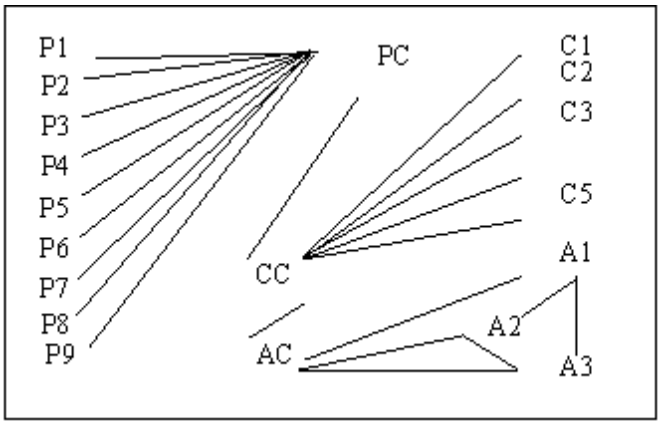

Figure 5.13: Représentation graphique de la relation de communication entre les rôles de l'organisation SDO-S de la table 5.12

Table 5.10: Description d'un modèle de rôles HM

8	Modèle de rôles		
Identificateur	HM		
Organisation	PGO		
Modèle de l'environnement	{Ds, Sc, C, M, Sm}		
Tâche	HMT (Figure 5.12)		
Savoir-faire	Identificateur	Paramètres	Description
	Ds.Br	Inputs: D, Outputs: Sc	Décompose une conception D en un ensemble de composants Sc
	Sc.Ex	Inputs: Sc, Outputs: C, Number of extracted Cs	Retire un composant C d'un ensemble de composants Sc
	Md.Mem	Inputs: A couple (C, M), Set of couples (C, M), Outputs: Set of couples (C, M))	Ajoute un module à l'ensemble des modules programmés sous forme de couples (Composant, Module)
	Sm.Intg	Inputs: Set of couples (C, M), Outputs: Software system	Intègre les modules programmés pour constituer le logiciel résultat
Légende	Symbole	Description	
Commentaires			

Table 5.11: Description d'un modèle de tâches de la substitution du modèle de tâches de l'organisations SDO

9	Modèle de tâches d'organisations élémentaires		
Identificateur	SDOT-S		
Tâche du système	SDT (Figure 5.1)		
Organisation	SDO-S (Table 5.12)		
Tâche substituée	SDOT (Table 5.3)		
Description de la tâche	Modèle de l'environnement	{Ds , Md, Sm, Sc} U {Ci / i ∈ [1, n]}	
	Modèle de tâches	Informelle	Figure 5.12
		Formelle	Figure 5.13
	Rôles	Identificateur	Description
		AC	...
		A1	...
		...	...
		Ak	...
		DC	...
		D1	...
		...	...
		Dj	...
		H	...
		P1	...
		...	...
		Pi	...
Légende	Acronyme	Description	
	...	...	
Commentaires			

(1) . . . ,
(2) // Design a model imitating the specification:
(a) H.D.Br: Décomposer une conception en composants,
(b) Pi.Pg, i ∈ [1, 9]: Programmer un composant et produire un module,
(c) H.Md.M: Mémoriser un module,
(d) H.Sm.Intg: Intégrer les modules selon la conception,
(3) . . . .

Figure 5.14: Description informelle de la substitution de la tâche de l'organisation de développement de logiciels

Table 5.12: Description de la substitution de l'organisation de développement de logiciels

10	Modèle d'une organisation élémentaire		
Identificateur		SDO-S	
Organisation mère		None	
Tâche		SDOT-S (Table 5.11)	
Organisation substituée		SDO	
Rôles	Identificateur	Modèle	
	...	...	
	H	HM (Table 5.10)	
	P1	PM (Table 5.9)	
	...	...	
	Pn	PM (Table 5.9)	
Relation de communication	Graphique	Analytique	
		Emetteur	Récepteur
	Figure 5.13	AC	DC
		DC	H
		AC	A1
		...	...
		AC	Ak
		DC	D1
		...	...
		DC	Dj
		H	P1
		...	...
		H	Pi
Légende	Identificateur	Description	
	...	...	
	PGO	Programming group organization	
	SDO	Software development organization	
	PGO-GT	Programming group organization-Global task	
	H	Programmer chief	
	Pi, i ∈ [1, n]	Programmers roles from 1 to n	
	HM	Programmer chief model	
	PM	Programmer model	
Commentaires	Cette sous-organisation est aussi une organisation. Elle sera utilisée comme un exemple de développement d'un modèle multi-agent.		

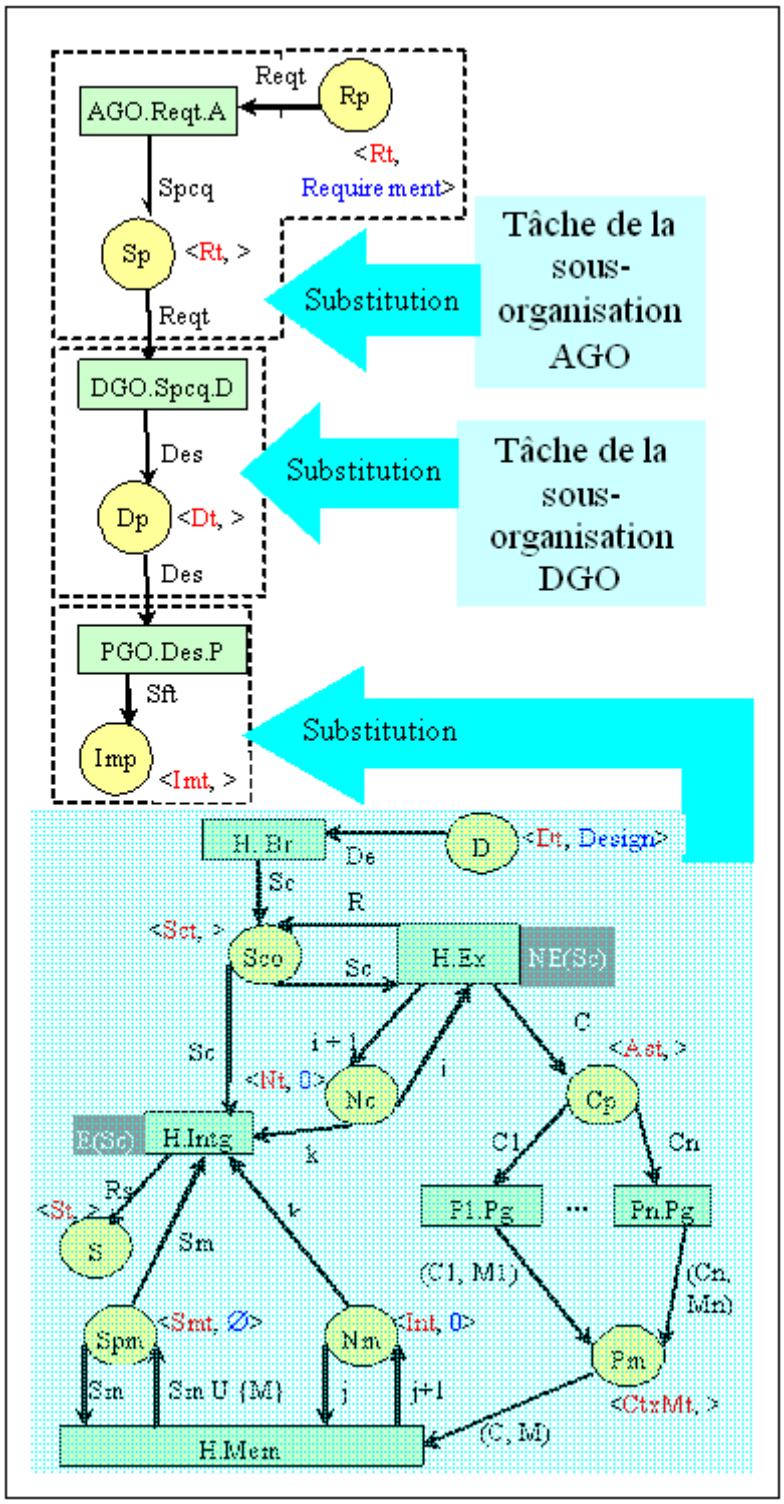

Figure 5.15: Description formelle de la substitution de la tâche de l'organisation de développement de logiciels

Table 5.13: Description d'un modèle de systèmes multi-agent de l'équipe de programmation

11	Modèle de systèmes multi-agent		
Identificateur	PG-MAS		
Organisation	PGO (Table 5.8)		
Attribution de rôles	Identificateur d'agent	Rôles	
	Eng	H	
	T1	P1, .., Pq	
	T2	Pq, ..., Pr	
	T3	Pr, ..., Ps	
Liens de communication	Graphique	Analytique	
		Emetteur	Récepteur
	Figure 5.16	Eng	T1
		Eng	T2
		Eng	T3
Légende	Acronyme	Description	
	Eng	Engineer	
	T1	Technician 1	
	T2	Technician 2	
	T3	Technician 3	
Commentaires			

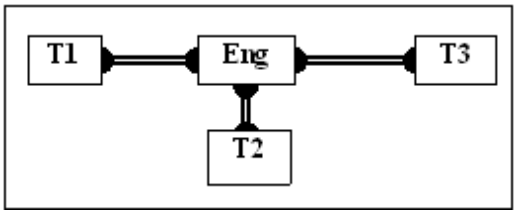

Figure 5.16: Représentation graphique des liens de communication entre les agents du système multi-agent de l'équipe de programmation

Table 5.14: Description du modèle d'agents Eng

12	Modèle d'agents		
Identificateur		Eng	
Modèle de systèmes multi-agent		PG-MAS	
Contrôle		Intelligent	
Rôles		H	
Communications	Senseurs	EngT1S: ..., EngT2S: ...,	EngT3S: ...
	Effecteurs	EngT1E: ..., EngT2E: ...,	EngT3E: ...
	Protocoles		
Relation avec l'environnement	Objets	Ds: ..., Sc: ..., C: ...,	M: ..., Sm: ...
	Savoir-faire	Ds.Br; ..., Sc.Ex: ...,	Md.Mem : ..., Sm.Intg : ...
	Senseurs	Clavier : ..., Souri : ...,	Disque : ...
	Effecteurs	Ecran : ...,	Disque : ...
Légende	Acronyme	Description	
	EngTiS	Senseur de communication de l'agent Eng avec le technicien Ti	
	EngTiE	Effecteur de communication de l'agent Eng avec le technicien Ti	
Commentaires	Les protocoles de communication seront décrits selon les directives données dans le chapitre 4		

Table 5.15: Description du modèle d'agents Ti

13	Modèle d'agent	
Identificateur		Ti
Modèle de systèmes multi-agent		PG-MAS
Contrôle		Procédural
Rôles		Pi, ..., Pj
Communications	Senseurs	TiEngS
	Effecteurs	TiEngE
	Protocoles	
Relation avec l'environnement	Objects	C: ..., M : ...
	Savoir-faire	Cp.Pg : ...
	Senseurs	Disque : ...
	Effecteurs	Disque : ...
Légende	Acronyme	Description
	TiEngS	Senseur de communication avec l'agent Eng
	TiEngE	effecteur de communication avec l'agent Eng
Commentaires	Les protocoles de communication seront décrits selon les directives données dans le chapitre 4	

5.3 Conclusion

Le système multi-agent du processus de programmation a été implémenté dans l'interaction des agents, leur coordination, et agents d'interface pour l'intégration de l'humain dans un système multi-agent. Les modèles conceptuels utilisés dans ces implémentations sont ceux de MERISE. Les règles de passage des modèles MERISE vers des programmes ont été implémentées. De telles réalisations ont été faites en utilisant des versions antérieures de MASA-Method. Ces réalisations avaient comme objectif d'expérimenter la méthodologie MASA-Method.

Pour ce qui est de la modélisation du processus logiciel, il faut noter la grande utilité :

1. Des sous organisations dans la modélisation du processus logiciel,
2. Des agents d'interface pour supporter les activités non automatisables du processus logiciel d'une manière cohérente.

Dans le cadre du développement d'environnements de développement de logiciels, nos efforts ont été orientés vers un environnement de développement de SMA selon MASA-Method :

1. Un environnement centralisé a été été réalisé. Cet environnement utilise un éditeur développé spécialement pour l'édition des tâches des organisations décrites par des CPN et de leur distribution,
2. Un environnement complètement distribué a été également réalisé. Dans cet environnement, plusieurs développeurs peuvent travailler en coopération à travers un réseau,
3. Finalement, il faut noter qu'un système de synthèse de SMA a été réalisé aussi. Ce système permet de produire un code Java à partir des modèles produits dans l'environnement de développement de SMA selon MASA-Method.

Chapitre 6

Conclusion

Les contribution de cette thèse peuvent être subdivisées en deux classes. Premièrement, la méthodologie développée a, comparée à certaines méthodologies multi-agent existantes, plusieurs avantages. Deuxièmement, la modélisation multi-agent du processus logiciel est faite plus facilement et elle répond à certains critères de modélisation tels qu'une justification claire des décisions prises durant la modélisation. Le reste de cette conclusion décrit le bilan du travail sur la modélisation multi-agent du processus logiciel et sur la méthodologie MASA-Method, et certaines perspectives de nos travaux.

6.1 Bilan du travail sur la modélisation multi-agent du processus logiciel

Tandis que la recherche sur les méthodologies multi-agent continue, ces méthodologies sont actuellement loin de la maturité nécessaire pour un développement cohérent des SMA. En plus, ces méthodologies ne peuvent pas satisfaire nos besoins particuliers concernant la modélisation du processus logiciel. La stratégie consistant à développer une méthodologie pour l'utiliser, ensuite, dans la modélisation du processus logiciel était ainsi un succès.

Le développement d'une méthodologie et son expérimentation dans divers domaines étaient certainement longs. Cependant, le résultat est une méthodologie qui répond mieux à nos besoins en modélisation du processus logiciel. On réalise alors que notre stratégie concernant le développement d'une nouvelle méthodologie est raisonnable puisque, jusqu'ici, les méthodologies développées ne permettent pas de répondre à nos besoins spécifiques.

L'approche de modélisation a été validée dans plusieurs réalisations. Le système multi-agent du processus de programmation a été implémenté dans trois réalisations

différentes (interaction des agents, coordination des agents et les agents d'interface pour l'intégration des humains dans un système multi-agent). Dans la modélisation du processus logiciel, notons la grande utilité des sous organisations et des agents d'interface. Les sous organisations ont permis une modélisation plus cohérente du processus logiciel. Les agents d'interface ont permis de traiter les activités non automatisables du processus logiciel d'une manière cohérente.

Dans le développement des environnements, un environnement de développement de systèmes multi-agent selon MASA-Method a été développé. Cet environnement emploie un éditeur développé spécialement pour l'édition des tâches des organisations (modélisées par des réseaux de Pétri colorés) et de leur distribution. Un environnement complètement distribué où plusieurs développeurs peuvent coopérer à travers un réseau est également développé. Finalement, un système de synthèse de SMA a été développé. Ce système permet de produire un code Java à partir des modèles générés dans l'environnement de développement de systèmes multi-agent selon MASA-Method.

Les travaux relatifs à la modélisation du processus logiciel utilisent des modèles classiques (fonctionnel, basés données, orientée objet, intelligence artificielle) ou des modélisations empiriques. Dans cette thèse, le décalage a été fait en utilisant une approche méthodologique multi-agent. De plus, le support de l'intervention des humains est fait également de façon méthodologique en utilisant l'approche basée coopération de MASA-Method. Finalement, la structuration organisationnelle du développement de logiciels et, en particulier, la structuration de cette organisation en sous organisations a conduit à une modularité naturelle.

Finalement, il faut noter que la modélisation totale du processus logiciel ne peut être réalisée sans la maîtrise complète des activités faites dans le processus logiciel. Cependant, actuellement ces activités sont soumises, non seulement à une évolution simple mais, à des bouleversements complets. Cette controverse ne peut être maîtrisée tant que ces activités n'ont pas atteint une certaine maturité pour qu'on puisse savoir l'objet de la modélisation. Il n'est pas suffisant d'utiliser alors une modélisation multi-agent classique alors que le sujet de la modélisation évolue continuellement. Ceci amène à penser aux systèmes multi-agent adaptatifs. Cependant, MASA-Method ne permet pas de développer de tels systèmes. Il est alors nécessaire d'évoluer MASA-Method pour qu'elle permette de développer des systèmes multi-agent adaptatifs. Un nouveau concept, appelé "Sociétés méthodologiques", a été alors introduit dans [Lahlouhi 02c]. Ce concept consiste en un développement de systèmes (en particulier, des SMA ou sociétés) enfermant la méthodologie de leur développement. Ceci permettra à de tels systèmes d'évoluer selon les méthodologies de leur développement.

6.2 Bilan du travail sur la méthodologie MASA-Method

La fondation d'une nouvelle méthodologie n'est pas une question simple et la justification des connaissances méthodologique n'est pas facilement accessible. En outre, l'expérimentation de la méthodologie MASA-Method dans le développement de systèmes multi-agent a nécessité une longue période. De telles difficultés sont particulièrement plus accrues parce que ces systèmes doivent être développés en utilisant une méthodologie expérimentale où elle exige des feedbacks entre sa correction, son amélioration et son utilisation sans assurer le résultat final. Malgré ces difficultés, nos efforts ont réussis à développer la méthodologie MASA-Method. Ce travail a aboutit à des réalisations pratiques concernant des domaines diversifiés :

1. Le processus logiciel,
2. Simulation multi-agent de la coopération de robots autonomes mobiles,
3. Système multi-agent avec des agents hétérogènes (robot, humain et logiciel),
4. Agents mobiles.

L'utilisation de MASA-Method dans ces réalisations a amené à la proposition de nouvelles approches. Ceci est le cas pour les agents mobiles, pour les sociétés hétérogènes et pour le processus logiciel. Elles ont permis également d'améliorer MASA-Method. Comparé à d'autres méthodologies existantes, elle présente certains avantages. Le reste de cette section décrit certaines de ces contributions et compare MASA-Method à certaines méthodologies.

6.2.1 Contribution

MASA-Method est suffisamment générale pour qu'elle puisse être utilisée dans le développement de systèmes multi-agent de différentes sortes. Elle est également suffisamment spécifique pour qu'elle puisse tenir compte des spécificités de la problématique de cette thèse qui est celle de modélisation du processus logiciel. Elle permet de supporter les agents humains et de structurer le processus logiciel en sous organisations. D'après la classification donnée dans [Lahlouhi 02c], MASA-Method est une méthodologie mixte et complète. MASA-Method a permis de dépasser plusieurs limites des méthodologies actuelles. Du point de vue global, elle intègre les exigences fondamentales d'une méthodologie qui sont le méta-modèle et le processus méthodologique. De plus, elle intègre un formalisme qui est un aspect pratique significatif et préférable.

Du point de vue formalisme, les diagrammes utilisés dans certaines méthodologies manquent des aspects de coordination. Ces aspects sont liés à la communication entre

les agents et à la synchronisation de leurs activités qui sont des propriétés importantes des SMA. MASA-Method utilise les CPN qui sont un formalisme puissant pour la modélisation de la synchronisation.

Du point de vue méta-modèle, MASA-Method a proposé deux principales originalités. Premièrement, son processus méthodologique a considéré la coopération dès le début du développement au niveau organisationnel en explicitant le développement de l'objectif de l'organisation. Ceci a permis un développement des organisations dirigé par le problème à résoudre. Il a permis également de structurer les organisations en sous organisations. Ce qui est une modularité significative dans le développement des organisations. La deuxième originalité est celle de l'approche basée coopération qui permet de développer des systèmes multi-agent intégrant des agents hétérogènes (humains et logiciels).

Du point de vue processus méthodologique, MASA-Method est complète et propose une approche mixte du développement permettant de créer de nouveaux agents et/ou de réutiliser des capacités des agents existants qu'ils soient logiciels ou humains. MASA-Method utilise l'approche basée coopération (proposée dans [Lahlouhi 02b]) pour l'interaction des humains et des agents logiciel. Cette approche a plusieurs avantages. Pour une méthodologie multi-agent, en général, et pour MASA-Method, en particulier, l'adoption de l'approche basée coopération permet de fonder un processus méthodologique cohérent en évitant aux développeurs de faire des changements incohérents de paradigmes. Les agents d'interface, développés selon l'approche basée coopération, permettent également l'intégration des agents humains dans les systèmes multi-agent.

Enfin, du point de vue plus spécifique, MASA-Method est la seule méthodologie qui intègre la détection de la terminaison des SMA. Elle permet aux développeurs d'éviter des incohérences dans les réalisations de leurs modèles de SMA.

6.2.2 Comparaison avec des travaux similaires

Cette sous-section compare MASA-Method à certaines méthodologies bien connues et plus similaires à MASA-Method. Nous résumons certains aspects de cette comparaison dans la Table 6.1-Page 141. Cette table complète la table 2.2-Page 33 donnée dans le chapitre 2. Trois méthodologies considérées comme proches de MASA-Method ont été choisies pour cette comparaison :

1. La méthodologie du projet ALAADDIN [Ferber 98, Ferber 01, Gutknecht 01],
2. Gaia [Wooldridge 00, Zambonelli 00b],
3. MaSE [DeLoach 01a, DeLoach 02b, DeLoach 00, Wood 00, DeLoach 02a].

Table 6.1: Table récapitutive de la comparaison des méthodologies Alaaddin, Gaia et MaSE avec la méthodologie MASA-Method

	Alaaddin	Gaia	MaSE	MASA-Method
Processus	Ascendant, Organisat-ionnel	Ascendant, Organisat-ionnel	Descendant, Complet	Mixt, Complet
Structures Organization-nelles	Agent, groupe et rôle	Basées sur les rôles	Basées sur les rôles	Organisations composées et tâche du système
Consideration de la coordination	Non Traitée	Niveau Organisat ionnel	Niveau modèle multi-agent	Niveau organisationnel
Considération de la cooperation	Niveau modèle multi-agent	Niveau modèle multi-agent	Niveau modèle multi-agent	Niveau organisationnel
Agents hétérogènes	Non Traités	Non Traités	Empirique	Approache basée cooperation
Diagrams	Empirique	Empirique	hierarchie et/ou, diagrammes UML, (relation non claire entre eux)	CPN et CPN synchronisés (Relation claire entre eux)
Divers	La seule méthodologie incluant méta-modèle défini formellement	La méthodologie la plus publiée et référencée	Méthodologie Complète, (Appliquée aux robots et aux agents mobile)	La seule méthodologie mixte

La méthodologie du projet ALAADDIN

La méthodologie du projet Alaaddin est composée d'un méta-modèle et d'un processus méthodologique [Ferber 98, Ferber 01, Gutknecht 01]. C'est une méthodologie organisationnelle ascendante. Le modèle organisationnel de ALAADDIN est basé sur les concepts d'agent, de groupe et de rôle. La structure du groupe est un ensemble de rôles et d'interactions parmi eux offrant un contexte commun et rationnel [Ferber 01]. Elle est également définie comme étant l'agrégat des agents et puis comme ensemble de rôles, de graphes d'interaction et de langage d'interaction [Ferber 01]. La structure d'organisation est un ensemble de structures de groupes [Ferber 01].

Dans ALAADDIN, le concept de groupe est seulement un groupement de rôles sans indiquer le critère de groupement particulier. Ceci ne permet pas de diriger la création des groupes. La notion de groupe peut être considérée comme celle d'une sous organisation dans MASA-Method. La différence entre eux est qu'une sous organisation a un objectif (tâche) qui peut être considérée comme un critère de groupement.

Finalement, contrairement à MASA-Method, ALAADDIN ne tient pas compte de la coordination dans le développement de l'organisation.

La méthodologie Gaia et sa révision

Gaia [Wooldridge 00] est une méthodologie organisationnelle ascendante. Gaia est basé sur des définitions des différents concepts et des processus méthodologiques qui permettent de les établir. Sa révision [Zambonelli 00b] a clarifié la nécessité de tenir compte de la coordination au niveau de l'organisation. C'est une étape méthodologique significative comparée à d'autres méthodologies organisationnelles. Cependant, MASA-Method prend une avancée sur Gaia en considérant au niveau organisationnel non seulement la coordination mais également la coopération. Gaia ne décrit pas explicitement les modèles qu'elle utilise, c'est-à-dire les différents formalismes. Ceci ne permet pas d'avoir des spécifications précises à implémenter. Ce qui n'est pas le cas de MASA-Method qui décrit ces modèles explicitement en partant de la fondation de l'organisation jusqu'à l'implémentation du système multi-agent.

La méthodologie MaSE et son extension

MaSE [DeLoach 01a] est une méthodologie complète descendante. Elle se compose d'un méta-modèle et d'un processus méthodologique global détaillé. Le grand reproche, qu'on peut faire à la méthodologie MaSE, est qu'elle est trop orientée objet. Elle emploie les principaux diagrammes des méthodologies orientées objet au niveau de la conception, en particulier, ceux du méta-modèle UML. Ses auteurs adoptent ceci

comme objectif pour MaSE. L'utilisation de l'approche orientée objet, dans MASA-Method, est laissé à la phase d'implémentation où elle emploie les éléments du modèle orienté objet dans les aspects d'implémentation des systèmes d'agents seulement. L'introduction de la description de l'objectif du système [DeLoach 01a, DeLoach 02b, DeLoach 00] dans MaSE à l'étape d'analyse permet de tenir compte de la coopération d'une manière plus cohérente telle qu'elle est exprimée dans les besoins du système. Cependant, la relation entre cette description et les autres parties du système développées dans les étapes qui la suivent n'est pas claire. Ce qui n'est pas le cas de MASA-Method.

En outre, cette description est une hiérarchie "et/ou". Ce qui signifie que l'accomplissement d'un objectif se fait par l'accomplissement de tous ses sous objectifs (relation "et") ou, au moins, d'un objectif d'entre eux (relation "ou"). Cette description serait meilleure si elle était faite au niveau organisationnelle, comme dans MASA-Method, et non pas au niveau modélisation multi-agent. Notons, enfin, que les diagrammes des objectifs de MaSE ne permettent pas la dérivation de la coordination entre les réalisations des sous objectifs.

6.3 Travaux futures

En dépit des efforts consentis dans sa fondation rigoureuse, MASA-Method présente encore des insuffisances comme c'est le cas des autres méthodologies multi-agent proposées jusqu'à présent. Les premières perspectives sont liées alors directement à la méthodologie. La première d'entre elles est celle de la formalisation des divers aspects de MASA-Method et du développement de leurs algorithmes et de leur implémentation. Ceci permettra de simplifier plus les tâches des développeurs. Si l'implémentation de tels algorithmes doit être faite dans un environnement de développement de systèmes multi-agent, le développement des algorithmes doit être fait dans la méthodologie. Ce développement doit être basé sur les connaissances méthodologiques qui doivent être rigoureusement justifiées et vérifiées.

La deuxième perspective est en relation avec les systèmes multi-agent adaptatifs (systèmes multi-agent ouverts et auto organisés et, agents adaptatifs). Les agents adaptatifs sont un peu loin de MASA-Method qui adopte une approche d'intégration d'agents hétérogènes. L'adaptation de chaque modèle d'agents peut être différente de celle des autres. Cependant, les systèmes multi-agent ouverts et auto organisés peuvent être traités dans MASA-Method en tant que "Sociétés méthodologiques". Les sociétés méthodologiques considèrent les méthodologies comme systèmes auxquels on peut appliquer les méthodes et les techniques utilisées dans les traitements des systèmes (modélisation, simulation et conception). La manipulation des "sociétés méthodologiques"

est facilitée par l'utilisation des algorithmes de dérivation des tâches de rôles (relation modèle multi-agent avec l'organisation) et de synthèse de SMA (relation modèle multi-agent avec l'implémentation). Certains de ces travaux ont déjà initiés comme la modélisation des méthodologies comme systèmes et l'automatisation de certaines activités faites dans l'environnement. Là où l'automation entière n'est pas encore possible, une automatisation partielle peut être faite. Cette automatisation partielle sera complétée par des interventions des humains qui seront coordonnés en utilisant l'approche basée coopération de MASA-Method.

Bibliographie

[Alloui 95] Alloui, I. et F. Oquendo, "PEACE+: Une approche intentionnelle pour le support à la coopération dans les environnements de génie logiciel assisté par ordinateur", 3ème journées francophones IAD et SMA, Chambéry, St Baldoph, 1995.

[Alloui 96] Alloui, I., "Peace+ : un formalisme et un système pour la coopération dans les environnements de génie logiciel centrés processus", Thèse de Doctorat, Université Pierre Mendès France, Grenoble, France, Septembre 1996.

[Amiour 97] Amiour M., "A Support for cooperation in software processes", 4th Doctoral Consortium of CAiSE'97, Barcelona, Spain, 1997

[Amiour 98] Amiour, M. and J. Estublier, "A Support for Communication in Software Processes". Proc. of SEKE'98 (Software Engineering and Knowledge Engineering), San Francisco, CA, June 1998.

[Amiour 99] Amiour M., "Vers une fédération de composants interopérables pour les environnements centrés procédés logiciels", Thèse de docteur en Informatique de l'université Joseph Fourrier, Grenoble I, 1999

[Arnold 92] Arnold, J. E. and G. Memmi, "Control integration and its role in software integration", 5th International conference on software engineering and its applications, Toulouse, France, December 7-11, 1992.

[Ashby 56] Ashby, W. R., "An introduction to cybernetics", Chapman & Hall, London, 1956.

[Babaoglu 02] Babaoglu, O., H. Meling, and A. Montresor. Anthill, "A framework for the development of agent-based peer-to-peer systems", In Proceedings of the 22nd International Conference on Distributed Computing Systems, pages 15–22. IEEE CS Press, Vienna (A), July 2002.

[Balzer 00] Balzer, Robert, "Keynote on current state and future perspectives of software process technology", In Reidar Conradi, editor, Proc. 8th European Software Process Workshop on Software Process Technology (EWSPT'2000), page 220, Kaprun (Salzburg), Austria, February 21-25 2000, Springer Verlag LNCS 1780.

[Belkhatir 91a] Belkhatir, N., J. Estublier and W. L. Melo, "Activity coordination in Adele: a software production kernel", Proc of the 7th International software process Workshop, IEEE Computer Society Press, San Francisco, CA, October 16-18, 1991.

[Belkhatir 91b] Belkhatir, N., J. Estublier, M. A. Nacer, W. L. Melo, "CASE for programming-in-the-large", Research Report, LGI, Grenoble, December 1st, 1991.

[Bernon 02a] Bernon, C., M.-P. Gleizes, G. Picard, P. Glize, "The ADELFE methodology for an intranet system design", In proceedings of the fourth international bi-conference workshop on agent-oriented information systems (AOIS 2002 at CaiSE'02) Toronto (Ontario, Canada), May 27-28, 2002.

[Bernon 02b] Bernon C., M.-P. Gleizes, S. Peyruqueou, and G. Picard, "ADELFE: a Methodology for adaptive multi-agent systems engineering", In Petta, P. and Tolksdorf, R. and Zambonelli, F., editor, Third International Workshop on Engineering Societies in the Agents World (ESAW-2002). Springer-Verlag (LNAI 2577), 2002.

[Bernon 01] Bernon C., Camps V., Marie Pierre Gleizes, Pierre Glize, "La conception de systèmes multi-agents adaptatifs : contraintes et spécificités", Atelier de Méthodologie et Environnements pour les Systèmes Multi-Agents (SMA 2001), Plate-forme AFIA, Grenoble 25-28 jun 2001.

[Bresciani 01] Bresciani, Paolo, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos, "Modeling early requirements in Tropos: a transformation based approach", In M. Wooldridge, P. Ciancarini, and G.Weiss, editors, Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001), 2001.

[Bush 01] Bush G., S. Cranefield and M. Purvis, "The Styx Agent Methodology", The Information Science Discussion Paper Series 2001/02, Department of Information Science, University of Otago, New Zealand., Jan. 2001.

[Caire 01] Caire G., G., J. Pavon, F. L., P. Chainho, P. Kearney, J. Stark, R. Evans, and P. Massonet, "Agent oriented analysis using MESSAGE/UML", In M. Wooldridge, P. Ciancarini, and G.Weiss, editors, Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001), pages 101—108, 2001.

[Ciancarini 93] Ciancarini, P., "Coordination rule-based software processes with ESP", ACM TSEM, Vol. 2, n°3, July 1993.

[Ciancarini 98] Ciancarini, P., R. Tolksdorf, F. Vitali, D. Rossi, A. Knoche, "Coordinating Multi-agents Applications on the WWW: A Reference Architecture", IEEE TSE, Vol. 24, No. 5, May 1998.

[Collinot 96a] Collinot, A., Ploix L., Drogoul A., "Application de la méthode CASSIOPEE à l'organisation d'une équipe de robots", Intelligence Artificielle Distribuée et Systèmes Multi-Agents, Proceedings of JFIADSMA'96, J.P. Müller and J. Quinqueton (Eds), pp. 136-152, Hermès, Paris, 1996.

[Collinot 96b] Collinot A., A. Drogoul, and P. Benhamou, "Agent-oriented design of a soccer robot team", In Proceedings of the 2nd International Conference on Multi-agent Systems, pages 41–47. IEEE CS Press, Kyoto (J), 1996.

[Conradi 98] Conradi, Reidar, Alfonso Fugetta, and Mari Letizia Jaccheri, "Six theses on software process research", In Volker Gruhn, editor, Software Process Technology, 6th European Workshop (EWSPT'98), pages 100–104, Weybridge, UK, September 16-18 1998. Springer Verlag LNCS 1487.

[Cugola 98] Cugola, Gianpaulo and Carlo Ghezzi, "Software Processes: a retrospective and a path to the future", SOFTWARE PROCESS – Improvement and Practice, 4(2):101–123, 1998.

[Dawson 87] Dawson, M., "Integrated project support with Istar"; IEEE Software, Vol. 4, N° 11, November 1987, pages: 6-15.

[DeLoach 02a] Deloach, S. A., "Modeling Organizational Rules in the Multi-agent Systems Engineering Methodology", R. Cohen and B. Spencer (Eds.), LNAI, AI 2002, LNAI 1738, Springer-Verlag Berlin Heidelberg, pp. 1-15, 2002.

[DeLoach 01a] Deloach, S. A., Wood M. F., Sparkman C. H., "Multiagent systems engineering", The International Journal of Software Engineering and Knowledge Engineering, 11(3), pp 171–258, 2001.

[DeLoach 02b] DeLoach, Scott A., Eric T. Matson, and Yonghua Li, "Applying Agent Oriented Software Engineering to Cooperative Robotics", Proceedings of the 15th International FLAIRS Conference (FLAIRS 2002). Pensacola, Florida. May 16-18, 2002.

[DeLoach 00] DeLoach, Scott A. and Mark Wood, "Developing Multiagent Systems with agentTool", in Y. Lesperance and C. Castelfranchi, editors, Intelligent Agents VII - Proceedings of the 7th International Workshop on Agent Theories, Architectures, and Languages (ATAL'2000). Springer Lecture Notes in AI, Springer Verlag, Berlin, 2001.

[Dijkstra 83] Dijkstra, E.W., Feijen, W.H.J., van Gasteren, A.J.M., "Derivation of a termination detection algorithm for distributed computations", Information Processing Letters 16 (1983) 217–219

[Dixon 88] D. Dixon, "Integrated support for project management", proceedings of the 10th International conference in software engineering, IEEE, 1988, pages 49-58.

[Drogoul 00] Drogoul, Alexis, "Systèmes multi-agents situés", Accreditation file to supervise researches, from the Pierre and Marie Curia University, Compuer science speciality, 2000

[Elammari 99] Elammari, M., W. Lalonde, "An agent-oriented methodology: high-level and intermediate models", Proceedings of AOIS 1999 (Agent-Oriented Information Systems), Heidelberg (Germany), 14-15 June 1999.

[Ferber 98] Ferber, J. and O. Gutknecht, "A meta-model for the analysis and design of rganizations in multi-agent systems", In Proceedings of the 3rd International Conference on Multi-Agent Systems, pages 128–135. IEEE CS Press, Paris (F), June 1998.

[Ferber 01] Ferber, J., O. Gutknecht, C. M. Jonker, J.-P. Müller, and J. Treur, "Organization Models and Behavioural Requirements Specification for Multi-Agent Systems", In proceedingd of 10th European workshop on MAAMAW (MAAMAW'01), Annecy, France, 2001

[Gervais 01] Gervais, M.-P., "Méthodologie ODAC : Le guide de spécification comportementale", Rapport de recherche, LIP6, Number 2001 / 018, 2001

[Gervais 00] Gervais, M.-P. , "ODAC : une méthodologie de construction de systèmes à base d'agents fondée sur ODP", research report, LIP6, Number 2000 / 028, 2000

[Giunchiglia 02] Giunchiglia, F., J. Mylopoulos, and A. Perini, "The Tropos Software Development Methodology: Processes, Models and Diagrams", Third international workshop on agent oriented software engineering (AOSE 2002), Bologna, Italy, July 15, 2002.

[Guessoum 99] Guessoum, Z. and Jean-Pierre Briot From Active Objects to Autonomous Agents, IEEE Concurrency, vol. 7, N° 3, pp. 68-78, July/september, 1999

[Gutknecht 01] Gutknecht O., "Proposition d'un modèle organisationnel générique de systèmes multi-agents Examen de ses conséquences formelles, implémentatoires et méthodologiques", Thèse de Doctorat en informatique de l'université Montpellier II, Soutenue le 14 septembre 2001

[Handy 76] Handy, C., "Understanding Organizations", Penguin Books, London (UK), 1976.

[Hayes-Roth 85] Hayes-Roth, B., "A blackboard architecture for control", Artificial intelligence, 26, Pages 251-321, 1985.

[Iglesias 96] Iglesias, C. A., M. Garigjo, J. C. Gonzalez and J. R. Velasco, "A Methodological Proposal for Multiagent Systems Development Extending CommonKADS", Knowledge Acquisition Workshop (KAW'96), Banff, Canada, 09-14/11/1996

[Iglesias 98] Iglesias, C. A., Garjo M., Gonzalez J. C., "A Survey of Agent-Oriented Methodologies", in Intelligent Agents V. Agents Theories, Architectures, and Languages, Lecture Notes in Computer Science, vol. 1555, J. P. Müller, M. P. Singh, and A. S. Rao (Eds.), Springer-Verlag, 1998.

[Jensen 94] Jensen, K., "An Introduction to the Theoretical Aspects of Coloured Petri Nets", J.W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.), A Decade of Concurrency, Springer-Verlag LNCS vol. 803, pp. 230-272., 1994

[Jensen 96] Jensen, K., "An Introduction to the Practical Use of Coloured Petri Nets", Course material from the Advanced Course on Petri Nets, Dagstuhl Germany, 1996, Lecture Notes in Computer Science, Springer-Verlag.

[Kaiser 87] Kaiser, G. E., P. H. Feiler, "An architecture for intelligent assistance in software development", in 9th International conference on software engineering, IEEE, Monterey, march 30- April 1,1987.

[Kaiser 88] Kaiser, G. E., P. H. Feiler, S. S. Popovich, "Intelligent assistance for software development and maintenance", IEEE Software, Vol. 5, n°3, May 1988.

[Kaiser 90] Kaiser, G.E., N. S. Barghouti, M.H. Socolsky, "Preliminary experience with process modeling in the MARVEL software development environment kernel", Proc. of the 23rd annual Hawaii international conf. on system sciences, IEEE, vol. 11, 1990.

[Kellner 99] Kellner, Marc I., Raymond J. Madachy, and David M. Raffo, "Software Process Simulation Modeling: Why? What? How?"", Kellner, Madachy, and Raffo, "Software Process Modeling and Simulation: Why, What, How," Journal of Systems and Software, Vol. 46, No. 2/3, 15 April 1999.

[Khammaci 91] Khammaci, Tahar, "Contribution à l'étude du processus de développement de logiciels : assistance à base de connaissance et modélisation des objets logiciels", Thèse de doctorat, Université de Nancy I, 1991.

[Khammaci 92] Khammaci, T. and B. El Ayeb, "Software development assistance by reasoning and Planing", 5th International conference on software engineering and its applications, Toulouse, France, December 7-11, 1992.

[Lahlouhi 97] Lahlouhi, Ammar, "Besoins d'une automatisation du processus logiciel: Approche multi-agent cognitifs", Proceedings des deuxième séminaire national en informatique (SNIB'97), 18-19 Novembre 1997.

[Lahlouhi 02a] Lahlouhi, Ammar, Z. Sahnoun, A. Refrafi, S. Azizi, H.-Z. Khelifa, F. Mimi, L. Kahloul and Z. Attaoua, "MASA-Method: A Multi-agent Development methodology" 6th world multi-conference on systemic, cybernetics and informatics (SCI 2002), Software engineer-

ing of multi-agents systems session (SEMAS 2002), Orlando, Florida, July 14-18, 2002.

[Lahlouhi 02b] Lahlouhi, Ammar , Z. Sahnoun, M. L. Benbrahim, and A. Boussaha, "Interface Agents Development in MASA for Human Integration in Multiagent Systems", F. J. Garijo, J. C. Riquelme, and M. Toro (Eds.): IBERAMIA 2002, LNAI 2527 (Springer-Verlag), pp. 566-574, 2002.

[Lahlouhi 02c] Lahlouhi, Ammar and Z. Sahnoun, "Multi-Agent Methodologies' Incoherencies", J. Debenham, B. Henderson-Sellers, N. Jennings & J. Odell (Eds). Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies. ISBN 0-9581915-0-6. COTAR (2002) soft binding.

[Lahlouhi 03a] Lahlouhi, Ammar et Z. Sahnoun, "Incohérences des méthodologies multi-agents", SETIT 2003, Mahdia, 17-21 Mars 2003.

[Lahlouhi 05] Lahlouhi, Ammar and Z. Sahnoun "Multi-agent development methodology: From multi-agent abstraction to object oriented implementation", Information technology journal, volume 4 issue 4, 2005.

[Lashkari 94] Lashkari, Yezdi, Max Metral, Pattie Maes, "Collaborative Interface Agents", Proceedings of AAI '94 Conference, Seattle, Washington, August 1994.

[Lehman 69] Lehman, Manny M., "The Programming Process", Technical Report Rep. RC 2722, IBM Research Centre, Yorktown Heights, NY 10594, September 1969.

[Lesh 98] Lesh, Neal, Charles Rich Candace L. Sidner, "Using plan recognition in human-computer collaboration", TR-98-23 December 1998, MERL-A Mistubishi electric research laboratory, http://www.merl.com

[Lieberman 98] Lieberman, Henry, "Integrating user interface agents with conventional applications", International Conference on Intelligent User Interfaces, San Francisco, January 1998.

[Lieberman 97] Lieberman, Henry, "Autonomous interface agents", In Proceedings of the ACM Conference on Human Factors in Computing Systems, CHI '97 (pp. 67-74). ACM Press, 1997

[Lieberman 01] Lieberman, Henry, Christopher Fry, Louis Weitzman, "Exploring the Web with reconnaissance agents", Communication of the ACM August 2001/Vol. 44, No. 8.

[Lind 99] Lind, J., "A process model for the design of multi-agent systems", German Research Center for AI (DFKI), Technical report number TM-99-03, April 1999

[Maes 93] Maes, P., and Robyn Kozierok, "Learning interface agents", AAAI Conference, 1993.

[Meslati 96] Meslati, D., M.S. Bendelloul et S. Ghoul, "Modélisation de méthodologies de processus logiciels: Approche coordination d'un système d'agents orienté-objet", SNITO'96, Tizi-ouzou 11-13 novembre 1996.

[Metral 93] Metral, Max Edward, "Design of a generic learning interface agent", Department of Electrical Engineering and Computer Science, Bachelor of Science in Computer Science and Engineering, MIT, Thesis supervised by Patricia E. Maes, May 1993.

[Middleton 01] Middleton, Stuart E., "Interface agents: A review of the field", Technical Report Number: ECSTR˜IAM01-001, ISBN: 0854327320, http://www.ecs.soton.ac.uk/˜sem99r, Intelligence, Agents and Multimedia group (IAM group), University of Southampton.

[Mintzberg 79] Mintzberg, H., "The structuring of organizations: A synthesis of the research", Prentice Hall, Englewood Cliffs (NJ), 1979.

[Müller 00] Müller, J.-P., "Modélisation organisationnelle en systèmes multi-agents", 7ème école d'été ARCo, Bonas, july 10-21, 2000.

[Nwana 99] Nwana, H. S., D. T. Ndumu, L. C. Lee and J. C. Collis, "ZEUS: A toolkit for building distributed multi-agent systems", Applied Artificial Intelligence Journal, vol. 1, no. 13, pp. 129-185, 1999.

[Odell 00] Odell, J., Parunak H. V. D., Bauer B., "Extending UML for Agents", Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence (AAAI), 2000.

[Omicini 00] Omicini, A., "SODA: Societies and infrastructures in the analysis and design of agent-based systems", In P. Ciancarini and M. Wooldridge (eds.), Agent-Oriented Software Engineering, Proceedings of the First

International Workshop (AOSE-2000), Springer-Verlag, Berlin, Germany, 2000.

[Osterweil 87] Osterweil, L., "Software processes are software too", In Proc. of the 9th International Conf. on Software Engineering, IEEE Computer Society Press. March 1987.

[Padgham 02] Padgham, L. and M. Winikoff, "Prometheus: A methodology for developing intelligent agents", Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002), Bologna, Italy, July 15-18, 2002.

[Parunak 01] Parunak, H. Van Dyke, Odell James, "Representing social structures in UML", Autonomous Agents'01, May 28-June 1, 2001, Montreal, Canada, 2001.

[Paulk 93] M. C. Paulk, C. V. Weber, B. Curtis, and M. B. Chrissis, "The Capability Maturity Model: Guidelines for Improving the Software Process", Pittsburgh: Addison Wesley, 1993.

[Perrin 92] Perrin, O. et N. Boudjlida, "L'intégration d'outils dans les environnements de développement de logiciels: l'aspect données", 5th International conference on software engineering and its applications, Toulouse, France, December 7-11, 1992.

[Perry 91] Perry, D.E. and G.E. Kaiser, "Models of software development environments", IEEE Transactions on Software Engineering, 17, March 1991.

[Rich 96] Rich, Charles, "Window sharing with collaborative interface agents", SIGCHI Bulletin, Vol. 28, No. 1, January 1996, pp. 70-78.

[Rich 97a] Rich, Charles, Candace L. Sidner, "Segmented interaction history in a collaborative interface agent", Third Int. Conf. on Intelligent User Interfaces, Orlando, FL, January 1997.

[Rich 97b] Rich, Charles, Candace L. Sidner, "COLLAGEN: When agents collaborate with people", First Int. Conf. on Autonomous Agents, Marina del Rey, CA, February 1997.

[Rich 98] Rich, Charles, Candace L. Sidner, "COLLAGEN: A collaboration manager for software interface agents", TR-97-21a

March 1998, MERL-A Mistubishi electric research laboratory, http://www.merl.com

[Rochfelo 83] Rochfelo, A., and Tardieu, H., "MERISE: An information system design and development methodology", Information Management, volume 6, pages 143-159, 1983

[Ross 85] Ross, D. T. "Applications and extensions of SADT", Computer 3 (4), pages 25-34, 1985.

[Sahnoun 97] [Sahnoun 97] Sahnoun, Z., M. Boufaida, P. Barril "Multi-Agent Based Model for the Software Engineering Process", International Conference on Software Quality Engineering (SQE97), Editors : C. Tasso, R.A. Adey, M.Pighin, ISBN 1 85312 403 6, Computational Mechanics Publications, PP 103- 112, Udine, Itlay, 1997

[Sanz 02] Gomez-Sanz, J. J. , J. Pavon, "Agent oriented software engineering with MESSAGE", Proceedings of the fourth international biconference workshop on agent-oriented information systems (AOIS 2002 at CaiSE'02) Toronto (Ontario, Canada), May 27-28, 2002.

[Searle 69] Searle, J. R., "Speechs acts", Cambridge university press, 1969.

[Sparkman 01] Sparkman, Clint H., "Transforming analysis models into design models for the multiagent systems engineering (MASE) méthodology", Master of science thesis, AFIT/GCS/ENG/01M-12, Department of the air force, air university, air force institute of technology, Wright-Patterson Air Force Base, Ohio, March 2001.

[Sutton 91] Sutton, S. M. Jr., D. Heimbigner, L. J. Osterweil, "Managing change in software development through process programming", Research report, University of Colorado at Boulder, Department of computer science, June 1991.

[Teichroew 77] Teichroew , D. and Hershey , E. A. "PSL/ PSA: A computer aided technique for structured documentation and analysis of information processing systems", IEEE Transaction on Software Engineering 3 (1), pages 48-58, 1977

[Thomas 89] Thomas I., "PCTE Interfaces: Supporting Tools in software engineering environments", IEEE Software, November 1989, pages: 15-23.

[Wang 01] Wang, Alf Inge, "Using a aobile, agent-based environment to support cooperative software processes", Thesis, Presented in Partial Fulfillment of the Requirements for the Degree of Dr.ing. at the Dept. of Computer and Information Science, Norwegian University for Science and Technology, February 5, 2001

[Wood 00] Wood, Mark F., Scott A. DeLoach, "An overview of the multiagent systems engineering methodology", In P. Ciancarini and M. Wooldridge, editors, Agent-Oriented Software Engineering, First International Workshop (AOSE'00), Limerick, Ireland, June 10, 2000, Lecture Notes in Computer Science. Vol. 1957, Springer-Verlag, Berlin, 2000.

[Wooldridge 97] Wooldridge, M., "Agent-based software engineering", In IEE Proceedings of Software Engineering, 1997, pages 26–37

[Wooldridge 00] Wooldridge, M., N. R. Jennings, D. Kinny, "The Gaia methodology for agent-oriented analysis and design", Autonomous Agents and Multi-Agent Systems, Kluwer Academic Press, 3 (3), p. 1-27, 2000.

[Wooldridge 01] Wooldridge, M., Ciancarini P., "Agent-oriented software engineering: The state of the art", In Agent-Oriented Software Engineering. Ciancarini, P. and Wooldridge, M. (eds), Springer-Verlag Lecture Notes in AI Volume 1957, 2001.

[Xu 01] Xu, H. and S. M. Shatz , "A framework for modeling agent-oriented software", Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS 2001), April 16-19, 2001, Phoenix, Arizona, USA. IEEE Computer Society, pages 57-64, 2001

[Xu 03] Xu, H., "A model-based approach of development of multi-agent software systems", PhD Thesis in Computer Science, Graduate College of the university of Illinois at Chicago, 2003.

[Zambonelli 00b] Zambonelli, Franco, Nicholas R. Jennings, Michael Wooldridge, "Organisational abstractions for the analysis and design of multi-agent systems", In P. Ciancarini and M. Wooldridge, editors, Agent-Oriented Software Engineering, First International Workshop (AOSE'00), Limerick, Ireland, June 10, 2000, Lecture Notes in Computer Science. Vol. 1957, Springer-Verlag, Berlin, 2000.

[Zambonelli 03] F. Zambonelli, N. R. Jennings, M., Wooldridge, "Developing multi-agent systems", ACM Transactions on software engineering and methodology, 12(3), October 3, 2003.