Modèle dialectique pour la synthèse de plans

Damien Pellier

Université Joseph Fourier – Grenoble 1

Thèse

pour obtenir le grade de

Docteur de l'ujf

Spécialité : « Informatique »

préparée au laboratoire Imag – Leibniz

dans le cadre de l'école doctorale
« Mathématiques, Sciences et Technologie de l'Information »

présentée et soutenue publiquement

par

Damien Pellier

le 8 décembre 2005

Modèle dialectique pour la synthèse de plans

Directeur de thèse : M. Humbert Fiorino

Jury

MM.	Rachid Alami	Rapporteur
	Pierre-Yves Cunin	Président
	Yves Demazeau	Examinateur
	Amal El Fallah-Seghrouchni	Rapporteur
	Humbert Fiorino	Directeur
	Ana Garcia-Serrano	Examinateur
	Philippe Mathieu	Examinateur

Ce travail de thèse a été accompli sous la direction de Monsieur Humbert Fiorino, Maître de Conférences dans l'équipe Magma au sein du Laboratoire Leibniz de Grenoble.

Je tiens à le remercier pour m'avoir laissé la plus grande liberté dans la conduite de mes travaux et encouragé dans cette tentative toujours difficile qui consiste à explorer des approches nouvelles. Je remercie également Yves Demazeau, Chargé de Recherche CNRS responsable de l'équipe Magma, de m'avoir accueilli et donné les moyens matériels de mener à bien ces travaux.

Je remercie mes rapporteurs pour le travail que leur a demandé la relecture de ma thèse : Monsieur Rachid Alami, Directeur de Recherche au Laboratoire d'Analyse et d'Architecture des Systèmes de Toulouse ainsi que Madame Amal El Fallah Seghrouchni, Professeur à l'Université de Nanterre pour leurs critiques constructives et les approfondissements qu'ils m'ont proposés comme autant de perspectives pour des développements futurs.

Je remercie particulièrement les membres du jury : Monsieur Pierre Yves Cunin, Professeur de l'Université Joseph Fourier, qui a accepté d'en être le président, Madame Ana Garcia Serrano, Professeur à l'Université Polytechnique de Madrid, et Monsieur Philippe Mathieu, Professeur à l'Université des Sciences et Technologies de Lille pour l'intérêt qu'ils ont porté à mon travail.

Finalement, je tiens à remercier spécialement toutes les personnes de l'équipe Magma qui m'ont aidé et supporté tout au long de ses trois années passées à leurs côtés ainsi que ma famille qui a toujours su être disponible, jamais oppressante, en un mot parfaite.

Table des matières

Introduction générale **1**

1 Mécanismes de la coordination **5**

Introduction . . . 5

1.1 Qu'est-ce que coordonner ? . . . 5

1.1.1 La problématique du contrôle distribué . . . 6

1.1.2 L'allocation de tâches . . . 9

1.2 Les structures organisationnelles . . . 12

1.2.1 Les organisations statiques . . . 12

1.2.2 Les organisations dynamiques . . . 13

1.3 La prise de décisions distribuée . . . 13

1.3.1 Le vote . . . 14

1.3.2 La négociation . . . 14

1.3.3 L'argumentation . . . 16

1.4 La coordination de plans . . . 16

1.4.1 Les interactions entre plans . . . 17

1.4.2 Les mécanismes centralisés de coordination . . . 20

1.4.3 La planification partiellement globale . . . 23

1.4.4 La synchronisation distribuée de plans . . . 26

1.4.5 Le paradigme de fusion incrémentale de plans . . . 27

Conclusion . . . 32

2 Modèles de coopération par états mentaux **35**

Introduction . . . 35

2.1 De l'intention à la coopération . . . 35

2.1.1 Engagement et intention individuels . . . 35

2.1.2 Engagement et intention conjoints . . . 37

2.1.3 Le protocole d'engagement . . . 39

2.2 Les état mentaux et la résolution de problèmes 39
2.2.1 L'identification d'actions coopératives 40
2.2.2 La formation d'un groupe 41
2.2.3 La formation de plans 41
2.2.4 La phase d'exécution 42
2.3 La théorie des plans partagés 42
2.3.1 Le modèle intentionnel de la collaboration 42
2.3.2 La définition de la notion de plan 43
2.3.3 Le processus d'élaboration de plans 45
2.4 Les implémentations de la coopération par états mentaux 45
2.4.1 Archon 45
2.4.2 Teamwork 48
2.4.3 La théorie des plans partagés et ses applications 50
Conclusion 50

3 Argumentation et dialogue **53**
Introduction 53
3.1 Le cadre théorique de l'argumentation 53
3.1.1 L'approche de Toulmin 54
3.1.2 L'approche de Dung 54
3.2 Les concepts des dialogues argumentatifs 56
3.2.1 La théorie des actes de langage 56
3.2.2 Les conventions et les normes dans le dialogue 57
3.2.3 Le tableau de conversation et les engagements 58
3.3 Les modèles dialectiques 59
3.3.1 La logique dialogique 59
3.3.2 Les dialogues critiques de Barth et Krabbe 60
3.3.3 Le modèle mathématique du dialogue de Hamblin 61
3.3.4 L'approche dialectique de Prakken 64
3.3.5 Un modèle de dialogue pour l'argumentation 68
3.4 L'argumentation et ses applications 71
3.4.1 Pleadings game 71
3.4.2 Persuader 73
3.4.3 Système Trains 75
Conclusion 78

4 Concepts de la synthèse dialectique de plans 79
Introduction . . . 79
4.1 Le fondement de l'approche . . . 79
4.1.1 Un aperçu du modèle . . . 81
4.1.2 Un exemple introductif . . . 83
4.2 Les notions préliminaires . . . 84
4.2.1 Les états de croyance . . . 84
4.2.2 Les opérateurs et les actions . . . 85
4.2.3 Les notions d'agent, de domaine et de problème . . . 89
4.2.4 Les conjectures et les plans . . . 93
4.2.5 Les plans solutions et les réfutations . . . 98
Conclusion . . . 101

5 Les mécanismes dialectiques 105
Introduction . . . 105
5.1 Raisonner par conjecture - réfutation . . . 105
5.1.1 Le principe . . . 106
5.1.2 Les règles de conjecture - réfutation . . . 108
5.1.3 Les règles de contextualisation . . . 108
5.2 Les cycles de conjecture - réfutation . . . 111
5.2.1 La boucle de raisonnement CoRe . . . 112
5.2.2 Les heuristiques pour la synthèse de plans . . . 115
5.3 Les opérateurs dialectiques . . . 119
5.3.1 Les mécanismes de raffinement . . . 119
5.3.2 Les mécanismes de réfutation . . . 122
5.3.3 Les mécanismes de réparation . . . 123
Conclusion . . . 127

6 Planifier sous hypothèses 129
Introduction . . . 129
6.1 Le modèle de la planification sous hypothèses . . . 129
6.1.1 La problématique . . . 129
6.1.2 Les opérateurs et les méthodes . . . 131
6.1.3 La génération des hypothèses . . . 133
6.2 La production des raffinements . . . 136
6.2.1 Le principe . . . 137

6.2.2 L'algorithme . . . 138
6.2.3 Les heuristiques et le contrôle . . . 142
Conclusion . . . 143

7 Étude de cas **145**
Introduction . . . 145
7.1 Une architecture pour des agents planifiants . . . 145
7.2 Cas 1 : « La production d'outils » . . . 146
7.2.1 Présentation . . . 146
7.2.2 Déroulement du dialogue . . . 148
7.2.3 Discussion . . . 153
7.3 Cas 2 : « Le transport de marchandises » . . . 155
7.3.1 Présentation . . . 155
7.3.2 Déroulement du dialogue . . . 157
7.3.3 Discussion . . . 161
Conclusion . . . 162

Conclusion générale **163**

A Core Assumption-Based Planner **167**
Introduction . . . 167
A.1 Présentation générale . . . 167
A.1.1 Principe . . . 168
A.1.2 Utilisation . . . 168
A.2 Exemples . . . 169
A.2.1 Exemple 1 : « le taquin » . . . 169
A.2.2 Exemple 2 : « The Settlers » . . . 175
A.3 Spécification de la syntaxe (BNF) . . . 179

B CoRe Distributed Message Service **181**
Introduction . . . 181
B.1 Installation de CoRe-DMS . . . 181
B.2 Architecture . . . 182
B.3 Un agent minimal . . . 184
B.4 Exemple : ChatAgent . . . 185
B.5 Administration des *AgentServers* . . . 187
B.5.1 Connexion . . . 187

B.5.2 Déploiement 188
B.5.3 Création 188
B.5.4 Exécution 189

Bibliographie **191**

Liste des tableaux

1.1 Problématique de l'auto-organisation 7

2.1 Règles de réévaluation d'engagement dans ARCHON 47
2.2 Règles de sélection d'actions en cas d'échec dans ARCHON 47

3.1 Sémantique des connecteurs logiques de Lorenzen 60
3.2 Coups du système proposé MacKenzie. 62
3.3 Règles de mise à jour des listes d'engagements 62
3.4 Règles du dialogue proposées par MacKenzie. 63
3.5 Exemple d'évolution des listes d'engagements 64
3.6 Tableau des actes de langage et de leurs relations 70

5.1 Tableau des actes de dialogue classés par niveau. 106
5.2 Règles du dialogue par conjecture - réfutation. 109
5.3 Taxonomie des réparations par ajout d'une contrainte d'ordre. 124

Table des figures

1.1 Problème de décomposition de tâche 7
1.2 Problème d'allocation de tâches 8
1.3 Réalisation des sous-tâches 8
1.4 Les mécanismes d'allocation de tâches 9
1.5 Les quatre phases du *contract-net* 11
1.6 Taxonomie des relations multi-agents 19
1.7 Architecture du système DSIPE 22
1.8 Réseau de capteurs de DVMT 23
1.9 Synchronisation de plans 27
1.10 La boucle de la fusion incrémentale de plans 29
1.11 Le protocole de délibération de PMP 31
1.12 Stratégie de résolution d'interblocages 32

2.1 Architecture d'un agent ARCHON 46
2.2 Les opérateurs dans STEAM 49
2.3 L'architecture de WEBTRADER 50

3.1 La cellule argumentative de Toulmin 55
3.2 Relations d'attaque dans un système argumentatif 56
3.3 Protocole de communication de Winograd et Flores 58
3.4 Structure des arguments dans PERSUADER 74

4.1 Schématisation du modèle de construction dialectique de plans. 82
4.2 Exemple du transporteur. 86
4.3 Exemple de violation des contraintes d'ordre 97
4.4 Exemple de conjecture 99
4.5 Exemple de réfutation 100
4.6 Exemple de plan-solution. 103

5.1 Exemple de conjecture initiale. 107
5.2 Exemple de raffinement. 107
5.3 Automate de contextualisation du dialogue 111
5.4 Représentation ET/OU du tableau de démonstration 118
5.5 Raffinement par ajout d'un lien causal 120
5.6 Raffinement par ajout de contraintes d'instanciation 121
5.7 Raffinement par ajout d'une sous-conjecture 122
5.8 Exemple de réfutation 123
5.9 Réparation par ajout de contraintes d'ordre 124
5.10 Réparation et suppression d'actions 127
5.11 Réparation par ajout d'un conjecture 128

6.1 Représentation d'un raffinement sous forme d'arbre. 138

7.1 Schéma de l'architecture interne des agents 147
7.2 Échec du processus de synthèse de plan. 154
7.3 Exemple de boucle de raffinements 155
7.4 État de croyance de l'agent conveyor1 156
7.5 État de croyance de l'agent conveyor2 158
7.6 Réfutation énoncée par conveyor1_2. 160
7.7 Réparation énoncée par conveyor2_4. 160

A.1 Le jeu du taquin 169
A.2 *Applet* du taquin 174
A.3 Graphe de dépendances entre les agents. 176

B.1 Exemple d'une topologie d'*AgentServers*. 182
B.2 Architecture de CoRe-DMS. 183
B.3 Console d'administration de CoRe-DMS. 187

Introduction générale

La planification est une problématique centrale de l'Intelligence Artificielle. Son objectif consiste à générer un plan d'actions à un niveau symbolique à partir d'un état initial pour atteindre un but défini *a priori*. En effet, si l'un des objectifs de l'IA est de rendre calculables certains aspects de l'intelligence, alors la planification est certainement, en tant que raisonnement sur l'action, l'une des composantes majeures permettant d'atteindre cet objectif. De notre point de vue, raisonner sur l'action est une condition *sine qua non* à l'autonomie décisionnelle et à la rationalité d'un agent artificiel.

L'évolution et le déploiement des systèmes intelligents, tels que les systèmes multi-agents, posent dorénavant ces questions à un niveau supérieur, celui d'un groupe d'agents. Par conséquent, l'étude de la synthèse de plans, dans un contexte multi-agent, définit un cadre théorique privilégié. Elle permet l'analyse des mécanismes de raisonnement nécessaires au développement d'agents capables de produire un comportement collectif adéquat.

D'un point de vue pratique, la distribution grandissante des applications dans lesquelles il n'est pas possible d'envisager un contrôle centralisé, pour des raisons organisationnelles ou techniques, motive la conception de systèmes multi-agents capables de collaborer pour réaliser un objectif commun. La mise en œuvre de tels systèmes passe par l'étude des mécanismes algorithmiques permettant aux agents de raisonner *a priori* sur l'action collective à mener. Considérons un groupe de robots mobiles dont le but est d'effectuer des prélèvements de roches sur Mars. L'ensemble des compétences nécessaires à l'accomplissement de la mission est réparti sur plusieurs robots : un robot sonde en charge de rapporter les échantillons sur Terre, un robot adapté au transport des échantillons au sol et finalement un robot en charge du forage. Pour qu'une telle mission réussisse, en l'absence d'opérateur humain, les robots sont dans l'obligation de se concerter pour décomposer la tâche collective en un ensemble de sous-tâches, de raisonner sur les capacités des autres agents ou encore d'élaborer une vue commune de l'état de l'environnement. L'enjeu ici n'est pas une « simple » coordination des activités des agents partageant un même environnement, mais la construction d'un plan pour atteindre un objectif commun.

En effet, dans le cadre de la problématique du contrôle et de la coordination des activités au sein d'un environnement partagé l'objectif n'est pas d'élaborer un plan collectif, mais d'assurer que chaque agent pourra localement atteindre ses buts dans un contexte global en maintenant l'intégrité fonctionnelle du système. Par consé-

quent, l'accent est mis sur les mécanismes de résolution de conflits entre les activités des agents. Ils peuvent être vus notamment comme un processus de négociation au cours duquel les agents doivent parvenir à un consensus pour l'accès à une ressource, un processus de synchronisation ou encore un processus de fusion de leurs différentes activités.

En revanche, dans le cadre de la planification collective qui nous intéresse plus particulièrement dans ce manuscrit, c'est la construction coopérative d'un plan par un groupe d'agents qui est le noyau fonctionnel du système. Les agents échangent des informations sur leurs plans respectifs qu'ils modifient itérativement jusqu'à aboutir à un plan global exempt de conflit. L'élaboration d'un plan global pose alors les questions suivantes : quelle représentation choisir pour modéliser les plans partiels des agents ? Par quels mécanismes les plans sont-ils produits ? Comment les tâches sont-elles décomposées en sous-tâches et allouées aux agents ? Comment et quelles informations les agents doivent-ils communiquer au cours de la synthèse du plan ? Quels sont les mécanismes de coordination à mettre en œuvre pour garantir l'exécution cohérente et efficace du plan ?

Dans la continuité de cette approche, nous proposons d'aborder la problématique de la synthèse d'un plan comme un raisonnement collectif et révisable à la croisée de deux domaines : d'une part, le domaine de la planification qui spécifie les mécanismes nécessaires à la production de plans et, d'autre part, le domaine de l'argumentation qui s'intéresse plus particulièrement à la formalisation du raisonnement produit par des agents au travers de leurs interactions. En effet, si la planification se définit comme un raisonnement sur l'action, l'extension de ses mécanismes dans un contexte multi-agent nécessite la prise en compte des aspects liés à la communication. Par conséquent, l'argumentation, qui fonde justement ses mécanismes sur les interactions et les oppositions entre agents, apparaît comme une voie de recherche prometteuse, notamment pour la conception de systèmes capables de délibérer conjointement à la réalisation d'un but commun.

Nous soutenons l'idée qu'au même titre que l'argumentation cherche à produire une preuve de la validité d'une proposition, la planification cherche à construire une preuve particulière permettant de démontrer qu'un but peut être atteint. Ce parallèle nous amène à défendre la thèse selon laquelle *la synthèse collaborative d'un plan* par un groupe d'agents peut être vue comme *un processus dialectique de conjecture - réfutation*. De manière similaire à un dialogue argumentatif, les agents échangent des propositions et des contre-propositions afin d'élaborer *une solution globale et collective*. Nous considérons, dans ce manuscrit, *la distribution des croyances et des compétences des agents*, ainsi que *l'impossibilité pour un seul agent de réaliser l'objectif* du système comme des propriétés intrinsèques du problème.

Notre approche s'articule autour de deux idées directrices :

Tout d'abord, outre la capacité des agents à débattre et à interagir, la synthèse de plans repose sur leur faculté à raisonner. Ils s'appuient sur des *hypothèses*, *i.e.,* des faits probables, par opposition au raisonnement se fondant sur des propriétés certaines du monde. C'est de cette caractéristique que naît la coopération entre agents.

En effet, les hypothèses sont des propositions avancées provisoirement par un agent pour soutenir qu'une action est réalisable. Ces hypothèses doivent alors être contrôlées et vérifiées ultérieurement par les autres agents. Reprenons l'exemple des robots mobiles dont le but est le prélèvement d'échantillons de roches sur Mars. Aucun des agents ne peut résoudre la mission seul. Le robot en charge du forage peut proposer ses services pour récolter les fragments de roches en supposant qu'ils seront transportables de sa position d'asservissement à la station de base. Le robot transporteur peut alors offrir son aide pour déplacer les échantillons en supposant, cette fois, qu'ils pourront être rapportés sur Terre. Il démontre ainsi que l'hypothèse formulée par le robot foreur est réalisable. Finalement, le robot sonde peut apporter sa contribution à l'élaboration du plan en ramenant les prélèvements. Il prouve alors que l'hypothèse du robot transporteur, mais également le but global confié au système, peut être atteint. Cet exemple illustre l'intérêt pour des agents de formuler des hypothèses qui apparaissent comme des sous-buts pour les autres agents. Ils peuvent ainsi proposer leurs compétences et leurs croyances pour en démontrer la validité et participer à la construction collaborative d'un plan. La mise en œuvre de tels mécanismes soulève plusieurs questions : quels sont les mécanismes qui permettent de produire des plans sous hypothèses, *i.e.,* des plans qui peuvent être réalisés si certaines conditions sont vérifiées ? Quelles propriétés du monde peuvent être « raisonnablement » supposées par les agents ?

La seconde idée directrice repose sur la métaphore de la logique de la découverte mathématique[1]. En effet, si les hypothèses donnent un cadre à la coopération, la notion de réfutation ou de conflit joue un rôle central dans le raisonnement produit à l'élaboration d'un plan-solution. Contrairement aux approches qui cherchent à éviter absolument les conflits, nous proposons de considérer les conflits autrement que comme contingents. Nous nous appuyons sur ceux-ci pour faire progresser la synthèse de plans. Supposons qu'un agent détecte qu'un plan n'est pas correct, il doit alors le réfuter. De cette réfutation naît alors l'obligation chez les autres agents, soit d'abandonner le plan en cours de construction, soit de le réparer. Lorsque la réparation du plan précédemment réfuté réussit, le plan devient plus robuste puisqu'il résiste dorénavant à cette réfutation mais peut être encore réfuté. Ce mécanisme de réfutation – réparation permet une synthèse de plans plus robustes et participe également aux partages des compétences et des croyances entre agents.

La première partie de ce manuscrit dresse un état de l'art de la planification multi-agent et des principaux modèles de dialogues argumentatifs (chapitres 1, 2 et 3). La seconde présente le modèle de synthèse de plans par conjecture – réfutation (chapitre 4, 5, 6 et 7).

État de l'art. Dans le premier chapitre, nous présentons les mécanismes de la coordination multi-agent. Nous abordons, en particulier, la problématique du contrôle distribué. Elle peut se décomposer en trois phases : la décomposition de tâches en sous-tâches, l'allocation et enfin l'exécution et la coordination des sous-tâches. Dans le deuxième chapitre, nous étudions les modèles de coopération par états mentaux.

[1] Popper K. (1973) : *La logique de la découverte scientifique.* Paris, Payot.

Ces travaux se fondent sur la nécessité, pour un groupe d'agents, de posséder une intention conjointe de réaliser un but. Dans le troisième chapitre, nous abordons plus spécifiquement l'argumentation ainsi que les modèles dialectiques. Nous essayons au travers de ce chapitre, de présenter une vue générale des concepts manipulés en argumentation et d'étudier dans quelle mesure ils peuvent être transposés dans le cadre de notre approche.

Modèle dialectique de synthèse de plans. Dans le quatrième chapitre, nous introduisons formellement les notions fondamentales du modèle de synthèse dialectique de plans. En particulier, nous formalisons les concepts d'hypothèses, de plans et de réfutations. Dans le cinquième chapitre, nous donnons les différents mécanismes nécessaires à la mise en œuvre de notre approche. Ce chapitre constitue le cœur du modèle. Dans le sixième chapitre, nous nous intéressons plus précisément aux mécanismes de planification utilisés par les agents pour démontrer la validité des hypothèses. Finalement, dans le septième chapitre, nous proposons une étude de cas mettant en exergue les points importants de l'approche.

Chapitre 1

Mécanismes de la coordination

Introduction

La coordination est une problématique centrale des systèmes multi-agents dans la mesure où les agents doivent partager un même environnement et sont dans l'obligation de mettre en commun leurs compétences pour réaliser une tâche complexe. Bien qu'il existe des techniques de coordination réactive (Ferber, 1995), nous ne nous intéressons dans ce chapitre qu'aux approches fondées sur des représentations. L'objectif est de dresser une vue générale de leurs différents mécanismes.

Nous présentons, dans un premier temps (§ 1.1), la problématique de la coordination multi-agent. Puis, nous introduisons (§ 1.2) les mécanismes de coopération fondés sur les structures organisationnelles et abordons les techniques de prise de décisions distribuées (§ 1.3). Finalement, nous consacrons la dernière partie de ce chapitre (§ 1.4) aux problèmes liés à la coordination de plans.

1.1 Qu'est-ce que coordonner ?

Commençons par définir de manière informelle les notions de coordination et de coopération au sein d'un système multi-agent. Ces notions sont souvent employées de manière analogue, toutefois elles présentent quelques différences.

La définition usuelle de la coordination peut s'exprimer comme la « mise en ordre, l'agencement calculé des parties d'un tout selon un plan logique en vue d'une fin déterminée ». Cette définition se traduit dans un contexte multi-agent par l'identification des relations entre les activités des agents dans le but d'éviter les conflits inhérents au nécessaire partage de l'environnement. La notion de coopération se définit comme « l'action de participer (avec une ou plusieurs personnes) à une œuvre ou à une action commune ». Par conséquent, dans un système multi-agent, la coopération implique l'existence d'un but commun à l'ensemble des agents.

Coordonner et coopérer sont donc des activités quelque peu différentes. Le lien toutefois essentiel entre ces deux notions réside dans le fait que la coopération im-

plique une certaine forme de coordination. En effet, pour que l'on puisse parler de coopération, les activités des agents doivent au moins être coordonnées afin de ne pas induire d'*interactions négatives* rendant le but commun inatteignable. Ceci nous amène à définir la coopération comme un processus de coordination dans lequel la société d'agents est conjointement engagée à réaliser un but. On parle alors d'agents *« coopératifs »* en opposition aux agents *« compétitifs »*.

1.1.1 La problématique du contrôle distribué

Les travaux précurseurs (Lesser and Corkill, 1981; Smith and Davis, 1981) dans le domaine de la résolution distribuée de problèmes, formulent l'hypothèse que les agents partagent implicitement un but commun et qu'il n'existe pas de conflit entre leurs activités. Ces hypothèses sont acceptables dans la mesure où les agents appartiennent à une même organisation et qu'ils s'aident mutuellement à résoudre le problème qui leur a été confié. Toutefois, ces hypothèses apparaissent trop restrictives. En effet, les agents ne partagent pas obligatoirement un but commun et peuvent avoir des intérêts contradictoires, même s'ils sont parfois obligés de coopérer pour atteindre leurs buts. Par conséquent, il est nécessaire de mettre en place des mécanismes de raisonnement et de délibération permettant aux agents de décider conjointement des actions à mener.

Ces mécanismes posent alors la problématique du contrôle distribué. En effet, chaque agent représente une entité autonome possédant ses propres mécanismes de décision ainsi que ses propores croyances sur le monde. La question qui se pose est la suivante : comment organiser le fonctionnement d'un groupe d'agents pour lui permettre de coopérer ? Cette problématique (Fiorino, 1999), reprise par le tableau 1.1, implique que les agents soient capables de décomposer la tâche qui leur a été affectée en sous-tâches, de se partager la charge de la réalisation, et finalement de coordonner l'exécution des sous-tâches en fonction des aléas. Détaillons maintenant ces trois niveaux d'analyse.

Décomposition de tâches en sous-tâches. Les agents décident de la meilleure façon de décomposer la tâche qui leur a été assignée en un ensemble de sous-tâches pour en réduire la complexité et permettre une résolution distribuée (cf. figure 1.1). La décomposition d'une tâche collective en un ensemble de sous-tâches conduit donc à définir une ébauche d'organisation. Par exemple, la tâche « transporter une table dans une salle » peut devenir « trois agents portent la table et un quatrième assume la navigation jusqu'à la salle ».

Allocation des sous-tâches aux agents. Les agents cherchent la meilleure répartition possible en fonction des ressources communes et des ressources propres (cf. figure 1.2). Ce problème a fait l'objet de nombreux travaux issus des idées de (Smith and Davis, 1981) et de leur protocole « contract-net ». Au cours de la phase d'allocation, les sous-tâches sont affectées aux agents suivant un critère de coût (*e.g.,* le temps nécessaire à la réalisation d'une tâche).

Sachant que...	**Comment obtenir...**
▷ la connaissance et le contrôle sont distribués ▷ les agents ont des connaissances partielles sur leurs activités respectives et sur l'environnement ▷ certaines tâches nécessitent plusieurs agents pour être réalisées ▷ certaines tâches sont réalisées de façon concurrente ▷ les tâches changent de façon imprévisible, en fonction des aléas d'exécution ou des requêtes d'un opérateur	▷ la coordination des tâches et l'évitement des conflits ▷ la parallélisation des tâches ▷ la non redondance des tâches ▷ le partage des ressources ▷ la robustesse du groupe face aux pannes

TAB. 1.1: Problématique du contôle distribué.

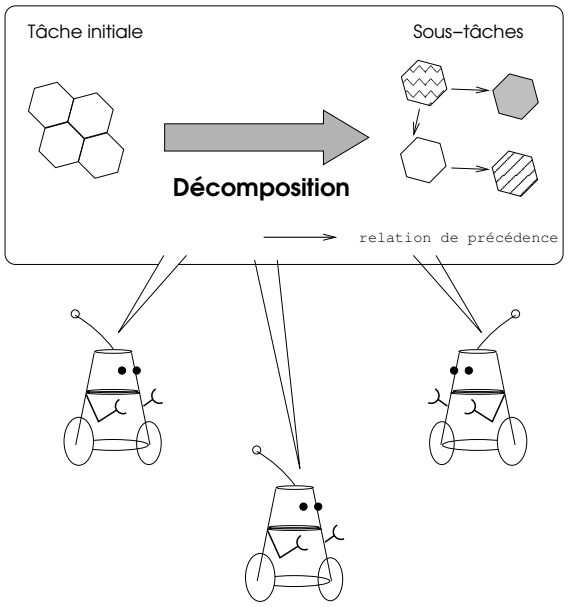


FIG. 1.1: Problème de décomposition de tâches.

Réalisation des sous-tâches. Les agents échangent des informations sur leurs activités pour détecter des relations de dépendance (cf. figure 1.3) :

1. Les agents peuvent être en conflit pour l'accès à une ressource, ou ne pas partager le même point de vue sur la situation globale, ou encore produire des actions contradictoires. Il existe une littérature abondante qui traite ce sujet (Adler et al., 1989). Le cadre formel le plus souvent cité est celui des protocoles de négociation où les agents échangent interactivement des propositions et des

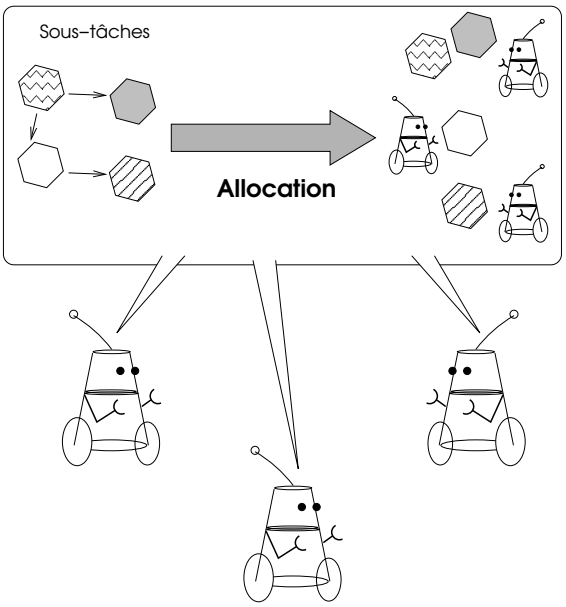


FIG. 1.2: Problème d'allocation de tâches.

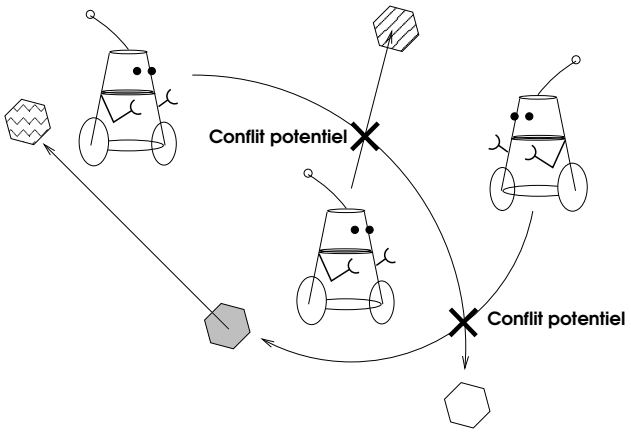


FIG. 1.3: Réalisation des sous-tâches.

contre-propositions jusqu'à trouver un accord « équitable » pour l'ensemble des parties (Ephrati and Rosenschein, 1991).

2. Les agents peuvent agir de façon redondante (Martial, 1992). Le problème ici consiste pour le concepteur à trouver le meilleur compromis entre la robustesse et les performances de son système. Cet équilibre peut être atteint dynamiquement par les agents qui s'échangent des informations sur leurs activités courantes. Si l'on dispose de stratégies qui permettent de déterminer quelles sont les informations pertinentes à transmettre, on peut contraindre l'espace de recherche d'un agent et lui éviter de refaire le travail d'un autre agent. Dans le cas contraire, on risque de « distraire » l'agent par le traitement d'informations non pertinentes ou même d'induire des réactions erronées de sa part.

3. Généralement, les tâches locales doivent être partiellement ordonnées dans le temps. Cela revient donc à élaborer un graphe de dépendances qui tient compte à la fois des contraintes locales (*e.g.,* estimation du temps nécessaire à la réalisation d'une tâche locale, gestion des ressources) et de la décomposition initiale de la tâche collective (Marc et al., 2003).

1.1.2 L'allocation de tâches

Dans ce paragraphe, nous proposons de nous intéresser plus particulièrement aux mécanismes d'allocation de tâches. Ces mécanismes peuvent être réalisés soit de manière *centralisée* soit de manière *distribuée* sur l'ensemble des agents du système (cf. figure 1.4).

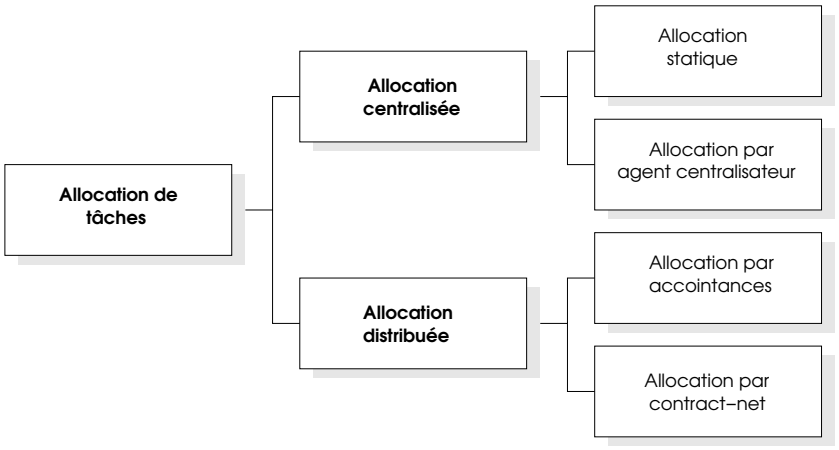


FIG. 1.4: Les mécanismes d'allocation de tâches.

Allocation centralisée. Le modèle le plus simple d'allocation peut être réalisé avec un seul agent centralisateur dans le système (Cammarata et al., 1983; Georgeff, 1983). Cet agent gère une table stockant les compétences respectives de chaque agent. Cette table peut être automatiquement mise à jour lorsqu'un agent entre dans le système. Lorsqu'un agent a besoin de faire réaliser une tâche par un autre agent, il interroge l'agent centralisateur pour trouver un agent capable de l'exécuter. L'agent centralisateur interroge à son tour tous les agents possédant les compétences pour réaliser la tâche. Si l'un d'entre eux accepte, il acquitte alors la demande de l'agent requérant. Dans le cas contraire, il l'informe de l'impossibilité de trouver un agent capable de réaliser la tâche souhaitée. Dans le cas où plusieurs agents répondraient de manière positive à la proposition de l'agent centralisateur, plusieurs métriques (*e.g.,* le temps d'exécution, les ressources consommées, *etc.*) peuvent être utilisées afin de choisir la meilleure proposition.

L'approche centralisée du processus d'allocation est sensible aux pannes et apparaît très peu efficace lorsque le nombre d'agents du système devient important. Pour répondre à ces critiques, il est possible de multiplier le nombre d'agents centra-

lisateurs dans le système afin de le rendre plus robuste. Toutefois, la multiplication des agents centralisateurs oblige à mettre en place des mécanismes pour maintenir la cohérence des différentes tables de compétences.

Allocation distribuée. Nous présentons, dans ce paragraphe, les deux principales techniques d'allocation distribuée de tâches : l'allocation par *réseau d'accointances* implémentée par exemple dans (Wooldridge et al., 1991; Gasser et al., 1988; Gasser et al., 1987) et l'allocation par *contract-net* (Smith and Davis, 1981).

Allocation par réseaux d'accointances. Les systèmes basés sur les réseaux d'accointances formulent l'hypothèse que chaque agent possède une table de compétences qui associe pour chaque agent connu la liste de ses compétences :

$$C_1 : [A_{11}, \ldots, A_{1k}]; \ldots; C_n : [A_{n1}, \ldots, A_{nm}]$$

ou C_i sont les compétences requises pour l'exécution d'une tâche T_i et A_{ij} les agents capables de réaliser ce type de tâche. Le plus souvent, cette table est représentée par une matrice dans laquelle les lignes expriment les compétences et les colonnes les accointances.

Ce type de technique permet de prendre en compte le caractère distribué et partiel des connaissances des agents au sein du système. Deux types d'allocation (Ferber, 1995) peuvent être distingués selon que les agents sont ou non autorisés à déléguer leurs demandes de compétences aux autres agents :

L'allocation directe. Dans ce type d'allocation l'agent qui doit déléguer une tâche ne peut le faire qu'avec les agents qu'il connaît directement (*i.e.,* ceux qui se trouvent dans sa table de compétences). Il demande alors à chaque agent possédant les compétences requises pour la tâche, s'il est prêt à la réaliser pour lui. Si aucun des agents connus n'accepte, le système échoue.

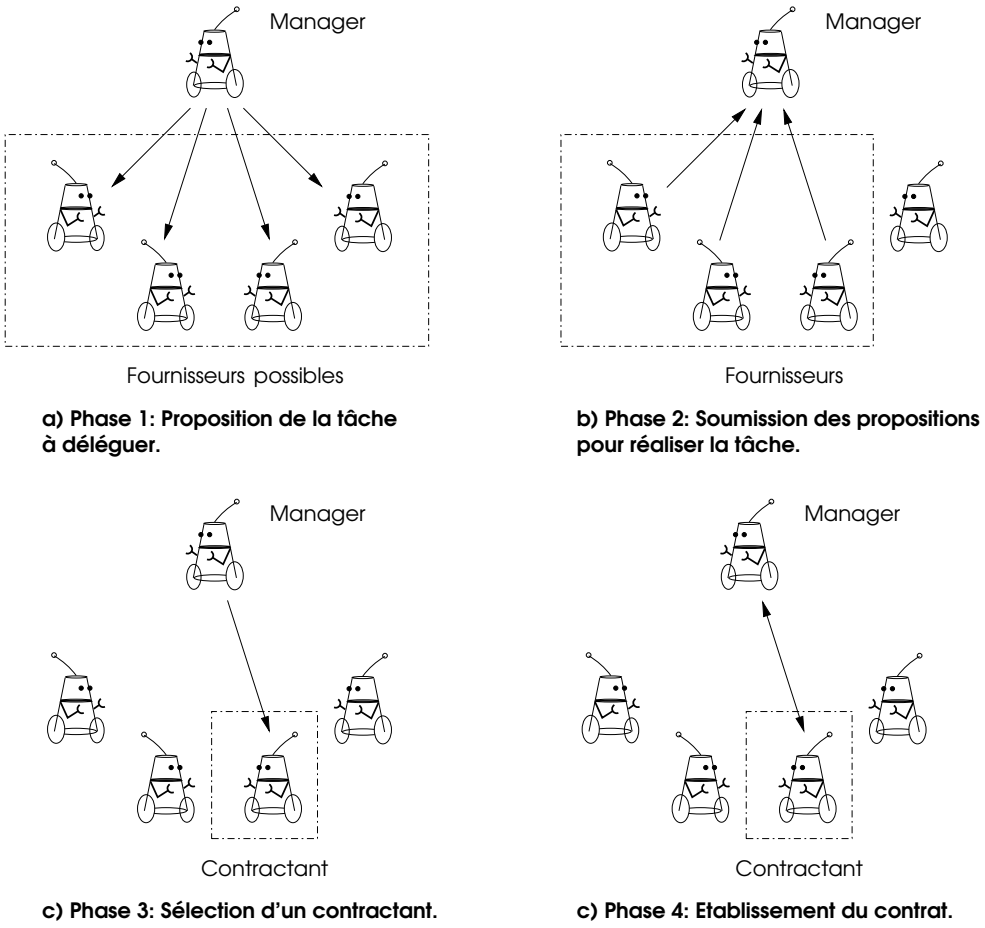
L'allocation par délégation. Le mécanisme d'allocation par délégation permet à des agents ne se connaissant pas directement de s'allouer des tâches. En effet, ce type d'allocation autorise un agent à questionner les autres agents sur leurs compétences et sur leurs connaissances des autres agents du système. L'agent demandeur diffuse sa demande à tous les agents connus qui eux-mêmes la propagent aux agents qu'ils connaissent jusqu'à ce qu'au moins un agent capable de réaliser la tâche soit trouvé. Cette technique peut être vue comme la construction de tables de routage de compétences au sein du système multi-agent.

Allocation par *contract-net*. Le *contract-net* (Smith and Davis, 1981) est un mécanisme d'allocation de tâches reposant sur un protocole de la délégation entre un manager et un ensemble de fournisseurs. Il se décompose en quatre phases (cf. figure 1.5) :

1. La premier phase est la phase d'appel d'offre. Le manager diffuse, à l'ensemble des fournisseurs qu'il considère capables de réaliser la tâche, la description de la tâche qu'il désire déléguer (cf. figure 1.5 a) ;

2. Lors de la deuxième phase et sur la base de la description fournie par le manager, les fournisseurs envoient une proposition au manager pour évaluation (cf. figure 1.5 b) ;
3. Au cours de la troisième phase, le manager reçoit et évalue les propositions des fournisseurs, puis choisit la meilleure proposition (cf. figure 1.5 c) ;
4. Finalement, au cours de la quatrième et dernière phase du protocole, le fournisseur, choisi par le manager, envoie une confirmation traduisant son engagement à réaliser la tâche déléguée (cf. figure 1.5 d).

Le *contract-net* offre un mécanisme d'allocation souple et efficace tout en permettant de contrôler le processus d'allocation de tâches entre un manager et un ensemble de fournisseurs. Toutefois, le *contract-net* n'est pas un modèle de résolution distribuée de problèmes, mais seulement un protocole permettant l'auto-organisation et la distribution dynamique des tâches au sein d'un système multi-agent. Comme tous les protocoles, il ne formule aucune hypothèse quant au contenu des messages échangés et par conséquent sous-tend l'existence d'un langage commun à la société d'agents.



a) Phase 1: Proposition de la tâche à déléguer.

b) Phase 2: Soumission des propositions pour réaliser la tâche.

c) Phase 3: Sélection d'un contractant.

c) Phase 4: Etablissement du contrat.

FIG. 1.5: Les quatre phases du *contract-net*

1.2 Les structures organisationnelles

Les structures organisationnelles peuvent jouer un rôle important pour définir et coordonner les agents d'un système. Par exemple, le code de la route édicte un ensemble de règles organisationnelles permettant de limiter les accidents. Une organisation peut être vue, de façon externe, comme une délimitation des responsabilités et des compétences des agents, dans laquelle chaque agent joue un rôle déterminé par ses capacités et son autorité au sein de la société d'agents. La littérature (Ferber, 1995) distingue classiquement trois dimensions organisationnelles en fonction de son caractère plus ou moins dynamique :

Distribution fonctionnelle. Comme son nom l'indique ce type d'organisation repose sur une distribution de l'organisation en fonction des rôles et des compétences des agents. En d'autres termes, les agents sont regroupés en fonction de leurs caractéristiques fonctionnelles ;

Distribution spatiale. La distribution spatiale est souvent liée au problème à traiter par la communauté d'agent. Ainsi, dans le cas de DVMT (Lesser and Corkill, 1983), l'organisation est fondée sur un découpage de l'environnement, chaque agent étant responsable de la surveillance d'une zone délimitée. Le découpage de l'environnement n'est pas forcément une partition de ce dernier car les recouvrements introduisent, au niveau des transitions de zones, une certaine robustesse du système.

Distribution temporelle. La dernière dimension organisationnelle se réfère au caractère dynamique que peut prendre l'organisation au sein d'un système multi-agent. Les compétences, ainsi que les positions spatiales des agents, peuvent évoluer au cours du temps. Une distribution fonctionnelle ou spatiale réalisée a priori peut par conséquent être invalidée. Les agents doivent alors posséder des mécanismes leur permettant de se réorganiser. Cette dimension s'appuie notamment sur les travaux issus de la formation de coalitions (Sandholm et al., 1999; Vauvert and El-Fallah-Seghrouchni, 2000).

1.2.1 Les organisations statiques

Dans ce type d'organisation, les rôles et les relations entre les agents peuvent être définis a priori. L'adaptation des système multi-agents, reposant sur une telle organisation, reste limitée. Ce type d'approche organisationnelle est souvent retenu dans les cas où le nombre d'agents reste peu important et, en particulier, dans les systèmes coopératifs d'agents spécialistes dans lesquels chaque agent sait où obtenir les informations et retourner les résultats.

Ce type de structure organisationnelle s'appuie sur des règles de comportement que les agents doivent respecter pour éviter les conflits potentiels. Les conflits sont ainsi résolus *a priori*. Cette technique de coordination est appelée dans la littérature *coordination par normes et lois sociales*, *e.g.,* (Lewis, 1969; Jennings, 1993; Sichman et al., 1993; Shoham and Tennenholtz, 1995; Castelfranchi, 1998). Les normes et les lois définissent des patrons de comportements. Leurs intérêts résident notamment

dans la limitation des coûts liés aux communications entre les agents du système (Briggs and Cook, 1995).

1.2.2 Les organisations dynamiques

Les organisations dynamiques revêtent un caractère particulièrement important notamment dans le cadre de la résolution coopérative de problèmes (Wooldridge and Jennings, 1999) ou encore dans les mécanismes d'allocation de tâches (Shehory and Kraus, 1995). Selon (Wooldridge and Jennings, 1999), la résolution coopérative de problèmes distribués peut être découpée en quatre phases dont l'une est le regroupement d'agents ayant identifié une tâche coopérative et sollicitant une assistance pour la réaliser. Si cette phase réussit, alors la coalition d'agents ainsi formée est collectivement engagée à réaliser une action coopérative.

Les travaux sur la formation de coalitions se sont fortement inspirés des modèles économiques, dans lesquels plusieurs compagnies s'allient pour répondre aux demandes du marché. La formation de coalitions (Ketchpel, 1994; Vauvert and El-Fallah-Seghrouchni, 2000) peut se définir comme un type d'organisation à court terme, contrairement aux approches organisationnelles statiques. Elles permettent à un ensemble d'agents de se regrouper pour résoudre une tâche complexe qu'ils ne pourraient exécuter seuls ou qu'ils préfèrent exécuter de manière collective. Toute coalition peut être construite et dissoute à tout moment de manière opportuniste, rendant ainsi le système plus flexible et adaptable à des environnements ouverts et dynamiques.

1.3 La prise de décisions distribuée

Une problématique récurrente (Davis and Smith, 1983), associée à la coordination dans un système multi-agent, réside dans les mécanismes de prise de décisions. La capacité d'atteindre un « consensus » ou un « compromis » par des agents coopératifs cognitifs est une propriété fondamentale des systèmes intelligents et autonomes, sans laquelle la société d'agents toute entière ne peut fonctionner.

Tous les mécanismes d'interactions, permettant d'atteindre un consensus au sein d'une société d'agent, doivent exhiber un certain nombre de propriétés, notamment l'absence d'interblocage (Holzmann, 1991), comme d'une manière générale dans tous les systèmes distribués. Toutefois, au sein d'un système multi-agent, les propriétés sont quelque peu différentes. (Sandholm, 1999) distingue, par exemples les propriétés suivantes :

- *La garantie de succès.* Un protocole de prise de décision doit garantir qu'un consensus sera trouvé ;
- *La maximisation des gains locaux.* Intuitivement, un protocole doit permettre à chaque agent de maximiser ses gains locaux et ainsi maximiser de manière globale l'ensemble des gains du système ;

- *La rationalité individuelle.* Un protocole est qualifié d'individuellement rationnel si les règles du protocole ne vont pas à l'encontre des intérêts des participants ;
- *La stabilité.* Un protocole est qualifié de stable s'il garantit que le comportement de tous les agents est cohérent, par exemple, en atteignant un équilibre de Nash (Nash, 1950). Dans ce cas, aucun des agents ne doit avoir intérêt à changer de stratégie face à la stratégie d'équilibre de ses concurrents. Autrement dit, aucun des agents ne doit pouvoir obtenir plus en choisissant une autre stratégie que sa stratégie d'équilibre, à condition que les autres agents continuent à choisir leurs stratégies d'équilibre ;
- *La simplicité.* Un protocole est « simple » s'il permet à chaque participant de déterminer facilement la stratégie optimale ;
- *La distribution.* Un protocole doit idéalement être distribué afin d'augmenter la robustesse et la tolérance aux fautes.

Dans ce paragraphe nous allons présenter trois types de mécanismes pouvant être mis en œuvre pour atteindre un consensus : le vote, la négociation et l'argumentation.

1.3.1 Le vote

Les mécanismes de vote permettent à une société d'agents de choisir une solution parmi un ensemble d'alternatives possibles. Classiquement, ces mécanismes peuvent se découpler en deux phases : une phase durant laquelle les agents proposent les différentes alternatives et une phase de vote au cours de laquelle les agents expriment leurs préférences, *i.e.,* pour ou contre telle ou telle alternative. Ces deux phases s'articulent généralement autour d'un protocole (Pitt et al., 2005) réglementant le processus de vote.

Lorsque le système n'est pas coopératif, certains agents peuvent mentir (*i.e.,* voter pour une préférence qui n'est pas la leur ou au contraire voter contre l'une de leurs préférences). Pour répondre à cette problématique un certain nombre d'approches ont été proposées. Par exemple, dans le projet CLARKE TAX (Ephrati and Rosenschein, 1991), le mécanisme de vote garantit que la seule stratégie valide est la sincérité, permettant ainsi de maximiser les gains des agents.

1.3.2 La négociation

La négociation est un processus au cours duquel les agents échangent et évaluent des propositions (des contrats) dans le but de maximiser leurs intérêts individuels. Dans le cadre de la coordination multi-agent, la négociation a été utilisée notamment pour prendre en compte l'allocation de ressources (Chevaleyre et al., 2005) et de tâches (Rosenschein and Zlotkin, 1994). La plupart des systèmes basés sur la négociation sont composés de quatre composants :

- *Un ensemble de propositions* qui représente l'espace possible des propositions que peut formuler un agent au cours de la négociation ;

- *Un protocole* qui définit l'enchaînement des propositions que peut énoncer un agent (*i.e.,* les règles de l'interaction). Ces règles sont connues par tous les agents ;
- *Un ensemble de stratégies* qui détermine quelle proposition un agent énonce pour atteindre au mieux le but fixé. La stratégie d'un agent est le plus souvent privée (*i.e.,* non visible par les autres agents) afin de maximiser son succès dans la négociation ;
- *Une règle de terminaison* qui permet aux agents de déterminer si la négociation est bloquée ou si elle a abouti à un accord.

Une négociation se décompose en une succession d'interactions au cours desquelles chaque agent énonce une proposition ou une contre-proposition. Les propositions énoncées sont contenues dans l'ensemble des propositions possibles. Chaque proposition est acceptable si elle respecte les règles du protocole de négociation. L'ordre des échanges est dirigé par la stratégie des agents. Si un accord est atteint (*i.e.,* défini par la règle de terminaison), alors la négociation est conclue et un consensus est établi.

L'une des principales contributions à la négociation, issue des travaux basés sur la théorie des jeux, fut celle de (Rosenschein and Zlotkin, 1994) qui ont introduit la distinction entre la négociation *orientée tâches* qui porte sur la redistribution des tâches et celle *orientée bénéfices* ou valeurs[1] qui porte sur une redistribution des buts. La littérature distingue également la négociation *orientée états* qui porte sur l'ordonnancement des plans des agents. Rosenchein et Zlotkin modélisent les dépendances entre les agents via un modèle quantitatif basé sur les fonctions d'utilité où chaque agent cherche à maximiser ses gains. Ils présupposent l'existence d'un protocole de négociation (sous forme de règles données par le concepteur), laissant à chaque concepteur le choix de la stratégie de prise de décisions.

Appliquée à la coordination multi-agent, la théorie des jeux fournit un cadre formel pour modéliser la négociation et la prise de décisions entre des agents dont les intérêts individuels sont représentés par des fonctions d'utilité. Cette formalisation permet notamment de prouver la convergence du processus de négociation. Toutefois, dans le cadre de la prise en compte d'hypothèses plus complexes (*e.g.,* incertitude sur les informations échangées, agents susceptibles de frauder, etc.) la théorie des jeux ne permet pas de garantir l'existence et l'unicité d'une solution du fait de l'explosion combinatoire induite.

Les approches issues des travaux sur l'aide à la décision s'appuient principalement sur des modèles de décisions multi-critères (Moraitis and Tsoukias, 1999; Moraitis and Tsoukias, 1996; El-Fallah-Seghrouchni et al., 2000). Ces approches visent à coordonner des plans en s'appuyant sur un cycle de planification, négociation, exécution (PNE). Au cours de la première phase, les agents planifient de manière individuelle leurs plans puis les communiquent aux autres agents du système. Les plans individuels échangés, les agents initient la phase de négociation pour détecter les conflits éventuels, les résoudre et « créer des situations de coopération » (*i.e.,*

[1] Le terme original est *« worth »*.

dans le cas où les agents découvrent que leur collaboration améliore leurs plans individuels). La négociation s'appuie sur une échelle de critères permettant de déterminer les gains d'une situation de coopération. Son but est de maximiser les plans individuels des agents en prenant en compte leurs préférences locales tout en maximisant le plan multi-agent global.

1.3.3 L'argumentation

Les approches basées sur la théorie des jeux présentent un certain nombre d'avantages dont le plus important est peut être de pouvoir prouver certaines propriétés du protocole de négociation. Toutefois, ces approches présentent des inconvénients comme le souligne (Jennings, 2001) :

1. *Les propositions des participants de la négociation ne sont pas justifiées.* Au cours d'une négociation humaine, chaque participant justifie ses propositions, ce qui permet de comprendre comment le compromis a été établi.
2. *Les positions des participants de la négociation sont statiques.* Les techniques de négociation issues de la théorie des jeux formulent l'hypothèse que chaque agent possède une fonction d'utilité fixé a priori. Le comportement des agents est par conséquent limité. Au contraire, dans une négociation humaine, les participants cherchent à se convaincre, ce qui se traduirait par des modifications de la fonction d'utilité.

Les limitations des approches issues de la théorie des jeux ont conduit à l'émergence de mécanismes de prise de décisions basés sur l'argumentation (Sycara, 1989a). Dans le contexte multi-agent, l'argumentation est un processus par lequel un agent essaie de convaincre un autre agent de la validité (ou de l'invalidité) d'une proposition (Parson et al., 1998; Amgoud et al., 2005). Ce processus implique que les agents échangent des arguments en faveur ou contre une proposition tout en justifiant l'acceptabilité de leurs arguments. Nous reviendrons sur ces aspects dans le chapitre 3.

1.4 La coordination de plans

Dans ce paragraphe, nous nous intéressons à la coordination coopérative de plans. On parle alors de planification distribuée. Le terme de planification distribuée est toutefois ambigu (Ferber, 1995; Durfee, 2001) car il n'explicite pas ce qui est « distribué ». En effet, les plans peuvent être construits de manière centralisée par un planificateur puis décomposés en un ensemble de sous-plans pouvant être exécutés par les agents du système. Les sous-plans décomposés sont synchronisés entre eux par l'ajout d'actions de synchronisation puis alloués aux agents. Au contraire, chaque agent peut élaborer son propre plan localement et le coordonner de manière distribuée. Dans le premier cas, seule l'exécution du plan est distribuée. En revanche, dans le second, la synthèse de plans, le processus de coordination ainsi que l'exécution sont réalisés de manière complètement distribuée.

Dans un premier temps, nous introduisons les relations entre les plans, *i.e.,* les activités des agents. Dans un deuxième temps, nous présentons les mécanismes de la coordination centralisée de plans. Finalement, nous nous intéressons aux approches de la planification distribuée au travers de trois mécanismes : la planification partiellement globale, la synchronisation distribuée de plans et la fusion incrémentale de plans.

1.4.1 Les interactions entre plans

La détection des relations entre les activités des agents est un aspect fondamental de la coordination multi-agent comme l'illustrent les travaux récents dans le domaine (Clement and Barrett, 2003; Tavares-Da-Silva, 2003; D'Inverno et al., 2004). L'un des premiers à formaliser ces relations fut (Martial, 1992). Pour Von Martial, la prise en compte des relations entre plans ne fait sens que dans la mesure où chaque action est située temporellement (Allen, 1984). Par exemple, l'accès à une ressource non consommable par deux agents ne génère un conflit que si cet accès est simultané. Ceci l'amena à proposer une classification de ces relations qu'il décomposa en trois catégories :

Les relations négatives. Ce type de relations caractérise l'ensemble des intéractions entre plans qui empêchent leur exécution. La détection des relations négatives est indispensable à l'exécution des plans des agents. Ces relations peuvent avoir pour origine l'utilisation d'une ressource[2], ou bien une incompatibilité de situation :

Les conflits de ressources. Ces conflits, dus à l'accès concurrents par des agents aux ressources de l'environnement, conduisent inévitablement à des relations négatives entre les plans. Ils peuvent porter sur des ressources *« consommables »* (une ressource est consommable si et seulement si sa quantité décroît après son utilisation par un agent) ou *« non-consommables »* (une ressource est non-consommable si et seulement si sa quantité reste inchangée après son utilisation par un agent).

Les conflits de situations. Deux actions d'un plan peuvent être également conflictuelles parce qu'elles nécessitent deux états du monde exclusifs. Dans ce cas, les actions sont incompatibles et ne peuvent s'exécuter en parallèle. La détection de ce type de relations négatives (*i.e.,* la détection d'états mutuellement exclusifs du monde) est étroitement liée au domaine. Toutefois, un certain nombre de ces relations peuvent être détectées de façon générique (Blum and Furst, 1997; Chapman, 1987).

Les relations positives. Ce type de relations caractérise l'ensemble des interactions entre plans à partir desquelles il est possible d'identifier un bénéfice mutuel à l'exécution combinée de ces plans. Ces relations sont souvent négligées mais permettent un comportement global plus efficace. En effet, contrairement aux relations

[2](van der Krogt et al., 2003) propose une logique permettant de caractériser les ressources.

négatives, la détection des relations positives n'est pas nécessaire à la bonne exécution du système. Les interactions positives peuvent se produire lorsqu'un agent exécute une action validant un but ou sous-but d'un autre agent. Dans ce cas, l'agent impliqué peut modifier son plan pour tirer parti de cette relation. Ce type de relations, qui peut être perçu comme une optimisation, formule l'hypothèse que les agents sont coopératifs. Trois sous-classes de relations positives sont distinguées :

Relation d'égalité. Une relation d'égalité entre deux plans signifie qu'une même action y est présente (*i.e.,* une action est redondante). Dans ce cas, un plan global plus efficace peut être construit en supprimant l'exécution de l'action de l'un des plans individuels des agents impliqués dans la relation. Autrement dit, si une action peut être exécutée par plus d'un agent et si les agents sont interchangeables pour l'exécuter alors, un agent suffit à son exécution tandis que l'autre la supprime de son plan ;

Relation de subsomption. Une relation de subsomption entre deux plans signifie que l'exécution d'une action appartenant au plan individuel d'un agent implique la réalisation d'une autre action. Une action peut être vue comme une instance d'une action à un niveau d'abstraction supérieur. Si une action est subsumée par une autre alors seule l'action subsumée sera exécutée. Une action a_2 est subsumée par une autre action a_1 si a_2 est un raffinement de a_1 (*i.e.,* il existe une décomposition de a_1 telle que a_2 est un élément de cette décomposition) ;

Relation favorable. Une relation favorable existe entre deux plans si une action d'un plan est utile à l'exécution d'une action d'un autre plan. Ce type de relations est favorable dans la mesure où l'agent qui exécutera l'action facilite l'exécution des plans des autres agents impliqués dans la relation.

Les relations explicites. Le dernier type de relations[3] caractérise l'ensemble des interactions entre plans dont certaines des actions requièrent explicitement la réalisation d'une action par un autre agent. Ce type de relations permet de différencier l'agent qui planifie et l'agent qui exécute. Ceci traduit le désir d'un agent de faire participer un autre agent à son plan. L'ajout d'une relation explicite nécessite l'adaptation du plan de l'agent participant. Deux types de relations explicites sont définis : les relations explicites demandant la participation d'un agent particulier, *« actor request »*, et les relations explicites ne spécifiant que l'action à faire exécuter, *« action request »*.

Von Martial propose également trois classes d'actions de coordination pour résoudre les situations conflictuelles et pour tirer parti des relations favorables (cf. figure 1.6) :

Résolution de conflits temporels. La résolution des conflits temporels a pour but d'éviter l'exécution de deux actions conflictuelles dans le même intervalle de temps. Von Martial introduit deux opérateurs permettant de synchroniser les actions de manières différentes :

(a) Étendre l'intervalle (*Spread-actions*) : cette opération vise à retarder l'exécution d'une action permettant ainsi aux actions en conflit temporel de

[3] Von Martial utilise le terme de *« Request Relation »*.

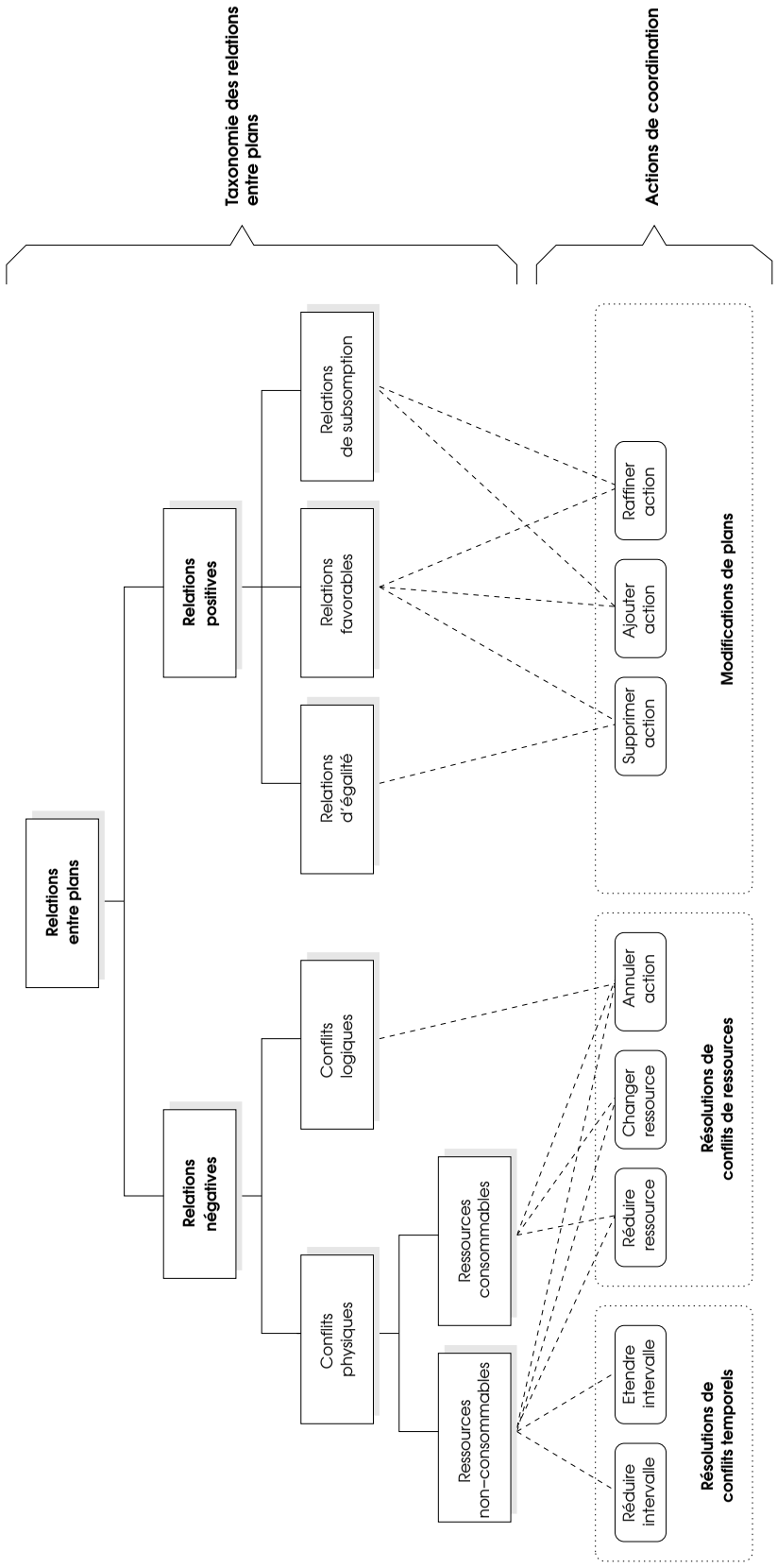


FIG. 1.6: Taxonomie des relations multi-agents.

s'exécuter sur des intervalles de temps différents. Cette opération est effectuée en ajoutant une constante de temps à la date de début et de fin d'une action ;

(b) Réduire l'intervalle (*Reduce-interval*) : cette opération vise à avancer l'exécution d'une action réduisant d'une constante de temps l'intervalle de temps associé à l'exécution d'une action.

Résolution de conflits de ressources. Dans le cas d'un conflit de ressources (*i.e.,* une ressource n'est pas présente en quantité suffisante dans l'environnement), l'action de coordination consiste à réduire la quantité de ressource nécessaire en réévaluant à la baisse les objectifs de l'agent consommateur (*reduce-resource*) ou à utiliser une ressource différente mais équivalente (*change-resource*) ou encore à annuler l'action (*cancel-action*) ;

Modification de plans. Les actions de coordination concernant la modification des plans sont associées à la prise en compte des relations favorables entre plans. Lorsqu'une relation favorable est détectée entre deux plans, une phase de négociation est initiée pour décider des modifications à apporter aux plans des agents concernés.

1.4.2 Les mécanismes centralisés de coordination

La coordination par planification centralisée repose toujours sur l'existence d'un agent coordinateur. Cet agent centralise l'ensemble des plans des agents du système et résout les conflits potentiels entre leurs activités en introduisant des actions de synchronisation. L'agent coordinateur peut : soit planifier pour l'ensemble des agents et dans ce cas il doit décomposer le plan global en sous-plans synchronisés pouvant être exécutés par les agents ; soit chaque agent peut planifier localement et dans ce cas le rôle de l'agent coordinateur se résume à la synchronisation des plans reçus.

Les approches par synchronisation

La synchronisation de plans s'appuie sur la notion de *plan non-linéaire.* En effet, un plan non-linéaire (Durfee, 2001) est un plan dont les actions ne sont pas strictement ordonnées. Par conséquent, il est possible de contraindre les relations d'ordre entres les actions en ajoutant des actions de synchronisation et de garantir ainsi que le plan sera exempt de conflits.

Cammarata, McArthur et Steeb (Cammarata et al., 1983) ont développé un système multi-agent fondé sur la coordination par planification centralisée. Leur domaine d'application est le contrôle du trafic aérien dans lequel un agent est associé à chaque avion. Le but de chaque agent est de construire un plan de vol pour atteindre sa destination tout en respectant une contrainte de distance devant le séparer des autres avions. Par l'intermédiaire d'un processus de négociation, un agent coordinateur est choisi. Chaque agent envoie les informations concernant son plan de vol à cet agent coordinateur qui les utilise pour construire un plan multi-agent. Le plan

multi-agent ainsi construit spécifie toutes les actions des différents agents du système et notamment les actions devant être exécutées pour éviter les collisions.

Georgeff (Georgeff, 1983) a proposé également une approche pour la coordination par planification centralisée dans laquelle chaque agent planifie localement un plan, puis l'envoie à un agent coordinateur. Celui-ci analyse et identifie alors les conflits entre les agents. Ces travaux ont permis la formalisation de la concurrence des activités au sein d'un système multi-agent. Son objectif est d'éviter les interférences destructrices entre les plans dues aux accès simultanés à des ressources partagées. La méthode de synchronisation proposée s'appuie sur des travaux relatifs au problème de satisfaction de contraintes[4] en évitant que des portions de plans conflictuelles soient exécutées concurremment. Georgeff modélise une action a comme une séquence d'ensemble d'états, $a = \{s_1, s_2, \ldots, s_n\}$. Une action se décompose en trois sous-parties : l'ensemble des états initiaux s_1, appelé domaine de l'action ; l'ensemble des états finaux s_n, appelé la portée de l'actions et l'ensemble des états intermédiaires $s_2, \ldots, s_{n-1}$, appelé les moments de l'action. Un problème de planification est un tuple $P = (S, I, A, G)$, où S est un ensemble d'états, I désigne l'ensemble des états initiaux $I \subseteq S$, A l'ensemble des actions primitives et G l'ensemble des états buts à atteindre $G \subseteq S$. Un plan individuel P est une séquence d'actions $a_1, a_2, \ldots, a_n$ telle que (*i*) a_1 peut être appliquée à partir de tous les états initiaux I, (*ii*) pour tout i, $1 < i < n$ les actions a_i peuvent être appliquées à partir de tous les états intermédiaires de a_{i-1} et finalement, (*iii*) l'action a_n atteint le but G (*i.e.,* la portée de a_n contient G). Un plan multi-agent pour un problème P est un ensemble de plans résolvant chaque sous-problème de P, synchronisés et applicables à l'ensemble des états initiaux I, atteignant le but G.

Les plans locaux des agents sont combinés pour identifier les conflits potentiels en essayant d'accroître leur parallélisation. L'élaboration de ces plans peut se décomposer en trois phases : (*i*) une phase d'analyse qui détermine les actions compatibles entre les activités des agents, « interaction analysis » ; (*ii*) une phase d'analyse des situations d'interblocage « safety analysis » (le rôle de cette phase est d'identifier les régions critiques dans les plans individuels des agents) ; (*iii*) finalement, une phase de synchronisation dont le but est de coordonner les régions critiques déterminées lors de la deuxième phase.

Les approches hiérarchiques

Corkill (Corkill, 1979) propose une implémentation distribuée de NOAH (Nets of Action Hierarchies) (Sacerdoti, 1977). NOAH s'appuie sur une approche hiérarchique de la planification et représente les plans, les actions et les effets des opérateurs au moyen d'un réseau procédural composé de plusieurs niveaux. À chaque niveau de la hiérarchie, un ensemble de procédures appelées *« Critics »* est déclenché afin d'identifier, de classer et de corriger les opérateurs redondants ou conflictuels présents à ce niveau de la hiérarchie. L'implémentation de Corkill est fondée sur l'allocation de sous-buts d'un certain niveau à des agents. Une telle décomposition des buts en sous-

[4] CSP : « Contraints Satisfaction Problem ».

buts est appropriée dans la mesure où l'ensemble des buts est presque totalement indépendant. Les plans sont synchronisés niveau par niveau (*i.e.,* les agents communiquent par variables partagées et résolvent les conflits de chaque niveau avant de raffiner le plan à un niveau inférieur). Un plan « *procedural net* » est un graphe dont les nœuds représentent les actions avec plus ou moins de détails. Ceux-ci sont organisés en séquences hiérarchiques partiellement ordonnées. Une action, à un niveau spécifique, est représentée par un nœud dans ce graphe. Les nœuds sont reliés pour former une description hiérarchique de l'activité d'un agent. Chaque nœud possède un ensemble de fils qui représentent les sous-actions composant l'action-père. Au cours de l'exécution, les nœuds sont exécutés en réalisant les effets de l'action du nœud-père.

Par la suite, ces travaux ont été repris dans le système SIPE-2[5] (Wilkins, 1988) qui propose une approche basée sur un réseau hiérarchique de tâches (HTN, pour *« Hierarchical Transition Networks »* (Erol et al., 1994)). Finalement, (DesJardins and Wolverton, 1999) en ont proposé une version distribuée, DSIPE. Dans cette version, les plans sont explicitement représentés comme des structures hiérarchiques de tâches et de sous-buts partiellement ordonnées. Le processus de planification repose sur l'architecture de la figure 1.7, dans laquelle chaque agent identifie les interférences entre leurs activités respectives et les propage aux agents concernés. DSIPE décompose le processus de planification en cellules, puis les coordonne en assignant et synchronisant les différentes cellules. DSIPE est un système qui a été appliqué à l'aide à la décision en introduisant un opérateur humain dans le processus de planification. Le système est responsable de l'identification des dépendances entre les sous-plans afin d'éviter les conflits potentiels au moment de leur fusion.

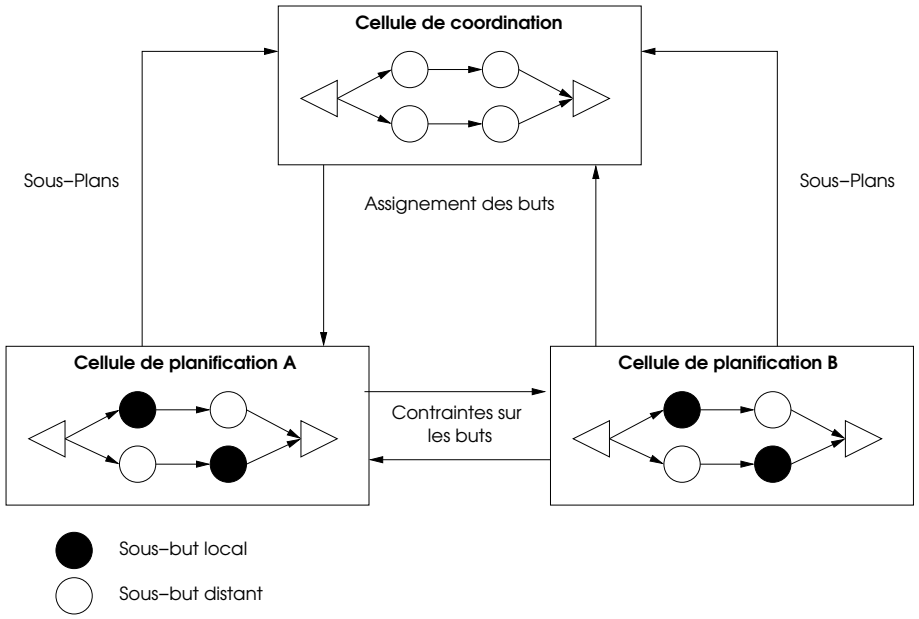


FIG. 1.7: Architecture du système DSIPE.

[5] *« System for Interactive Planning and Execution ».*

La notion d'abstraction hiérarchique se retrouve également dans d'autres planificateurs, notamment O-Plan (Tate et al., 1999) ou encore IxTeT (Laborie, 1995).

1.4.3 La planification partiellement globale

La planification partiellement globale, Pgp *« Partial Global Planning »* (Durfee and Lesser, 1987; Durfee and Lesser, 1991) est un schéma générique de coordination distribuée pour la résolution de problèmes qui s'appuie sur une architecture de type tableau noir (Erman et al., 1980). Ce schéma de coordination ne vise pas à résoudre des conflits entre les plans des agents, mais à permettre un gain de performance en termes de temps de calcul (*i.e.,* en réduisant l'inactivité de certains agents, en minimisant les tâches redondantes) en se focalisant sur les aspects d'allocation de tâches (Durfee and Lesser, 1989). Cette approche a été validée dans le cadre de réseaux de capteurs, Dsn *« Distributed Sensors Network »* appliqué à la reconnaissance de trajectoires de véhicules Dvmt *« Distributed Vehicles Monitoring Testbed »* (Lesser and Corkill, 1983). Des capteurs sonores répartis dans l'environnement doivent reconstruire la trajectoire des véhicules traversant les zones couvertes par ces capteurs (cf. figure 1.4.3 a).

Chaque agent est capable de percevoir les modifications de la zone de l'environnement dont il est responsable, et d'interagir avec les autres pour confronter ses connaissances. Les interactions entre agents reposent sur la diffusion d'une structure de données, appelée plan partiel global. Un Pgplan traduit l'activité de l'agent ainsi que ses relations avec les autres.

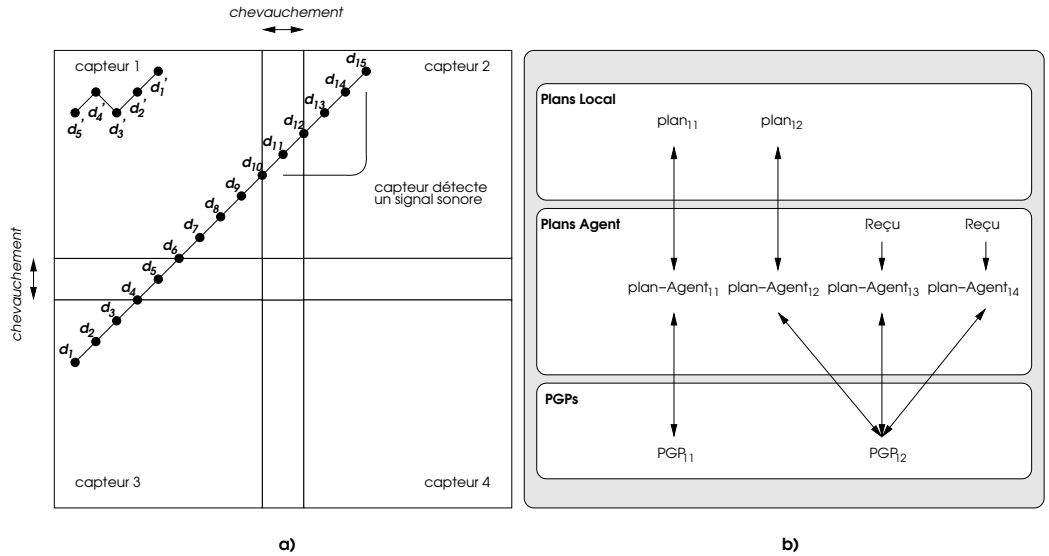


Fig. 1.8: *a) Réseau de quatre capteurs de* Dvmt *b) Vue globale des plans des agents.*

Représentation des activités des agents. Afin de coordonner ses activités, un agent doit posséder une représentation des activités des autres agents du système. Un agent modélise les activités à trois niveaux d'abstraction différents :

1. les plans locaux modélisent les perceptions locales de l'agent. Par exemple, l'agent$_1$ (cf. figure 1.4.3 a) possède deux plans locaux qui correspondent aux deux signaux sonores détectés plan$_{11}$ $\langle d'_1, \ldots, d'_5 \rangle$ et plan$_{12}$ $\langle d_4, \ldots, d_{10} \rangle$;
2. Les *« plan-agents »* modélisent l'activité globale de l'agent. Ce type de plan est une synthèse des différents plans locaux permettant de spécifier les buts poursuivis par l'agent mais également d'estimer le temps d'exécution de chaque action. À partir de ces informations, chaque agent peut construire un plan d'activité précis spécifiant les dates de début et de fin de réalisation de chaque tâche ;
3. Les plans partiellement globaux qui modélisent l'interaction entre les différents plans-agents. Chaque agent échange ses plan-agents pour détecter les buts communs entre les différents agents du système. Par exemple (cf. figure 1.4.3 b), il existe un but commun entre les agents ag_1 et ag_3 relatif à la construction de la trajectoire $\langle d_1, \ldots, d_5 \rangle$. Au contraire, l'activité d'identification de la trajectoire $\langle d'_1, \ldots, d'_{15} \rangle$ est indépendante et constitue un autre PGPLAN. Ces plans sont globaux dans le sens où ils sont partagés par plusieurs agents simultanément. Toutefois, ils restent partiels car ils ne sont pas communs à tous les agents du système. Un PGPLAN est composé de quatre informations :
   (a) *Un objectif* contenant le but du PGPLAN (*i.e.,* le but poursuivi par un agent ou un groupe d'agents) et la priorité associée à l'exécution de celui-ci ;
   (b) *Un plan d'activités « Plan Activity Map»* représentant la liste des sous-buts intermédiaires que l'agent doit atteindre. À chacun de ces sous-buts est associé les résultats attendus ainsi que la durée d'exécution prévue ;
   (c) *Un plan de construction*, *« Solution Construction Graph»*, contenant les informations relatives aux interactions (*i.e.,* quelles informations sont échangées et à quel moment) :
   (d) *Un état* contenant des références sur des informations importantes, comme la réception d'informations d'autres agents.

La coordination des activités. Elle repose sur les relations entre les buts des agents (Corkill et al., 1982; Lesser et al., 1989) et sur une organisation hiérarchique (Corkill and Lesser, 1983). Cette organisation est définie à la création du système et se base sur les compétences et les rôles des agents. Les agents propagent leurs plan-agents à l'agent coordinateur supérieur dans la hiérarchie, appelé PGPLANNER. Un agent coordinateur peut être un agent spécifique du système dont la seule activité est de coordonner celle des autres agents sous sa responsabilité. L'organisation s'appuie sur deux niveaux :

1. Le *« meta-level organization »* définit les relations hiérarchiques qui existent entre les agents. Ainsi, lorsque qu'un agent reçoit des plan-agents, il peut dé-

terminer le plan à traiter en priorité. Cette organisation est indépendante du domaine. Elle est définie statiquement lors de la création du système ;

2. Le *« domaine-level organization »* est lié au problème traité. Il caractérise les responsabilités de chaque agent dans le processus de résolution. Ce niveau peut, par exemple, définir des règles de propagation associées au placement géographique des capteurs dans l'environnement. Dans le cadre de DVMT, les agents géographiquement proches auront très souvent des buts communs et par conséquent devront plus fréquemment se coordonner.

Le PGPLANNER réalise la synthèse des différents plan-agents reçus et identifie les buts communs des agents sous sa responsabilité à partir des plans locaux. Pour chaque but identifié, il crée un PGPLAN qu'il communique aux agents concernés. L'exemple de la figure 1.4.3 b) montre la vue globale que possède l'agent coordinateur sur les agents 1, 2, 3, 4. L'agent coordinateur s'appuie sur trois stratégies de prise de décisions pour l'ordonnancement local des buts afin de minimiser l'exécution des tâches redondantes. Le PGPLANNER privilégie les actions qui réalisent de façon concurrente plusieurs buts, requièrent le moins de ressources et approuvent ou réfutent la poursuite de certains buts.

Généricité et planification partiellement globale. (Decker and Lesser, 1992; Lesser, 1998) se sont attachés à étendre le schéma originel de planification partiellement globale au travers d'un schéma de coordination appelé GPGP. Decker et Lesser formulent quatre critiques à l'approche PGP :

1. L'hétérogénéité des agents : les agents peuvent être humains ou posséder des méthodes d'expertise différentes. L'hypothèse faite par PGP d'agents homogènes (*i.e.,* partageant les mêmes critères de décision) n'est pas toujours appropriée ;
2. Le caractère dynamique des agents : l'organisation hiérarchique des agents n'est pas toujours définissable a priori. Une organisation statique peut ne pas répondre parfaitement dans le cadre d'environnements dynamiques ;
3. La gestion du temps réel : certaines applications requièrent des temps d'exécution préétablis (*i.e.,* le temps fait partie intégrante des buts des agents). PGP ne fournit pas de mécanisme pour traduire des contraintes temporelles sur les buts ;
4. La négociation pour la résolution de conflits : le schéma PGP est basé sur la diffusion de plans locaux. Toutefois, les informations transmises ne sont pas forcément pertinentes pour la prise de décisions.

GPGP introduit la notion d'engagement entre les agents ce qui permet d'améliorer la coordination entre leurs activités. Si un agent ne tient pas ses engagements, la tâche sur laquelle il s'était engagé est allouée aux autres agents. Ceci permet de prendre en compte les contraintes temporelles associées à la réalisation des buts des agents. Par ailleurs, GPGP définit un langage commun entre les agents appelé TÆMS (Decker, 1996) permettant une description précise de la structure des tâches des agents. TÆMS permet notamment d'exprimer les tâches et leurs priorités, les

méthodes de résolution mises en œuvre, les relations entre les tâches ainsi que les liens qui existent entre les tâches et les ressources. Des agents hétérogènes peuvent par conséquent communiquer de manière beaucoup plus pertinente pour résoudre les conflits inhérents à leur coordination.

1.4.4 La synchronisation distribuée de plans

Les travaux basés sur la synchronisation voient les activités des agents comme des processus concurrents qui doivent être synchronisés. Dans ce cadre, (El-Fallah-Seghrouchni and Haddad, 1996a) propose un algorithme distribué qui permet de réaliser une telle coordination. Les principaux avantages de cet algorithme résident dans la distribution du processus de coordination entre tous les agents ; l'absence de discontinuité dans l'exécution des plans (*i.e.,* aucun agent n'est suspendu au cours du processus de coordination) et l'absence de phase de régénération de plans.

Ces travaux s'appuient sur une représentation du monde par états. Plus formellement, un état du monde est représenté par un ensemble de propositions cohérentes basées sur un formalisme logique. Une action a peut se définir comme un triplet $< \mathcal{N}, \mathcal{P}, \mathcal{E} >$ de manière similaire au formalisme STRIPS (Finke and Nilsson, 1971) :

- $\mathcal{N}$, le nom de l'action a :
- $\mathcal{P}$, les préconditions qui représentent un état partiel du monde (*i.e.,* l'ensemble de conditions nécessaires à l'exécution de a) ;
- $\mathcal{E}$, les effets qui représentent l'ensemble des conditions qui seront satisfaites après l'exécution de a.

Un plan Π est un graphe orienté dont les nœuds représentent les actions et les arêtes les relations d'ordre entre les actions. La production des plans est réalisée par les agents en fonction de leurs buts locaux. Chaque agent est alors responsable de la synchronisation de ses plans avec les autres agents.

Pour qualifier un plan local validé dans un contexte multi-agent, *i.e.,* exempt de conflits, El Fallah Seghrouchni introduit la notion de *« plan structuré »*. La construction d'un plan structuré se décompose en deux phases. Dans la première, l'agent élabore un plan de manière indépendante sans tenir compte des plans produits par les autres agents. Puis dans la seconde, l'agent essaie de synchroniser son plan en le diffusant aux autres agents.

Pour bien comprendre ce processus de synchronisation, considérons un ensemble d'agents $\{g_1, g_2, \ldots, g_n\}$ auxquels on associe un plan Π_i. Les plans $\Pi_1, \Pi_2, \ldots, \Pi_{n-1}$ sont déjà synchronisés (*i.e.,* $\Pi_1, \Pi_2, \ldots, \Pi_{n-1}$ sont des plans structurés). Nous allons nous intéresser à la construction du plan structuré de l'agent g_n qui se décompose en quatre phases successives :

1. *Une phase de propagation de plan* par g_n de son plan Π_n aux agents $\{g_1, g_2, \ldots, g_{n-1}\}$;
2. *Une phase de coordination*, durant laquelle chaque agent $\{g_1, g_2, \ldots, g_{n-1}\}$ essaie de synchroniser son plan déjà coordonné avec le plan de g_n. L'agent g_i synchronise Π_i avec Π_n en créant une arête entre chaque paire d'actions dans

$End_{\Pi_i} \times Init_{\Pi_n}$, où End_{Π_i} représentent les actions finales du plan Π_i et $Init_{\Pi_n}$ les premières actions de Π_n (cf. figure 1.9 situation 1). Puis, g_i calcule les interactions entre les actions de Π_i et Π_n en prenant soin de vérifier qu'aucune des actions de Π_i ne supprime une condition nécessaire à l'exécution d'une action de Π_n (cf. figure 1.9 situation 2) ;

3. *Une phase de propagation des interactions* entre les plans pendant laquelle chaque agent g_i renvoie le résultat de la synchronisation à g_n. g_n connaît dorénavant les dépendances pour chaque action de son plan avec les autres agents ;
4. *Une phase d'exécution* au cours de laquelle g_n, avant toute exécution d'une action, attend les acquittements de synchronisation des agents g_i avant de l'exécuter.

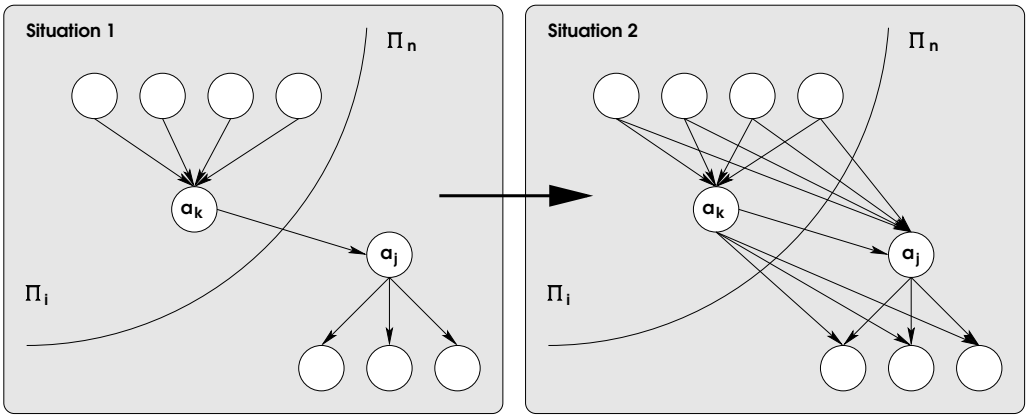


FIG. 1.9: Synchronisation de plans.

(El-Fallah-Seghrouchni and Haddad, 1996b; Boussetta et al., 1998) ont proposé d'étendre cet algorithme en introduisant les notions d'abstraction et de raffinements. Le formalisme repose sur l'utilisation des réseaux de Petri récursifs qui fournissent un cadre formel pour la validation et la gestion des plans. Les réseaux de Petri constituent des squelettes de plans qui sont manipulés (*i.e.*, fusion, raffinement) par les agents. L'objectif de cette approche n'est pas de produire un plan global, mais de garantir l'exécution des plans locaux des agents. Le modèle présente un certain nombre d'avantages :

1. Le modèle permet de représenter les actions concurrentes des agents à différents niveaux d'abstraction ;
2. La synchronisation des plans s'appuie sur la fusion de réseaux de Petri. Par conséquent, les mécanismes de synchronisation sont indépendants du domaine.
3. Le modèle tire parti des relations favorables entre les activités des agents.

1.4.5 Le paradigme de fusion incrémentale de plans

Nous présentons ici un schéma générique de coordination, l'insertion incrémentale de plans (« Plan-Merging Paradigm » PMP), présenté pour la première fois par

(Alami et al., 1994) et étendu par la suite dans (Alami et al., 1995; Robert, 1996; Alami et al., 1998b). La motivation principale de cette approche est de permettre aux agents de planifier, de coordonner leurs plans et de les exécuter d'une manière distribuée et incrémentale sans être contraints de suspendre l'exécution des plans précédemment coordonnés. Le domaine principal d'application de PMP est celui de la robotique distribuée (Alami et al., 1998a).

PMP permet à un grand nombre d'agents de concilier leurs plans « individuels », afin d'éliminer tout risque de conflit durant la phase d'exécution. Ce schéma, qui respecte au mieux l'autonomie de chaque agent, permet aux agents de prendre leurs propres décisions indépendamment d'un processus centralisé. La validation d'un plan individuel est réalisée par insertion incrémentale de ce plan dans un graphe de contraintes établi à partir des différents plans en cours d'exécution. Cette insertion transforme le plan individuel en un plan coordonné exécutable, tout en imposant des contraintes d'ordre entre les plans en cours d'exécution et le plan à valider. Cette idée a notamment été reprise dans les travaux de (Tonino et al., 2002; de Weerdt, 2003).

Ce mécanisme de coordination repose sur l'entrelacement de trois phases distinctes (cf. figure 1.10) :

- *Une phase de planification* correspondant à l'élaboration d'un plan individuel ;
- *Une phase de délibération* pendant laquelle le plan individuel est inséré dans le contexte multi-agent ;
- *Une phase d'exécution* pendant laquelle le plan coordonné, exempt de conflit avec les autres agents, est exécuté.

Lorsqu'un agent exécute un plan coordonné, il prend régulièrement connaissance de l'état courant du système afin de mener à bien les délibérations lui permettant de valider à court terme les portions suivantes de son plan individuel courant. Ce processus lui permet de prendre en compte le contexte d'exécution multi-agent. Cette opération de validation du plan individuel est appelée insertion incrémentale de plans, PMO *« Plan-Merging Operation »* (détaillée dans § 1.4.5). Les insertions de plans ont donc une portée temporelle limitée correspondant aux portions de plans individuels validées. Tout plan validé est alors considéré comme coordonné.

L'entrelacement de ces différentes phases est réalisé par l'intermédiaire d'événements propagés à tous les agents du système. L'exécution d'un plan coordonné peut déclencher une nouvelle phase de planification suivi d'une phase de délibération et ainsi initialiser une boucle planification - délibération - exécution. L'exécution devient alors un processus permanent déclenchant de manière événementielle des processus intermittents de délibération lorsque le plan coordonné se termine ou de planification lorsqu'un nouveau but est assigné à l'agent (cf. figure 1.10).

L'opération d'insertion incrémentale de plans. À la réception d'un nouveau but $\mathcal{G}_i^{k+1}$, l'agent $\mathcal{A}_i$ élabore un plan individuel $\mathcal{IP}_i^{k+1}$ pour l'atteindre. Toutefois, avant l'exécution de tout ou partie de son plan, un agent doit s'assurer que son exécution est valide dans le contexte multi-agent (*i.e.,* sans conflit avec les plans des

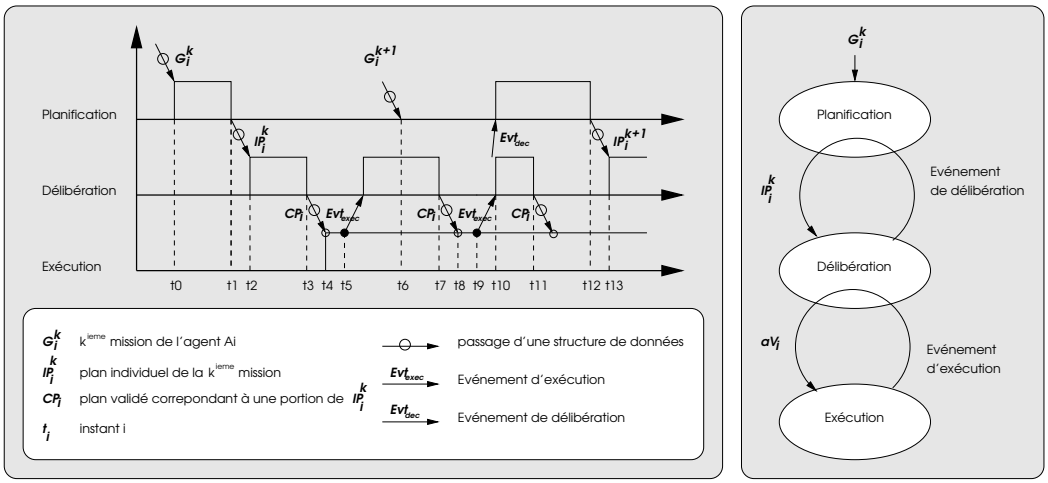


FIG. 1.10: Les boucle planification - délibération - exécution.

autres agents). Cette opération d'insertion incrémentale de plan PMO a pour résultat un *plan coordonné* $\mathcal{CP}_i$ (*i.e.,* un plan valide dans le contexte multi-agent). $\mathcal{A}_i$ collecte l'ensemble des plans coordonnés qui peuvent interférer avec son plan individuel et construit un *plan global* $\mathcal{GP}$ représentant l'ensemble des contraintes temporelles entre les différents plans coordonnés (*i.e.,* $\mathcal{GP} = \bigcup_i \mathcal{CP}_i$). $\mathcal{GP}$ constitue un graphe orienté acyclique décrivant l'état courant du système ainsi que l'ensemble des plans déjà coordonnés. Tout fonctionne comme si $\mathcal{GP}$ était produit et maintenu par l'ensemble des agents.

Si l'insertion de $\mathcal{IP}_i^{k+1}$ dans le plan global $\mathcal{GP}$ réussit, $\mathcal{IP}_i^{k+1}$ devient un plan coordonné $\mathcal{CP}_i^{k+1}$ exempt de conflit (*i.e.,* n'introduisant pas de cycle dans $\mathcal{GP}$). Lors d'un échec (*i.e.,* lorsque l'état final d'un plan d'un agent $\mathcal{A}_i$ empêche l'insertion du plan individuel $\mathcal{IP}_i^{k+1}$), deux techniques sont adoptées pour y remédier :

- *Une technique de raffinement :* le plan individuel $\mathcal{IP}_i^{k+1}$ est raffiné, ce qui permet de générer un plan à un niveau d'abstraction plus fin. Cette technique ne garantit pas une insertion du plan individuel, toutefois elle maximise sa probabilité de réussite en explorant plus en détails les causes de l'échec d'une insertion ;
- *Une technique d'attente :* $\mathcal{A}_i$ attend pour ré-exécuter une PMO qu'au moins l'un des agents interdisant l'insertion de son plan individuel réalise une PMO avec succès, modifiant ainsi le plan global et permettant l'insertion de son plan individuel $\mathcal{IP}_i^{k+1}$.

Le protocole de délibération. Pour prévenir toutes modifications concurrentes du plan global $\mathcal{GP}$, chaque PMO est réalisée en exclusion mutuelle[6]. Le processus de délibération suit le protocole défini par une machine à états finis constituée de quatre états (cf. figure 1.11) :

[6]Ceci peut être réalisé par des algorithmes à base de jetons ou par l'utilisation d'horloges de Lamport (Lamport, 1978).

- Un état d'attente correspondant à l'absence de processus de délibération en cours (état 1) ;
- Un état d'attente d'entrée en section critique : étant donné que la délibération tient compte de l'ensemble des plans des autres agents du système, toute insertion d'un nouveau plan individuel doit être réalisée de manière exclusive afin de garantir la cohérence du système (état 2) ;
- Un état de délibération (état 3) : au cours de cette étape, l'agent $\mathcal{A}_i$ ayant initié la PMO collecte l'ensemble des plans coordonnés des autres agents et construit le graphe de contraintes temporelles $\mathcal{GP}$ correspondant à l'évolution prévue du monde. La validation du plan individuel $\mathcal{IP}_i^{k+1}$ est effectuée par insertion dans $\mathcal{GP}$. L'insertion peut aboutir à deux résultats différents suivant le contexte courant :
  1. L'insertion réussit : une nouvelle portion de plan individuel de $\mathcal{A}_i$ est validée et concaténée à son plan coordonné courant. Cette réussite implique la libération du verrou associé à la section critique et un retour à l'état 1 du protocole de délibération ;
  2. L'insertion échoue en spécifiant la liste des dépendances entre les agents conduisant à l'échec de l'insertion du plan individuel de $\mathcal{A}_i$: la gestion des dépendances est effectuée dans l'état 4. Cet échec implique également la libération du verrou associé à la section critique.
- Un état de gestion d'échec de délibération (état 4). L'agent $\mathcal{A}_i$ attend la réussite d'au moins un agent qui interdisait l'insertion de son plan individuel avant de réinitier une nouvelle PMO. Les informations relatives au succès d'une PMO sont diffusées par l'intermédiaire d'événements à l'ensemble des agents du système. Ainsi, chaque agent $\mathcal{A}_i$ peut maintenir les relations de dépendances entre les différentes opérations d'insertion de plans dans un graphe de dépendances $\mathcal{PDG}_i$, *« Planning Dependency Graph »*.

Le graphe de dépendances $\mathcal{PDG}_i$ est construit de manière complètement distribuée par un mécanisme de propagation. Lorsque $\mathcal{A}_i$ initie une PMO, la liste des agents $Pred_i$ lui interdisant l'insertion de son plan individuel est vide. Si l'insertion de $\mathcal{IP}_i^{k+1}$ réussit, $\mathcal{A}_i$ propage l'information à tous les agents bloqués (*i.e.,* dans l'état 4 du protocole de délibération). Si l'insertion échoue, $\mathcal{A}_i$ construit $Pred_i$ et vérifie que les nouvelles dépendances n'introduisent pas de cycles dans $\mathcal{PDG}_i$. Dans ce cas, une situation d'interblocage est détectée. Ceci signifie que les buts des agents impliqués ne sont pas indépendants et qu'un processus de planification centralisé doit être mis en œuvre. Dans le cas contraire, $\mathcal{A}_i$ propage le graphe de dépendances contenant les nouvelles informations à l'ensemble des agents contenus dans $Pred_i$.

Lorsqu'un agent $\mathcal{A}_k$ reçoit un graphe de dépendances, il propage à son tour $\mathcal{PDG}_i$ à l'ensemble des agents contenus dans $Pred_k$ diffusant le nouveau graphe de dépendances à tous les agents du système. Ce mécanisme de diffusion est connu sous le terme de « vague », et a fait l'objet de nombreuses recherches dans le domaine des systèmes distribués et répartis (Lynch, 1996).

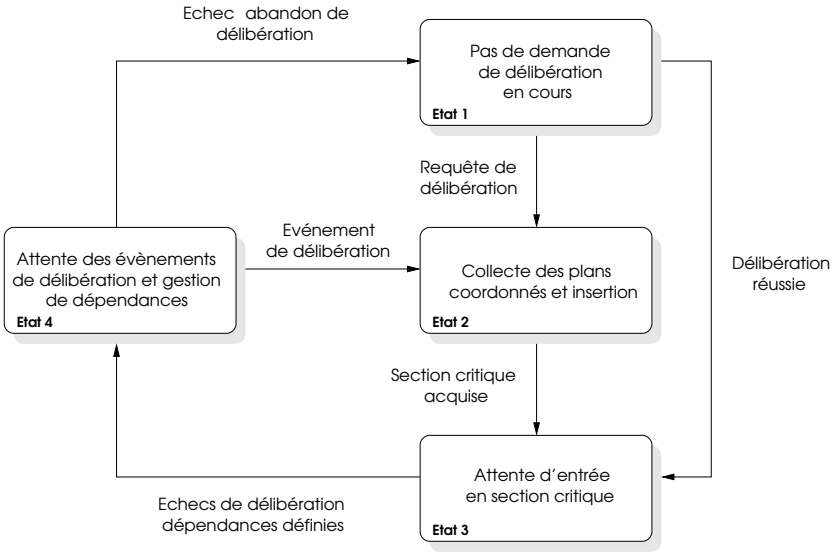


FIG. 1.11: Le protocole de délibération.

La stratégie de résolution d'interblocages. Dans la section précédente, nous avons mis en évidence les problèmes liés aux interblocages dus à l'interdépendance forte entre les buts des différents agents du système. Pour remédier à cette difficulté, une extension du paradigme a été proposée par (Qutub, 1997), permettant au système de converger progressivement vers une solution de plus en plus centralisée lorsque la situation devient de plus en plus complexe.

Lorsque $\mathcal{A}_i$ détecte une situation d'interblocage, il possède toutes les informations nécessaires, $\mathcal{PDG}_i$, pour élaborer un plan pour tous les agents bloqués. L'ensemble des agents bloqués, $\mathcal{DL}_i^j$, sont dans l'impossibilité d'ajouter de nouvelles portions de plans à leur plan coordonné. Par conséquent, à la fin de l'exécution de leur plan coordonné, l'ensemble des agents de $\mathcal{DL}_i^j$ seront bloqués dans l'état 4 du protocole de délibération. Pour résoudre cet interblocage, $\mathcal{A}_i$ est nommé *agent coordinateur* pour l'ensemble des agents de $\mathcal{DL}_i^j$. L'agent coordinateur construit un plan permettant d'atteindre l'union de tous les buts des agents de $\mathcal{DL}_i^j$. Si un tel plan existe alors $\mathcal{A}_i$ décompose et synchronise le plan global en sous-plans pouvant être exécutés par les agents de $\mathcal{DL}_i^j$. Puis $\mathcal{A}_i$ insère chaque sous-plan dans le plan global afin de les valider. L'insertion peut aboutir à deux résultats différents suivant le contexte courant :

1. L'insertion réussit. L'agent coordinateur envoie à tous les agents dans $\mathcal{DL}_i^j$ son plan à exécuter et chaque agent contenu dans $\mathcal{DL}_i^j$ retrouve son autonomie ;
2. L'insertion échoue. L'agent coordinateur élabore la liste des dépendances $Pred_i$ et vérifie que ces dépendances n'introduisent pas à nouveau de cycle dans le graphe de dépendances. Si aucun cycle n'est détecté, l'agent coordinateur diffuse le nouveau graphe de dépendances à l'ensemble des agents du système et attend qu'un agent de $Pred_i$ réussisse une nouvelle PMO. Dans le cas contraire, l'agent coordinateur met à jour la liste des agents bloqués $\mathcal{DL}_i^{j+1}$ et applique

récursivement le même processus de coordination avec un plus grand ensemble d'agents.

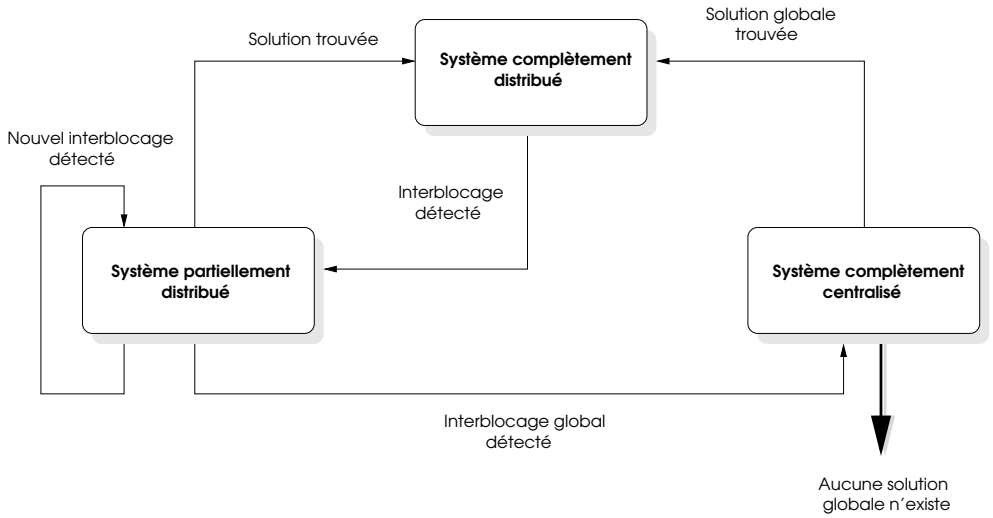


FIG. 1.12: Stratégie de résolution d'interblocages.

Il est à noter que plusieurs interblocages peuvent se produire parallèlement et être résolus de manière indépendantes. Dans un cas extrême, le système peut converger vers une solution complètement centralisée dans laquelle un agent coordinateur planifie pour l'ensemble des agents du système (cf. figure 1.12).

Conclusion

Ce chapitre a présenté une vue générale des mécanismes de coordination et de coopération mis en œuvre dans le cadre des systèmes multi-agents. Dans un premier temps, nous avons décrit la problématique liée à la coopération entre agents autonomes. Puis, nous avons introduit les mécanismes basés sur les structures organisationnelles ainsi que les techniques de prise de décisions distribuées. Finalement, nous avons plus particulièrement traité les aspects associés à la coordination de plans.

Tous les mécanismes présentés cherchent à maintenir l'intégrité fonctionnelle du système, en garantissant que les activités des agents sont exemptes de conflits. En revanche, l'objectif de ces approches n'est pas de produire un plan global. Bien que ces mécanismes apportent un certain nombre de réponses à la problématique de la coordination et du contrôle distribué, ils ne sont pas totalement satisfaisants et ceci pour les raisons suivantes :

1. La coopération entre les agents existe uniquement lorsqu'un conflit doit être résolu. Par conséquent, ces mécanismes ne mettent en œuvre la coopération que dans un cadre limité et ne permettent pas un réel partage de compétences et de croyances entre les agents pour réaliser une tâche commune.

2. Le découpage en trois phases des mécanismes de coordination : une phase de planification locale puis une phase de coordination et finalement une phase d'exécution, met en évidence un certain nombre de limites lorsque les buts des agents sont fortement couplés. En effet, ces trois phases ne sont pas totalement indépendantes. Ceci oblige certains modèles de coordination à avoir recours à un traitement plus ou moins centralisé de la coordination. De plus, la phase de planification se limite souvent à l'instanciation de plans préétablis au sein d'une librairie à cause du surcoût important engendré par la synthèse dynamique de plans.

Chapitre 2

Modèles de coopération par états mentaux

Introduction

Les modèles cognitifs ou « Bdi » postulent que les activités d'un agent, tout comme celles d'un être humain, doivent être dictées par des représentations abstraites du monde, incluant la représentation de ses propres capacités, de ses buts ainsi que ceux des autres agents : ces représentations sont les états mentaux, *« Belief »*, *« Desire »* et *« Intention »*. L'approche de la coopération, présentée dans ce chapitre, propose une vue d'ensemble de ces modèles.

Ce chapitre est organisé de la manière suivante : dans un premier temps, nous présentons la théorie de l'intention conjointe (§ 2.1) ; puis nous introduisons sa généralisation dans le contexte de la résolution coopérative de problèmes distribués (§ 2.2) ; dans un deuxième temps, nous nous intéressons à la théorie des plans partagés qui formalise les états mentaux nécessaires à la collaboration pour construire un plan et finalement (§ 2.3), nous terminons par la présentation des principales implémentations des modèles de coopération reposant sur ces deux théories (§ 2.4).

2.1 De l'intention à la coopération

La théorie de l'intention conjointe (Levesque et al., 1990; Cohen and Levesque, 1991) s'appuie sur un modèle Bdi dans lequel l'intention est vue comme un choix suivi d'un engagement à exécuter une action (Cohen and Levesque, 1990).

2.1.1 Engagement et intention individuels

Cette théorie est décrite par une logique modale : (BEL x p) et (GOAL x p) exprime que x croit que la proposition p est satisfaite et possède le but p ; (MB x y p) exprime que x et y croient mutuellement que p est vraie ; (AGT $x_1 \ldots x_n$ e) exprime

que seuls les agents $x_1 \ldots x_n$ produisent la séquence d'événements e ; $e_1 \leq e_2$ exprime que e_1 est une sous séquence de e_2 et finalement (HAPPENED a), (HAPPENING a), (HAPPENS a) expriment qu'une séquence d'événements décrite par une expression représentant respectivement une action a vient de se produire, est en train de se produire ou se produira. Une expression représentant une action est construite à partir de variables en utilisant la logique suivante : a ;b traduit la composition des actions a et b ; a|b traduit le choix non déterministe entre a et b ; a||b traduit la concurrence entre a et b ; p ? traduit une action de test et finalement a* traduit la répétition.

Les opérateurs BEL et GOAL sont définis dans la sémantique des mondes possibles, dans laquelle un monde est modélisé par une fonction donnant, pour chaque instant, un ensemble d'événements primitifs. Il est formulé l'hypothèse que chaque agent peut accéder à ses croyances et à ses buts. Une expression est évaluée en fonction de l'état du monde mais également à un instant de ce monde. La valeur de vérité d'une expression utilisant HAPPENED, HAPPENING, HAPPENS diffère seulement par l'instant à laquelle elle est évaluée : immédiatement après, pendant, immédiatement avant. La théorie de l'intention conjointe utilise les abréviations suivantes :

Définition. 2.1 (Actions)

$$\begin{aligned}
(\text{DONE } x_1 \ldots x_n \text{ a}) &\stackrel{\text{def}}{=} (\text{HAPPENED a}) \wedge (\text{AGT } x_1 \ldots x_n \text{ a}) \\
(\text{DOING } x_1 \ldots x \text{ a}) &\stackrel{\text{def}}{=} (\text{HAPPENING a}) \wedge (\text{AGT } x_1 \ldots x_n \text{ a}) \\
(\text{DOES } x_1 \ldots x_n \text{ a}) &\stackrel{\text{def}}{=} (\text{HAPPENS a}) \wedge (\text{AGT } x_1 \ldots x_n \text{ a})
\end{aligned}$$

Définition. 2.1 (Éventualité)
Une séquence d'événements se produit, incluant la séquence d'événements vide, après laquelle p est vraie (*i.e.,* p sera atteint dans le futur) :

$$\Diamond p \stackrel{\text{def}}{=} \exists e \, (\text{HAPPENS e ; p ?})$$

Définition. 2.2 (Toujours)
La formule bien formée p est toujours vraie :

$$\Box p \stackrel{\text{def}}{=} \neg \Diamond \neg p$$

Définition. 2.3 (Jusqu'à)
Jusqu'à ce que la formule bien formée p soit vraie, q demeura vraie :

$$(\text{UNTIL p q}) \stackrel{\text{def}}{=} \forall c \, (\text{HAPPENS c ; q ?}) \supset \exists a \, (a \leq c) \wedge (\text{HAPPENS a ; p ?})$$

À partir de ces définitions, il est possible d'exprimer l'engagement d'un agent x à réaliser un but p : l'agent doit croire que p est faux mais veut qu'il devienne vrai dans le futur et continue à vouloir satisfaire p jusqu'à ce qu'il croit que le but est atteint ou qu'il ne le sera jamais. Ceci introduit la notion de but persistant :

Définition. 2.2 (But persistant)

$$(\mathsf{PGOAL}\ x\ p\ q) \stackrel{\mathsf{def}}{=} (\mathsf{BEL}\ x\ \neg p) \wedge (\mathsf{GOAL}\ x\ \Diamond p) \wedge (\mathsf{UNTIL}\ [(\mathsf{BEL}\ x\ p) \vee (\mathsf{BEL}\ x\ \Box\neg p) \vee (\mathsf{BEL}\ x\ \neg q)]\ (\mathsf{GOAL}\ x\ \Diamond p))$$

L'expression q permet d'exprimer une condition sur la conservation du but p. Cette condition est le plus souvent utile dans une expression telle que (PGOAL x p (GOAL x q)) pour exprimer l'engagement à réaliser p comme étant un sous but de q. Finalement, l'intention d'un agent x d'exécuter une action a se définit de la manière suivante :

Définition. 2.3 (Intention)

$$(\mathsf{INTEND}\ x\ a\ q) \stackrel{\mathsf{def}}{=} (\mathsf{PGOAL}\ x\ (\mathsf{DONE}\ x\ [\mathsf{UNTIL}\ (\mathsf{DONE}\ x\ a)\ (\mathsf{BEL}\ x\ (\mathsf{DOING}\ x\ a))]\ ?\ ;a)\ q)$$

Un agent a l'intention d'exécuter une action s'il possède un but persistant pour réaliser cette action et croit qu'il est en train de l'exécuter.

2.1.2 Engagement et intention conjoints

Un groupe d'agents Θ possède l'intention conjointe de réaliser une action si chaque agent s'est conjointement engagé à réaliser cette action et qu'ils croient mutuellement la réaliser. L'engagement conjoint se définit comme étant un but conjoint persistant (JPG). La première tentative de formalisation s'est intéressée à remplacer la notion de PGOAL par celle de croyance mutuelle (MB) et la notion de (GOAL x ◊p) par la croyance mutuelle d'un but (MG) :

Définition. 2.4 (But conjoint persistant)

$$(\mathsf{JPG}\ \Theta\ p\ q) \stackrel{\mathsf{def}}{=} (\mathsf{MB}\ \Theta\ \neg p) \wedge (\mathsf{MG}\ \Theta\ p) \wedge (\mathsf{UNTIL}\ [(\mathsf{MB}\ \Theta\ p) \vee (\mathsf{MB}\ \Theta\ \Box\neg p) \vee (\mathsf{MB}\ \Theta\ \neg q)]\ (\mathsf{MG}\ \Theta\ p))$$

où

$$(\mathsf{MG}\ \Theta\ p) \stackrel{\mathsf{def}}{=} (\mathsf{MB}\ \Theta\ (\forall a \in \Theta\ (\mathsf{GOAL}\ a\ \Diamond p)))$$

Cependant, cette définition n'est pas tout à fait correcte. Supposons qu'un agent x vienne à croire que le but p est inatteignable ; x doit abandonner ce but et, par conséquent, (MG Θ p) n'est plus satisfait. Toutefois, il ne peut abandonner ce but tant que les autres agents du groupe ne possèdent pas la connaissance mutuelle que ce but est inatteignable. Un engagement conjoint ne peut être considéré comme un simple engagement individuel où le groupe d'agents est perçu comme un seul agent, pour la simple raison que les membres du groupe peuvent avoir des croyances divergentes.

Lorsqu'un agent membre d'un groupe découvre que le but poursuivi est inatteignable ou atteint, il doit nécessairement en informer les autres. Pour y remédier, la notion d'engagement conjoint introduit la notion de *« but faible »*[1] (WG) qui traduit l'état d'un agent x désirant atteindre le but p relativement à un groupe d'agents Θ :

Définition. 2.5 (But faible)

$$
\begin{aligned}
(\mathsf{WG}\ x\ \Theta\ q) \stackrel{\mathsf{def}}{=} \\
&[\neg(\mathsf{BEL}\ x\ p) \wedge (\mathsf{GOAL}\ x\ \Diamond p)] \vee \\
&[(\mathsf{BEL}\ x\ p) \wedge (\mathsf{GOAL}\ x\ \Diamond(\mathsf{MB}\ x\ \Theta\ p))] \vee \\
&[(\mathsf{BEL}\ x\ \Box\neg p) \wedge (\mathsf{GOAL}\ x\ \Diamond(\mathsf{MB}\ x\ \Theta\ \Box\neg p))]
\end{aligned}
$$

La notion de but faible implique trois cas mutuellement exclusifs : soit x possède le but $\Diamond$p, ou pense que p est atteint et veut que la croyance « p ne sera jamais atteint » soit partagée par tous les membres du groupe.

Lorsqu'un agent est conjointement engagé à la réalisation d'un but p, les agents du groupe ne font plus l'hypothèse que chacun d'entre eux possède le but p, mais seulement qu'ils possèdent p comme but faible. Ceci permet à chaque agent de croire localement que p est inatteignable ou atteint indépendamment d'une croyance mutuelle et d'avoir le but d'en informer les autres. Par conséquent, la définition d'un JPG, *« Joint Persitant Goal »* s'exprime de la manière suivante :

Définition. 2.6 (But conjoint persistant)

$$
\begin{aligned}
&(\mathsf{JPG}\ \Theta\ p\ q) \stackrel{\mathsf{def}}{=} \\
&\quad (\mathsf{MB}\ \Theta\ \neg p) \wedge (\mathsf{MG}\ \Theta\ p) \wedge \\
&\quad (\mathsf{UNTIL}\ [(\mathsf{MB}\ \Theta\ p) \vee (\mathsf{MB}\ \Theta\ \Box\neg p) \vee (\mathsf{MB}\ \Theta\ \neg q)]\ (\mathsf{WMG}\ x\ p)) \\
&\text{où} \\
&\quad (\mathsf{WMG}\ \Theta\ p) \stackrel{\mathsf{def}}{=} (\mathsf{MB}\ \Theta\ (\forall x, \forall y \in \Theta\ (\mathsf{WG}\ x\ y\ p))
\end{aligned}
$$

Ayant donné la définition de l'engagement conjoint, il est possible de définir la notion d'intention conjointe comme étant une généralisation de l'intention individuelle :

Définition. 2.7 (Intention conjointe)

$$
\begin{aligned}
&(\mathsf{JI}\ \Theta\ p\ q) \stackrel{\mathsf{def}}{=} \\
&\quad (\mathsf{JPG}\ \Theta\ (\mathsf{DONE}\ \Theta\ [\mathsf{UNTIL}\ (\mathsf{DONE}\ \Theta\ a)\ (\mathsf{MB}\ \Theta\ (\mathsf{DOING}\ \Theta\ a))]\ ?;a)\ q)
\end{aligned}
$$

[1] Dans la version originelle un but faible est noté WG mais par la suite WAG dans (Smith and Cohen, 1996).

2.1.3 Le protocole d'engagement

La définition de l'engagement et de l'intention conjointe fait apparaître que les agents d'un groupe, désirant réaliser une tâche collective, doivent satisfaire des croyances et des engagements mutuels pour établir un JPG. Il en découle un nécessaire échange de requêtes et de confirmations basées sur des actes de langage permettant ainsi au groupe de parvenir à un consensus relatif à leurs croyances et à leurs engagements mutuels (Smith and Cohen, 1996). Cette phase d'échanges est appelée protocole *d'établissement d'engagements* puisqu'elle permet l'établissement d'un JPG. Supposons qu'un agent x appartenant au groupe Θ initie le protocole d'établissement d'engagements, il peut alors se décomposer en trois étapes :

1. x envoie une requête en précisant le WG qu'il désire atteindre. Cette requête traduit non seulement le but de x de réaliser p mais également celui de le faire adopter aux autres membres du groupe. Après cette requête, tous les agents partagent la croyance que x a adopté le WG p.
2. Chaque membre du groupe répond par une confirmation ou un refus relatif à l'adoption du but faible p. Lorsqu'un agent confirme le but faible, il en informe également les autres.
3. Si tous les agents répondent par une confirmation à la requête de x, alors les conditions d'établissement d'un JPG(Θ, p) sont satisfaites.

Au cours de l'établissement d'un JPG, le protocole d'établissement d'engagements synchronise les agents du groupe. En particulier, grâce à ce protocole, les agents entrent simultanément dans un engagement conjoint pour réaliser une tâche collective représentée par le but p. À l'étape 1, l'adoption d'un WG par x implique que si x croit dorénavant localement que p est atteint ou inatteignable, il doit en informer les autres agents. L'étape 2 contraint de manière similaire les autres agents du groupe. Comme l'étape 3 l'indique, tous les agents du groupe doivent parvenir à un consensus pour qu'un JPG soit établi. Un JPG n'est, par conséquent, établi que lorsque tous les membres du groupe l'ont confirmé. Ce protocole permet de propager toutes modifications concernant le but à atteindre. Ainsi, le suivi par chaque agent de l'état de la tâche collective est assuré.

2.2 Les état mentaux et la résolution de problèmes

Dans ce paragraphe, nous proposons une généralisation de la coordination par intention conjointe à la résolution coopérative de problèmes distribués. Ces travaux (Wooldridge and Jennings, 1994b; Wooldridge and Jennings, 1994a; Wooldridge and Jennings, 1999) proposent une décomposition simple du processus de résolution qui peut être découpée en 4 phases successives :

1. *une phase d'identification d'actions collaboratives* : cette identification peut survenir lorsqu'un agent doit atteindre un but qu'il ne peut réaliser seul ou bien parce qu'il privilégie une solution coopérative.

2. *une phase de regroupement* : au cours de cette phase, les agents ayant identifié une tâche coopérative lors de la première phase sollicitent une assistance pour réaliser leurs buts. Si cette phase réussit, alors l'ensemble des agents du groupe sont collectivement engagés à réaliser une action coopérative.
3. *une phase de planification* : au cours de cette phase les agents tentent de négocier un plan conjoint en se basant sur leurs croyances pour atteindre le but désiré.
4. *une phase d'exécution* : au cours cette dernière phase le plan collectif est exécuté par les agents du groupe.

2.2.1 L'identification d'actions coopératives

La phase préliminaire du processus coopératif de résolution de problèmes est l'identification d'actions coopératives. Il est possible de distinguer deux raisons principales qui conduisent un agent à initier cette phase. Tout d'abord, un agent peut être dans l'impossibilité d'atteindre son but seul. L'agent doit, par conséquent pour parvenir à l'atteindre, coopérer. Dans le cas contraire, il peut délibérément choisir de privilégier une action coopérative pour des raisons propres à l'agent (*e.g.,* but atteint plus rapidement de manière coopérative, préservation de ses ressources propres, *etc.*).

Définition. 2.4 (Capacité d'un agent)
Un agent i peut satisfaire un but φ, si et seulement si, il existe une action complexe α telle que i est le seul agent à savoir qu'après l'exécution de α, φ sera satisfait ou qu'il pourra satisfaire φ.

$$\begin{aligned}(\mathsf{CAN\ i}\ \varphi) \stackrel{\mathsf{def}}{=}\ & \exists\alpha\ (\mathsf{KNOW\ i}\ (\mathsf{AGT}\ \alpha\ \mathsf{i}) \wedge (\mathsf{ACHEIVES}\ \alpha\ \varphi)) \vee \\ & \exists\alpha\ (\mathsf{KNOW\ i}\ (\mathsf{AGT}\ \alpha\ \mathsf{i}) \wedge (\mathsf{ACHIEVES}\ \alpha\ (\mathsf{CAN\ i}\ \varphi)))\end{aligned}$$

où $(\mathsf{KNOW\ i}\ \varphi) \stackrel{\mathsf{def}}{=} \varphi \vee (\mathsf{BEL\ i}\ \varphi)$ et

$(\mathsf{ACHIEVES}\ \alpha\ \varphi) \stackrel{\mathsf{def}}{=} \mathsf{HAPPENS}(\alpha) \Longrightarrow \mathsf{HAPPENS}(\alpha\,;\varphi\,?)$

Définition. 2.5 (Capacité d'un groupe d'agents)
Un groupe d'agent g peut satisfaire un but φ, si et seulement si, il existe une action complexe α et un groupe d'agents g' tel que chaque agent de g croit mutuellement que les agents de g' peuvent réaliser α, et que chaque agent de g croit qu'après l'exécution de α, φ sera satisfait ou que g aura la capacité de satisfaire φ.

$$\begin{aligned}(\mathsf{JCAN\ g}\ \varphi) \stackrel{\mathsf{def}}{=}\ & \exists\alpha,\ \mathsf{g'}\ (\mathsf{MKNOW\ g}\ (\mathsf{g'} \subseteq \mathsf{g}) \wedge (\mathsf{AGT}\ \alpha\ \varphi) \wedge (\mathsf{ACHIEVES}\ \alpha\ \varphi)) \vee \\ & \exists\alpha,\ \mathsf{g'}\ (\mathsf{MKNOW\ g}\ (\mathsf{g'} \subseteq \mathsf{g}) \wedge (\mathsf{AGT}\ \alpha\ \varphi) \wedge (\mathsf{ACHIEVES}\ \alpha\ (\mathsf{JCAN\ g}\ \varphi)))\end{aligned}$$

où $(\mathsf{MKNOW\ g}\ \varphi) \stackrel{\mathsf{def}}{=} \varphi \vee (\mathsf{MB\ g}\ (\mathsf{MKNOW\ g}\ \varphi))$

Définition. 2.6 (Coopération)
Une situation de coopération (PfC[2]) existe pour un agent i qui possède le but φ, si et seulement si, il existe un groupe g tel que i croit que g peut conjointement réaliser φ, et que i ne peut pas réaliser seul φ ou alors que i croit que pour toutes actions α qui permettent de réaliser φ, il possède le but de ne pas exécuter α.

$$\begin{aligned}(\mathsf{PfC\ i}\ \varphi) \stackrel{\mathsf{def}}{=} &(\mathsf{GOAL\ i}\ \varphi) \wedge \exists \mathsf{g}\ (\mathsf{BEL\ i}\ (\mathsf{JCAN\ g}\ \varphi)) \wedge \\ &[\neg(\mathsf{CAN\ i}\ \varphi) \vee \\ &(\mathsf{BEL\ i}\ \forall\alpha\ (\mathsf{AGT}\ \alpha\ \mathsf{i}) \wedge (\mathsf{ACHIEVES}\ \alpha\ \varphi) \Longrightarrow (\mathsf{GOAL\ i}\ (\neg\mathsf{DOES}(\alpha))))]\end{aligned}$$

2.2.2 La formation d'un groupe

Lorsque les actions pouvant être réalisées de manière coopérative sont identifiées, un agent doit solliciter l'assistance d'un groupe d'agents capables de réaliser le but. Si la formation du groupe réussit alors tous les agents de ce groupe sont conjointement engagés à exécuter une action collective atteignant ce but. Toutefois, un agent ne peut garantir que la formation du groupe réussira. On parle alors de tentative de regroupement.

Définition. 2.7 (Tentative de regroupement)
Une tentative de regroupement initiée par un agent i pour atteindre un but φ est l'exécution d'une action complexe α après laquelle φ est satisfait ou au moins un sous-état ψ.

$$\{\mathsf{ATTEMPT\ i}\ \alpha\ \varphi\ \psi\} \stackrel{\mathsf{def}}{=} \left[\begin{array}{ll}(\mathsf{BEL\ i}\ \neg\varphi) \wedge (\mathsf{AGT}\ \alpha\ \mathsf{i}) & \wedge \\ (\mathsf{GOAL\ i}\ (\mathsf{ACHIEVES}\ \alpha\ \varphi)) & \wedge \\ (\mathsf{INTEND\ i}\ (\mathsf{DOES}\ \alpha ; \psi\,?)) & \end{array}\right] ?;\alpha$$

La proposition φ est le but que l'agent cherche à atteindre et ψ la proposition suffisante pour que φ soit satisfaite. Si le regroupement réussit alors chaque agent du groupe possède un engagement conjoint à réaliser le but de i.

2.2.3 La formation de plans

Lorsque la phase de regroupement est validée, l'ensemble des agents est conjointement engagé à réaliser une action collective et croit pouvoir atteindre le but désiré. Toutefois, l'action collective ne peut débuter que lorsque les agents se sont accordés sur les sous-actions à exécuter. Par conséquent, la troisième étape du processus de résolution est la *formation de plans*.

Initialement, la coalition d'agents connaît au moins une action permettant de se rapprocher du but ou même de l'atteindre. Cependant, un agent peut croire que l'exécution d'une action entraîne la non satisfaction des propositions nécessaires pour

[2] *« Potential for Cooperation »*.

atteindre le but. Il apparaît alors nécessaire à la coalition de parvenir à un consensus pour définir exactement la succession d'actions à réaliser (cf. § 1.3). Atteindre un consensus au sein d'un groupe d'agents implique que les agents soient capables de raisonner sur les plans et les actions ; de formuler des arguments pour ou contre une action et de proposer des solutions alternatives jusqu'à ce qu'un compromis soit atteint.

2.2.4 La phase d'exécution

Si la troisième phase de formation de plans réussit, la coalition est engagée conjointement sur la manière de réaliser le but qui lui a été confié. Cette quatrième phase, dite d'exécution, requiert simplement que la coalition d'agents ait l'intention conjointe de réaliser les actions du plan précédemment discutées.

Définition. 2.8 (Une action collective)
Un groupe d'agents g est considéré comme une coalition désirant atteindre le but φ s'il existe une action α telle que : (*i*) α atteint φ et (*ii*) chaque agent i de g a l'intention conjointe de réaliser le but φ.

$$(\mathsf{TEAM}\ \mathsf{g}\ \varphi\ \mathsf{i}) \stackrel{\mathsf{def}}{=} \exists\ \alpha\ (\mathsf{ACHIEVES}\ \alpha\ \varphi) \wedge (\mathsf{JI}\ \mathsf{g}\ \alpha\ (\mathsf{GOAL}\ \mathsf{i}\ \varphi))$$

La définition de l'intention conjointe (JI) permet d'assurer que l'ensemble de la coalition reste mutuellement engagée à réaliser l'action collective. De plus, elle garantit que si l'un des agents croit que le but n'est plus atteignable alors il en informera la coalition et l'action collective se terminera (cf. § 2.1.1).

2.3 La théorie des plans partagés

Cette théorie fut originellement élaborée pour modéliser la structure intentionnelle d'un discours (Grosz and Sidner, 1990). Par la suite, la formalisation a été généralisée dans le cadre de plus de deux agents pour permettre la prise en compte de la construction de coalitions d'agents collaboratifs (Grosz and Kraus, 1996; Grosz and Kraus, 1999). Ce modèle a été adapté dans plusieurs domaines : pour modéliser la structure intentionnelle du dialogue (Lochbaum, 1995; Lochbaum, 1998) ; pour structurer des interfaces homme-machine (Ortiz and Grosz, 2002) ; ou encore dans le projet STEAM (Tambe, 1997).

2.3.1 Le modèle intentionnel de la collaboration

La théorie des plans partagés (Grosz and Kraus, 1993) est un modèle de coordination basé sur les intentions et les croyances des agents[3]. Ce modèle formule deux hypothèses. Tout d'abord, la coordination ne peut être perçue comme la simple

[3]On retrouve également cette idée dans les travaux de (Mandiau, 1993). Ses travaux reposent sur le concept que tout plan construit par un agent reflète ses différents états intentionnels.

analyse des activités en termes de plans individuels des agents, mais plutôt comme un processus d'intégration des intentions et des croyances des différents agents collaboratifs. Cette approche de la coordination peut également être présentée comme un processus de raffinement itératif de plans par des agents.

La théorie des plans partagés s'appuie sur les notions de plans et d'états mentaux (Pollack, 1990). Le modèle cognitif de Pollack ne prend pas en compte la notion de coopération. L'exécution d'une action par un agent est définie à deux niveaux. Au niveau des croyances de l'agent, l'exécution d'une action α implique que l'agent possède les connaissances (*i.e.,* une succession d'actions β_i) permettant d'exécuter l'action α (l'ensemble des actions β_i constituant la recette R_α pour réaliser α) et que chaque action β_i est exécutable par un agent. Au niveau des intentions de l'agent, l'exécution de α se traduit par l'intention d'effectuer chaque action de R_α et l'intention d'effectuer α en appliquant la recette R_α. Cette formalisation de la coopération permet de traiter des actions complexes et des plans partiels ; de modéliser les délégations de tâches entre les agents ; de fournir un entrelacement entre les phases de planification et d'exécution, et de distinguer les informations nécessaires aux agents directement impliqués dans le processus de planification.

La théorie des plans partagés distingue deux formes d'intention ; (1) « *l'intention de* » réaliser une action et (2) « *l'intention que* » une proposition soit satisfaite. Un agent, qui possède l'intention de réaliser une action, doit non seulement s'être engagé à exécuter cette action, mais il doit également posséder les croyances relatives aux compétences requises en termes de plans pour la réaliser (*i.e.,* individuellement ou collectivement). Un agent, qui possède l'intention qu'une proposition soit satisfaite, doit s'être engagé à mettre en œuvre toutes ses compétences pour que cette proposition soit vérifiée. Cette seconde forme d'intention est utilisée pour représenter l'engagement des agents envers l'activité du groupe (*i.e.,* formaliser la notion de collaboration entre les agents) et fournir les bases nécessaires aux agents pour construire des croyances mutuelles (Grosz and Kraus, 1999). Ces deux formes d'intention sont raffinées en introduisant la notion de potentialité. Cette notion permet d'exprimer, au travers d'*intentions potentielles*, les choix possibles de l'agent au cours de sa délibération et avant tout engagement de sa part.

2.3.2 La définition de la notion de plan

La théorie des plans partagés distingue les *plans complets* et les *plans partiels*. Un *plan complet partagé* (FSP) caractérise un plan dans lequel chaque action est réalisée par un agent du groupe. Par conséquent, un FSP est le résultat de la construction collaborative des activités coordonnées de la coalition d'agents et peut être défini comme le but du processus d'élaboration collaboratif de plans (*i.e.,* l'ensemble des états mentaux que doivent posséder les agents pour exécuter l'action qui leur a été confiée). Au cours du processus collaboratif de construction du plan, toutes les actions ne sont pas entièrement définies. On parle alors de plans partagés partiels (PSP). Un PSP définit les états mentaux minimaux requis pour exprimer la collaboration entre les agents et les critères gouvernant le processus de construction du plan complet.

Les agents impliqués dans le processus de construction d'un plan pour réaliser une action doivent posséder un certain nombre de croyances (individuelles ou mutuelles), relatives à la façon dont les actions et les sous-actions qui la composent sont réalisées. Ces croyances sont spécifiées en termes de *recettes*. Une recette est un plan prédéfini permettant de réaliser une action. Il est possible de distinguer : les *recettes complètes* qui sont composées uniquement d'actions définies et les *recettes partielles* composées d'actions encore indéfinies. Une recette partielle devient complète dès lors que toutes ses actions sont définies.

Finalement, une action doit être exécutée par un agent de manière individuelle. Par conséquent, la théorie des plans partagés introduit la notion de plans individuels qui caractérise l'activité d'un agent particulier au sein du plan partagé. Un agent possède un *plan complet individuel* (FIP) pour réaliser une action A_α si (*i*) il connaît une recette pour A_α, (*ii*) il croit qu'il est capable de réaliser chaque action de cette recette et (*iii*) il a l'intention de réaliser toutes les actions la composant. De façon similaire à la définition des plans partagés, au cours du processus de construction, les agents ne possèdent que des plans partiels. On parle alors de *plans partiels individuels* (PIP). Un agent possède un PIP pour réaliser une action A_α si, (*i*) il ne connaît qu'une recette partielle pour réaliser A_α et (*ii*) qu'il ne possède que des plans partiels pour réaliser les actions la composant.

L'élaboration, à partir d'un plan partiel d'un plan complet, implique le choix ou l'extension (de manière incrémentale ou non) d'une recette partielle. Cette élaboration implique également de définir récursivement, pour chaque action de la recette, un sous-plan permettant de la réaliser. Le processus se termine lorsque le plan n'est composé que d'actions élémentaires (*i.e.,* d'actions qu'un agent peut individuellement exécuter). Le plan ainsi obtenu est un plan complet.

Les concepts essentiels[4] à la formalisation de la théorie des plans partagés sont : (1) l'engagement du groupe à réaliser une activité ; (2) l'engagement à réaliser une action au service de l'activité du groupe ; (3) l'existence et l'engagement du groupe à choisir une recette ; (4) l'existence et l'engagement du groupe à allouer des actions à des sous-groupes. Comme la définition l'illustre, la planification collaborative introduit plusieurs problématiques non présentes dans la planification « mono-agent » : (*i*) les agents peuvent posséder des connaissances différentes (*e.g.,* seuls les agents directement impliqués dans le processus de planification et d'exécution de certaines sous-actions ont besoin de connaître les détails de la réalisation de celles-ci) ; (*ii*) l'élaboration d'un plan dans un contexte multi-agent nécessite l'allocation des agents (resp. de sous-groupes d'agents) à des actions individuelles (resp. multi-agents), au sein de la recette ; (*iii*) les agents doivent s'engager non seulement à réaliser leurs propres actions, mais également à soutenir les actions des autres et l'activité globale de l'ensemble de la coalition ; (*iv*) finalement, la planification collaborative nécessite des mécanismes de prise de décisions distribués (*e.g.,* la négociation, les protocoles d'interaction, *etc.*.).

[4] Une description formelle et détaillée est disponible dans (Grosz and Kraus, 1996).

2.3.3 Le processus d'élaboration de plans

Afin d'élaborer collectivement un plan, un groupe d'agents doit mettre en œuvre un certain nombre de mécanismes pour construire, à partir d'un plan partiel, un plan complet : (*i*) compléter et étendre la recette du plan (si elle est partielle) ; (*ii*) sélectionner un agent ou un sous-groupe d'agents pour réaliser une action du plan en construction et (*iii*) établir les engagements individuels et les croyances mutuelles nécessaires à la collaboration. Dans le cadre de la théorie des plans partagés, l'ensemble de ces mécanismes, reposant sur des techniques de prise de décisions distribuées, constitue le processus d'élaboration de plans par lequel un plan partiel évolue en plan complet.

2.4 Les implémentations de la coopération par états mentaux

Cette section présente les principales implémentations fondées sur les modèles de coopérations par états mentaux. Nous abordons l'implémentation de la théorie de l'intention conjointe au travers des projets ARCHON et STEAM et celle des plans partagés par l'intermédiaire des projets STEAM, GIGAGENT et WEBTRADER.

2.4.1 Archon

Jennings (Jennings, 1993; Jennings, 1995) s'est intéressé à l'utilisation de l'engagement conjoint (JPG) dans le domaine de la coordination d'un système industriel, appelé ARCHON (Wittig, 1992; Jennings and Wittig, 1992; Perriolat et al., 1996) dans lequel les engagements sont décrits sous forme de règles. Ce formalisme de description des engagements rend ainsi possible le codage explicite de la coordination dans le mécanisme de raisonnement des agents.

L'architecture d'un agent dans le système ARCHON est présenté à la figure 2.1. Les agents sont composés de trois couches :

1. *Une couche de contrôle* qui modélise les capacités dépendantes du domaine de l'agent. Cette couche permet de dissocier les capacités de l'agent intrinsèques au domaine d'application et les capacités liées à la coopération.
2. *Une couche de coopération* qui définit les règles d'engagements entre agents. Pour illustration, les tableaux 2.1 et 2.2 montrent une partie des règles de réévaluation d'engagement et de sélection d'actions (Jennings, 1995). Les règles du tableau 2.1 expriment les règles dans le cas où le but conjoint a été atteint normalement et les règles du tableau 2.2, les règles mises en œuvre dans le cas d'un échec.
3. *Une couche de communication* qui gère les interactions avec les autres agents du système.

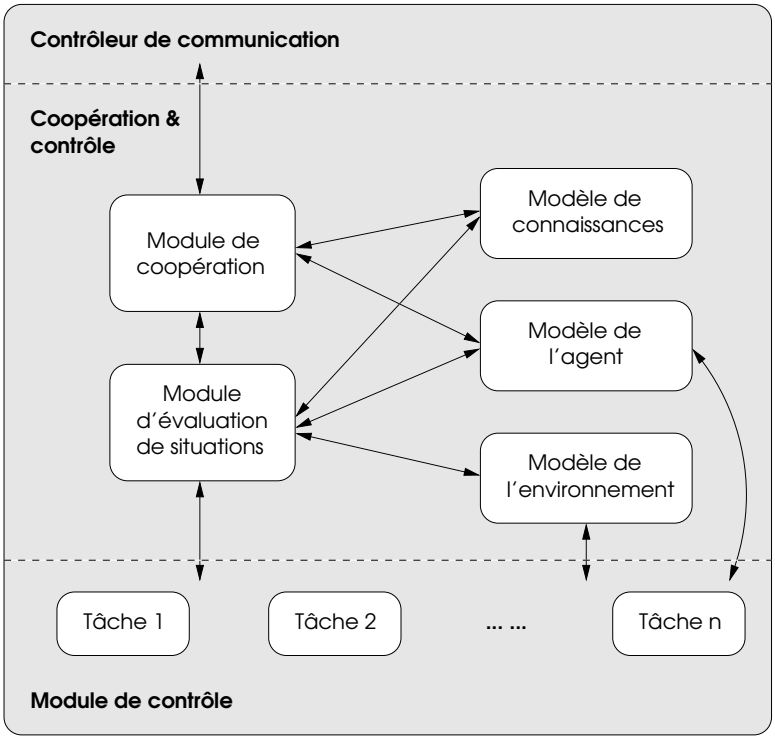


FIG. 2.1: Architecture d'un agent ARCHON.

La mise en œuvre de la coopération repose sur un réseau d'accointances repris du système MACE (Gasser et al., 1988; Gasser et al., 1987) décrivant trois types d'informations :

1. *Un modèles de connaissances* qui contient des informations relatives aux autres agents. Le modèle de connaissances maintient les informations suivantes :
   - *Classe :* les agents sont organisés en groupes appelés classes, identifiés par un nom unique.
   - *Nom :* chaque agent possède un nom unique dans sa classe. L'adresse d'un agent est le couple $\langle classe, nom \rangle$.
   - *Rôles :* un rôle décrit la place que joue un agent dans une classe.
   - *Compétences :* les compétences représentent ce qu'un agent est capable de réaliser.
   - *Buts :* les buts modélisent ce que veut atteindre un agent.
   - *Plans :* les plans modélisent la façon dont un agent veut atteindre ses buts.
2. *Un modèle d'agent* qui contient les informations relatives aux compétences et capacités de l'agent ;
3. *Un modèle de l'environnement* qui contient les informations relatives à l'environnement de l'agent ;

Règles de réévaluation d'engagement		
R1	si	tâche t terminée et t a produit l'action conjointe désirée
	alors	le but conjoint est atteint.
R2	si	réception de l'information i i concerne le but conjoint G et i invalide la croyance nécessaire à G
	alors	se désengager du but conjoint G.
R3	si	suppression de la tâche t1 et t1 est une sous-partie de la recette R et t1 doit être synchronisée avec t2 dans R
	alors	R est violée.
R4	si	la recette courante R est terminée le résultat attendu par R n'est pas obtenu et il existe une autre recette
	alors	R est invalide.

TAB. 2.1: Règles de réévaluation d'engagement dans ARCHON.

Règles de sélection d'actions en cas d'échec		
R1	si	le but conjoint est satisfait
	alors	abandonner toutes les tâches associées et informer le module de coopération.
R2	si	désengager envers le but conjoint
	alors	abandonner toutes les tâches associées et informer le module de coopération.
R3	si	la recette courante R est violée et R ne peut être réordonnancée
	alors	suspendre les tâches associées à R et réinitialiser le timer et la description associés à R et informer le module de coopération.
R4	si	la recette courante R1 et invalide et il existe une recette alternative R2
	alors	abandonner toutes les tâches associées à R1 et informer le module de coopération que R1 est invalide et proposer R2 au module de coopération.

TAB. 2.2: Règles de sélection d'actions en cas d'échec dans ARCHON.

2.4.2 Teamwork

Ce paragraphe présente un modèle général de coopération entre agents pour atteindre un but commun appelé STEAM[5] (Tambe, 1997; Tambe and Zhang, 1998). Ce modèle s'appuie sur la théorie de l'intention conjointe (Levesque et al., 1990) et s'inspire des travaux de la théorie des plans partagées (Grosz and Kraus, 1996). Plus précisément, STEAM s'intéresse à la mise en œuvre de systèmes collaboratifs (Tambe, 1998; Pynadath et al., 1999; Tambe, 1996) en considérant une approche hiérarchique à deux niveaux :

1. *Un niveau organisationnel qui s'appuie sur des rôles* : une équipe d'agents peut avoir une organisation plus ou moins hiérarchisée (*e.g.,* une équipe d'agents peut être composée de plusieurs sous-équipes). Cette organisation hiérarchique s'appuie sur deux types de rôles :
   (a) *Les rôles persistants :* ce type de rôle est affecté à long terme à un agent ou à une sous-équipe de l'organisation ;
   (b) *Les rôles associés à une tâche spécifique :* ce type de rôle est affecté à court terme en fonction de la tâche à exécuter et de la situation.

   L'affectation d'un rôle à un agent ou à une sous-équipe d'agents est réalisée en fonction de leurs capacités au moment de l'exécution. En effet, si les capacités d'un agent ou d'une sous-équipe d'agents décroissent au cours du temps, son rôle doit alors être réaffecté pour limiter la dégradation des performances. Cette affectation peut être effectuée de deux manières : soit un agent se porte volontaire pour assumer le rôle ; soit un autre agent lui demande explicitement de l'assurer.
2. *Un niveau tâches qui s'appuie sur des opérateurs :* STEAM s'appuie sur une représentation explicite des activités collectives sous la forme d'*opérateurs collaboratifs* (un opérateur peut être vu comme un plan réactif et collaboratif). Par exemple, la figure 2.2 montre une hiérarchie d'opérateurs extraite d'un domaine d'application de STEAM (la simulation d'hélicoptères de combat). Cette hiérarchie d'opérateurs s'inspire des hiérarchies de plans réactifs dans les architectures telles que RAPS (Firby, 1987), PRS (Rao et al., 1993) ou SOAR (Newell, 1992). Cependant l'apport de STEAM réside dans la formalisation des activités des agents. STEAM distingue les activités nécessitant plusieurs agents représentées par des opérateurs collaboratifs (*e.g.,* Attaquer (cf. figure 2.2)) et les opérateurs individuels qui expriment qu'une activité peut être réalisée par un seul agent.

   Comme les opérateurs individuels, les opérateurs collaboratifs sont constitués de règles permettant leurs déclenchements ; de règles définissant les effets de l'opérateur et de règles de terminaison. Cependant, la principale différence entre ces deux types d'opérateurs réside dans leur domaine d'application. Un opérateur individuel agit sur les croyances individuelles d'un agent tandis qu'un opérateur collaboratif agit sur les croyances mutuelles d'un groupe d'agents

[5] « *A Shell for Teamwork* ».

collectivement engagé à réaliser la tâche associée à cet opérateur. Par conséquent, chaque agent engagé dans une tâche collective doit garder en mémoire les croyances mutuelles relatives à cette tâche.

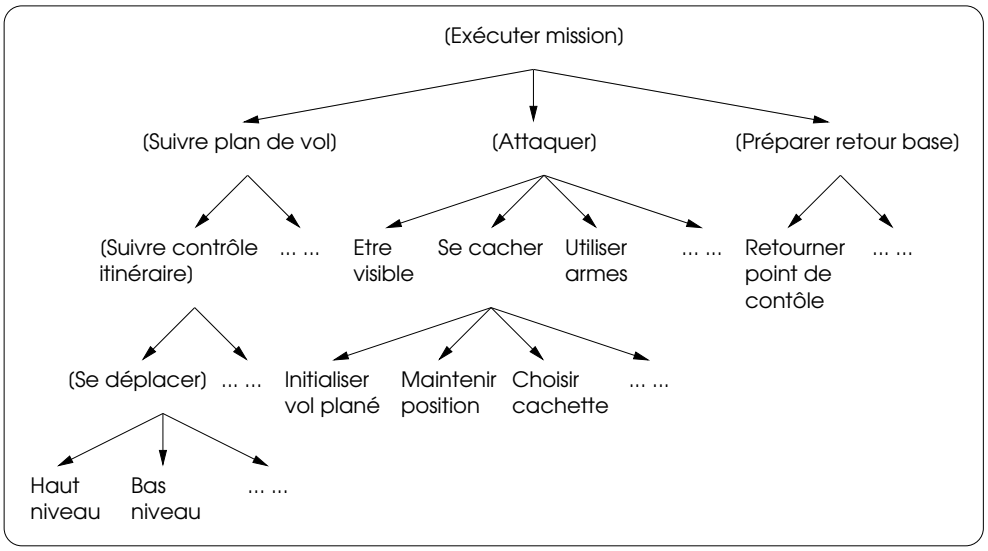


FIG. 2.2: Exemple de hiérarchie d'opérateurs extrait du domaine d'application des hélicoptères de combat. Les opérateurs entre crochets représentent les opérateurs collaboratifs (*i.e.,* nécessitant plusieurs agents).

Finalement, STEAM distingue trois classes d'actions indépendantes du domaine facilitant la communication et la coordination des agents d'une équipe :

1. *Une classe d'actions préservant la cohérence* : cette classe d'actions requiert que les membres de l'équipe communiquent pour s'assurer de la cohérence de l'initiation et de la terminaison des opérateurs collaboratifs. Lorsqu'un agent découvre localement que l'opérateur collaboratif est atteint ou inatteignable, il ne doit pas simplement terminer la tâche associée à l'opérateur mais également en informer les autres membres de l'équipe, pour éviter la perte de ressources et conclure de manière cohérente cette tâche ;
2. *Une classe d'actions de contrôle et de réparation* : cette classe d'actions détecte si une tâche n'est pas réalisable à cause de l'échec d'un agent de l'équipe. Ce type d'actions conduit à une réorganisation, par l'intermédiaire de réassignement de rôles, pour surmonter cet échec.
3. *Une classe d'actions filtrant les communications* : cette classe d'actions permet de limiter les informations transmises aux membres de l'équipe. L'idée principale est de raisonner explicitement sur les coûts et les bénéfices de transmettre des informations pour atteindre des croyances mutuelles.

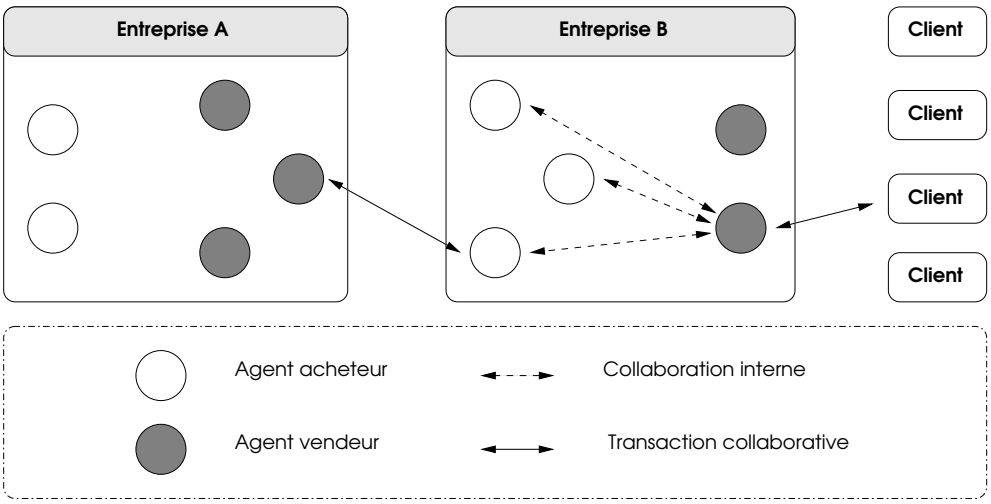


FIG. 2.3: Architecture de WEBTRADER

2.4.3 La théorie des plans partagés et ses applications

La théories des plans partagés a été appliquée notamment dans deux projets (Grosz et al., 1999) : le projet GIGAGENT qui permet la collaboration entre un groupe d'agents hétérogènes et le projet WEBTRADER appliqué au commerce électronique.

Le système GIGAGENT permet la collaboration entre agents humains et informatiques. Le but du système est de fournir une plate-forme pour étudier la collaboration dans un système multi-agent. Ce système trouve son application dans deux domaines : la musique et l'administration de systèmes. Dans la première application, chaque agent humain est secondé par un agent informatique pour l'aider dans ses tâches de composition. Dans la seconde application, l'agent humain est secondé dans sa tâche d'administrateur réseau par des agents responsables des sauvegardes et des installations de logiciels.

Dans WEBTRADER (cf. figure 2.3), il existe plusieurs entreprises composées d'acheteurs et de vendeurs. Les clients peuvent interagir avec les vendeurs pour acheter un bien. Au sein du système, les bénéfices des agents sont calculés en termes d'entrées d'argent pour l'entreprise. Le but des agents d'une même entreprise est de maximiser l'argent gagné. Le projet WEBTRADER propose une approche originale pour aborder le domaine du commerce électronique. Contrairement aux autres approches (Takahashi et al., 1996; Wurman et al., 1998), elle s'appuie sur un mécanisme de collaboration et non de compétition entre les agents.

Conclusion

Ce chapitre a présenté deux théories permettant la formalisation de la coopération au travers des états mentaux des agents. De façon générale, les modèles cognitifs

et les définitions des états mentaux ne font pas l'objet d'un consensus parmi les chercheurs du domaine. Les critiques les plus courantes sont les suivantes :

1. Les modèles sont formalisés sous la forme d'axiomes logiques utilisant des modalités « auto-épistémiques » (*e.g.,* l'agent *croit qu'il croit p*). La combinatoire induite par cette formalisation rend ces modèles difficiles à implémenter dans leur formalisme d'origine.
2. Sur un plan théorique, l'un des inconvénients majeurs de ces approches réside dans le fait que les agents soient omniscients, *i.e.,* l'ensemble des croyances d'un agent contient toutes les formules valides pouvant être déduites. Par conséquent, ces modèles ne prennent pas en compte les capacités d'inférence obligatoirement limitées des agents.
3. Ces modèles se fondent entièrement sur le postulat philosophique de l'intention (Bratman, 1992) et ne reposent sur aucune donnée expérimentale.

Malgré ces inconvénients, la formalisation de la coopération par états mentaux doit être perçue comme un modèle théorique vers lequel devrait converger les modèles effectifs.

Chapitre 3

Argumentation et dialogue

Introduction

L'argumentation est un sujet d'étude très vaste au regard du nombre de théories élaborées depuis Aristote. Chaque théorie est empreinte du cadre que lui impose sa discipline d'origine (*e.g.,* la linguistique, la philosophie, la logique mathématique) et porte un regard spécifique sur le phénomène étudié au point que l'argumentation peut avoir des sens bien différents. Dans le contexte multi-agent, l'argumentation fait référence à l'ensemble des raisonnements par lesquels on déduit les conséquences logiques d'un principe, d'une cause ou d'un fait, en vue de prouver le bien-fondé d'une affirmation, et de convaincre. La capacité d'un agent à argumenter repose essentiellement sur sa capacité à lier les différents faits entre eux, à décider quels arguments soutiennent quels faits, mais également à interagir avec les autres agents.

Dans une première partie (§ 3.1), nous présentons l'approche théorique de l'argumentation. Puis, nous proposons d'introduire les concepts manipulés dans le domaine de l'argumentation (§ 3.2). Finalement, nous présentons les principaux modèles dialectiques (§ 3.3) et une brève description de quelques réalisations effectuées dans ce cadre (§ 3.4).

3.1 Le cadre théorique de l'argumentation

Dans ses *Essais sur l'argumentation*, Plantin (Plantin, 1990) propose une classification méthodique de l'argumentation. La première définition donnée par Plantin de l'argumentation est la suivante :

> *« L'argumentation est l'opération par laquelle un énonciateur cherche à transformer par des moyens linguistiques le système de croyances et de représentations de son interlocuteur. »*

Cette définition exprime de manière pragmatique le concept d'argumentation comme la formulation de propositions et l'intention de l'énonciateur de modifier les

croyances et les représentations du locuteur. (Vernant, 1997) rejoint cette définition en précisant que tout énoncé a pour but de transformer le monde du locuteur. Toutefois, cette définition ne précise nullement les moyens à mettre en œuvre pour parvenir à ces transformations ni celles à effectuer.

La classification de l'argumentation proposée par Plantin distingue l'argumentation comme *fait de langue* ou comme *fait de discours*. Dans le premier cas, l'argumentation est perçue comme un moyen de faire converger le locuteur vers certains enchaînements. Selon cette conception, l'argumentation ne peut être assujetti à une norme. Au contraire, si l'argumentation est perçue comme un fait de discours, elle doit être soutenue par un schéma argumentatif de base. Il s'agit alors d'évaluer la caractéristique du discours selon des normes. On peut alors distinguer d'une part les discours qui poursuivent un but pratique (on parle alors de rhétorique) et d'autre part les discours dont le but est de montrer la validité d'un propos (on parle alors de dialectique). C'est ce dernier type de discours que nous traitons dans ce chapitre.

3.1.1 L'approche de Toulmin

Dans le cadre des dialogues argumentatifs, Toulmin (Toulmin, 1958), l'un des pionniers de l'étude formelle de l'argumentation, formalise la notion d'argument en la décomposant en conclusion (*claims*), justifications (*data*), et règles d'inférence (*warrants*). Certaines conditions exceptionnelles (*rebuttal*) permettent de ne pas appliquer ces règles. Enfin, les règles elles-mêmes doivent être justifiées par des justifications de règles (*backing*).

En d'autres termes, un schéma argumentatif de base est soutenu par une prémisse et une conclusion et par une opération inférentielle, appelée loi de passage (Toulmin et al., 1979). Cette loi constitue le noyau de la cellule argumentative (cf. figure 3.1). Dans ce cadre, l'opération inférentielle est régie par une norme de discours, qui distingue les lois de passage valides des inférences fallacieuses. Il s'agit alors d'évaluer la caractéristique du discours selon des normes externes tant sur l'activité linguistique que langagière afin de déterminer les inférences fallacieuses.

3.1.2 L'approche de Dung

L'une des approches importante, dans le domaine de l'argumentation formelle est issue des travaux de (Dung, 1995; Bondarenko et al., 1997). Ces travaux proposent un modèle argumentatif indépendant de la structure interne des arguments qui s'appuie sur la nature des relations d'attaque qui les lient (cf. figure 3.2). Un argument est une entité abstraite dont le rôle est déterminé par ses relations avec les autres arguments. Aucune hypothèse n'est faite quant à la nature des relations d'attaque entre les arguments.

Définition. 3.1 (Système argumentatif) Un système argumentatif $\mathcal{AF}$ est un couple

$$\mathcal{AF} = \langle \mathcal{A}, \leftarrow \rangle$$

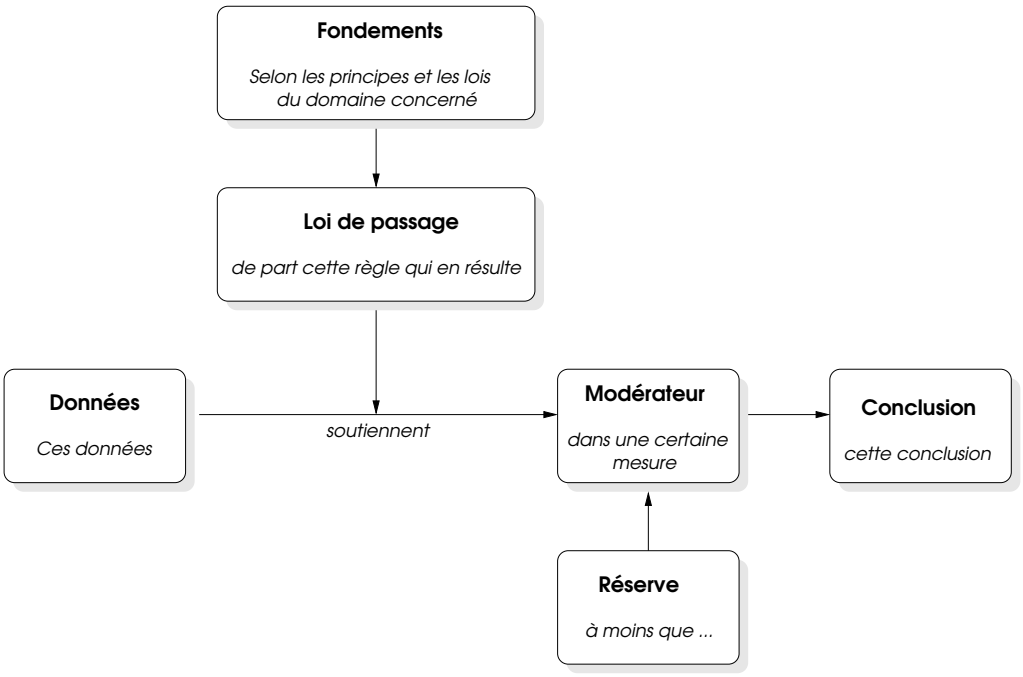


FIG. 3.1: La cellule argumentative de Toulmin.

où $\mathcal{A}$ est un ensemble d'arguments et $\leftarrow$ est une relation entre les paires d'arguments de $\mathcal{A}$. L'expression $a \leftarrow b$ signifie que « a est attaqué par b », « b est un attaquant de a » ou « b est un contre-argument de a ».

Exemple 3.1 Le couple $\mathcal{AF} = \langle \mathcal{A}, \leftarrow \rangle$ avec comme arguments

$$\mathcal{A} = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, p, q\}$$

et $\leftarrow$ comme le montre la figure 3.2, est un exemple abstrait d'un système argumentatif.

Pour un agent rationnel, un argument est acceptable s'il est capable de le défendre contre toute attaque. L'acceptabilité caractérise le principe suivant : un agent peut défendre les arguments qu'il accepte et il accepte tous les arguments qu'il peut défendre. Les arguments *admissibles* par un agent rationnel sont définis de la manière suivante :

Définition. 3.1 (Arguments admissibles)

1. Un argument $a \in \mathcal{A}$ est *acceptable* en fonction d'un ensemble d'arguments $\mathcal{S}$ si et seulement si pour tous les arguments $b \in \mathcal{A}$: si b attaque a, alors b est attaqué.
2. Un ensemble d'arguments exempt de conflit $\mathcal{S}$ (*i.e.,* tel qu'il n'existe pas a, b dans $\mathcal{S}$ avec $a \leftarrow b$) est *admissible* si tous les arguments de $\mathcal{S}$ sont acceptables.

À partir de cette définition, Dung introduit deux sémantiques pour caractériser les agents crédules et les agents septiques (Dung, 1995).

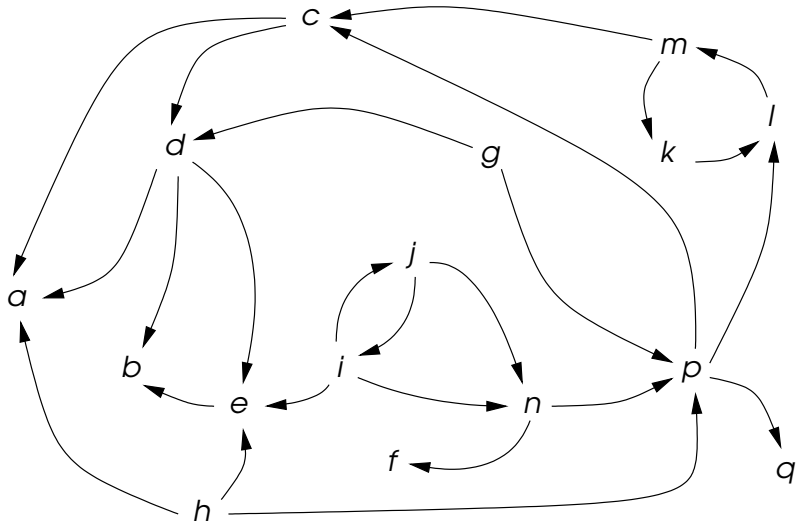


FIG. 3.2: Relations d'attaque dans un système argumentatif.

3.2 Les concepts des dialogues argumentatifs

Nous abordons, dans ce paragraphe, les concepts sur lesquels s'appuient les dialogues argumentatifs. Nous introduirons succinctement la théorie des actes de langage, puis nous présentons les notions de *convention*, d'*engagement* et de *tableau de conversation*.

3.2.1 La théorie des actes de langage

La théorie des actes de langage, issue de la philosophie et de la linguistique, est une approche *pragmatique* du langage. Cette théorie est classiquement attribuée à Austin (Austin, 1962). Dans ses travaux, « *How To Do Things With Words* », il souligne que le langage humain repose sur des classes d'actes de langage[1] constituées non seulement d'affirmations sur des faits, mais également d'affirmations qui cherchent à modifier l'état du monde des interlocuteurs. Austin appelle de tels actes de langage des *performatifs*. Il postule que l'expression d'une proposition par un agent peut se composer de trois types d'actes :

- *Un acte locutoire* est réalisé simplement par l'expression d'un énoncé syntaxiquement acceptable ;
- *Un acte illocutoire* est réalisé avec par l'expression d'un énoncé contenant explicitement ou implicitement un verbe performatif (*e.g.*, « *demander* », « *informer* », « *argumenter* », etc.). À chaque verbe est associée une *force illocutoire* ;
- *Un acte perlocutoire* est un acte qui produit un effet voulu par l'énonciateur sur l'auditeur.

[1] Le terme d'acte de langage est souvent ramené à la seule classe des actes illocutoires.

L'une des étapes du développement de la théorie des actes de langage fut proposée par Searle (Searle, 1969; Searle, 1979). La thèse défendue réside dans le fait que toute conversation est régie par des règles comportementales. Ceci amène Searle à formuler les actes de langage en termes de règles, en dérivant les conditions *nécessaires* et *suffisantes* à leur réalisation. Mais les travaux de Searle se sont surtout attachés à proposer une classification des actes illocutoires selon leur force. La typologie des actes de langage comporte cinq catégories :

- *Un acte assertif* engage le locuteur sur la vérité d'une proposition exprimée (*e.g.,* une affirmation) ;
- *Un acte directif* est une tentative de la part du locuteur d'obtenir quelque chose de l'auditeur (*e.g.,* une demande, un ordre) ;
- *Un acte commissif* engage le locuteur à réaliser une action (*e.g.,* une promesse) ;
- *Un acte expressif* exprime un état psychologique du locuteur (*e.g.,* un remerciement) ;
- *Un acte déclaratif* a pour but de modifier un état du monde (*e.g.,* une déclaration).

Bien que cette typologie fut souvent critiquée, elle a le mérite d'avoir permis l'identification d'au moins trois types syntaxiques d'énoncés communs à la plupart des langages : déclaratifs, impératifs et interrogatifs.

3.2.2 Les conventions et les normes dans le dialogue

La convention évoque, dans le sens commun, un accord passé entre différents membres d'une communauté. Selon (Lewis, 1969), une convention doit être perçue comme une règle existant au sein d'une communauté mais qui ne fait pas forcément l'objet d'un accord explicite. Les règles peuvent être exprimées par exemple de manière impérative, *e.g., « fais x »* ou *« ne fais pas x »*.

Les règles, qui nous intéressent dans le cadre du dialogue entre agents, sont les règles qui régulent et maintiennent la cohérence du dialogue. Une façon simple de mettre en place ce type de règles est de définir un protocole de communication définissant les interactions possibles entre agents. C'est principalement dans le domaine de la recherche sur les langages de communication entre agents artificiels qu'a été développée la notion de protocole de communication. Les modèles KQML (*Knowledge Query and Manipulation Language*) (Finin et al., 1994) et ceux de la FIPA (*Foundation for Intelligent Physical Agents*) (Labrou and Finin, 1997) sont directement inspirés de la théorie des actes de langage. Les protocoles de communication spécifient les séquences autorisées d'actes communicatifs, sans préciser le contenu sémantique des actes. Généralement, ces protocoles sont modélisés au moyen d'automates à états finis. Un exemple célèbre (cf. figure 3.3) est celui proposé par Winograd et Flores (Winograd and Flores, 1987). Ce protocole définit les enchaînements des actes de langage autorisés au cours d'un dialogue argumentatif.

Les protocoles souffrent pourtant de la faiblesse des langages de communication avec lesquels ils sont utilisés. Cela contraint les protocoles à spécifier à la fois la

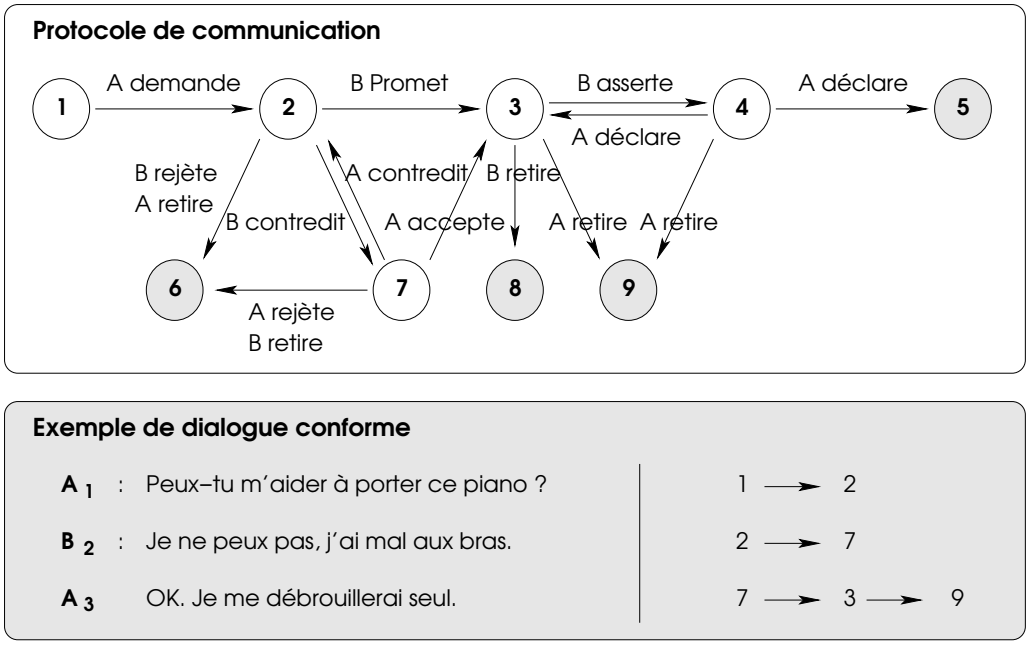


FIG. 3.3: Protocole de communication de Winograd et Flores.

coordination et la sémantique des actes de communication. Ils ont finalement un rôle ambigu alors que dans l'idéal ils ne devraient spécifier que les règles d'enchaînement du dialogue.

3.2.3 Le tableau de conversation et les engagements

La notion de tableau de conversation permet de mettre en avant l'aspect public d'un dialogue en enregistrant l'état d'une conversation. Par conséquent, le tableau de conversation peut être vu comme une base de faits partagés de laquelle il est possible de déduire certaines informations. Par exemple, si lors d'une promenade dans un parc, il se met à pleuvoir, il est raisonnable de déduire de cette situation que tous les personnes présentes croient qu'il pleut. Ce fait appartient maintenant au *fonds commun*. Il est ensuite possible d'en déduire que chaque personne de la communauté a accès à cette information. Dans le cadre du dialogue, les énoncés forment la base de faits partagés à partir de laquelle certaines informations sont supposées communes (ou publiques). Cette hypothèse paraît être suffisamment réaliste (Hamblin, 1971) pour traiter la majorité des situations dialogales, tout en permettant de se concentrer sur les mécanismes de mises à jour du tableau de conversation.

Il nous reste à présent à préciser la nature des faits qui peuvent être stockés dans le tableau de conversation. Une solution naïve viserait à conserver tous les énoncés de chaque participant. Le tableau peut alors être considéré comme un simple historique du dialogue.

(Singh, 2000) propose de raffiner la notion de tableau de conversation en introduisant le concept d'*engagement*. Dans cette approche, le tableau ne représente plus simplement un enregistrement des propositions énoncées mais traduit également les engagements pris par les agents au cours du dialogue. Par exemple, si un agent affirme une propriété du monde, il s'engage à ce que cette propriété soit effectivement vérifiée dans sa base de croyances (engagement propositionnel). Les travaux de (Walton and Krabbe, 1995; Singh, 1999; Piwek, 2000) permettent de dégager les différents types d'engagements que les interlocuteurs peuvent prendre au cours d'un dialogue. Nous donnons ci-dessous les principaux :

Les engagements sociaux. Ce sont les engagements pris vis-à-vis d'autres membres d'une communauté. Dans le cadre du dialogue, ce sont donc des engagements envers l'interlocuteur. Par exemple, un agent peut s'engager envers un groupe d'agents à réaliser telle ou telle tâche.

Les engagements propositionnels ou en actions. Les engagements propositionnels sont généralement considérés comme des engagements dirigés vers le présent, portant sur une propriété du monde, et sont typiquement véhiculés par les assertifs, *e.g.,* « j'affirme que la porte est ouverte ». Les engagements en actions portent généralement sur les actions à réaliser dans le futur. Les actions sur lesquelles s'engagent les agents peuvent être des actions à réaliser dans le cadre du dialogue, *e.g.,* « je m'engage à affirmer que la porte est ouverte », ou plus généralement des actions à exécuter, *e.g.,* « je m'engage à fermer la porte ».

Les engagements conditionnels. Ils permettent d'exprimer des engagements du type *« Si ϕ alors ψ »*. Autrement dit, si la condition ϕ est vérifiée alors je m'engage à réaliser ψ. La condition peut être, par exemple, une propriété du monde qui doit être satisfaite ou un état du dialogue qui doit être atteint.

Les engagements explicites ou implicites. Les engagements explicites sont représentés par des actes de langage. Au contraire, les engagements implicites font partie d'une connaissance partagée par un groupe d'agents. Ce type d'engagement correspond aux *habitudes d'interactions* (Singh, 1999). Par exemple, la non réponse d'un agent à une question peut exprimer son acceptation.

3.3 Les modèles dialectiques

Au travers de ce paragraphe, nous traitons les différentes approches mises en œuvre dans le cadre des dialogues argumentatifs.

3.3.1 La logique dialogique

Les travaux de Lorenzen (Lorenzen, 1987) s'appuient sur les connecteurs classiques de la logique propositionnelle pour formaliser les règles nécessaires à la mise en œuvre de dialogues argumentatifs entre agents artificiels. Ces travaux ont donné naissance à la logique dialogique. Pour Lorenzen, les connecteurs logiques utilisés dans les énoncés expriment les règles du dialogue (cf. tableau 3.1). Par exemple,

c'est le fait qu'il soit nécecessaire de prendre tour à tour la défense de P puis de Q quand on a affirmé $P \wedge Q$, ou encore que pour réfuter $P \wedge Q$, il faille attaquer successivement P puis Q, que la conjonction logique $P \wedge Q$ prend tout son sens. Le sens des connecteurs logiques n'est plus défini en termes de langage, mais en termes de règles de dialogue.

Connecteur	**Assertion**	**Attaque**	**Défense**
Implication	$A \rightarrow B$	A	B
Conjonction	$A \wedge B$	A ou B	A ou B
Disjonction	$A \vee B$	A puis B	A puis B
Négation	$\neg A$	A	—
Quantificateur existentiel	$\exists x A(x)$	quel x ?	$x = a$
Quantificateur universel	$\forall x A(x)$	$\neg A(a)$	$A(a)$

TAB. 3.1: Sémantique des connecteurs logiques de Lorenzen.

Selon Lorenzen, le dialogue argumentatif est une construction interactive d'une preuve logique dans laquelle une proposition est considérée vraie si elle conduit à l'affirmation par les deux participants des mêmes propositions (*i.e.,* l'opposant accepte toutes les propositions du proposant).

3.3.2 Les dialogues critiques de Barth et Krabbe

Les travaux de Barth et Krabbe (Barth and Krabbe, 1982) qui s'appuient sur ceux de Lorenzen propose un modèle plus général pour l'étude des dialogues critiques. Les dialogues critiques débutent par l'assertion d'une thèse (*i.e.,* une proposition énoncée par l'un des deux participants est attaquée par l'autre). Un dialogue critique est modélisé par un quadruplet $\langle \mathcal{C}, \mathcal{T}, \mathcal{O}, \mathcal{P} \rangle$ où $\mathcal{T}$ est la thèse énoncée par le proposant $\mathcal{P}$; $\mathcal{O}$ l'opposant à $\mathcal{T}$ et $\mathcal{C}$ l'ensemble des concessions faites par $\mathcal{O}$. Chaque participant à un dialogue possède un rôle dialogique (*i.e.,* proposant ou opposant) dictant les comportements rationnels admis au sein du dialogue.

Les participants s'appuient sur quatre actes de langage que ce soit pour l'attaque ou la défense :

- L'affirmation : *« J'affirme p »* ;
- La proposition : *« Je soutiens p »* ;
- La mise en doute : *« Est-ce que p ? »* ;
- La défense inconditionnelle : *« Tu as dit toi-même que p »*.

La discussion se termine dès lors que le participant ne peut plus énoncer d'argument ou déroge aux règles dialogales (*e.g.,* en prenant la défense de deux propositions contradictoires p et $\neg p$). Sur cette base, Barth et Krabbe proposent plusieurs règles permettant de structurer le dialogue en s'appuyant sur les états que peut prendre un énoncé pour les agents, *i.e., pour*, *neutre* ou *contre*. Ces règles peuvent être résumées de la manière suivante :

- Le proposant est *pour* toutes les propositions qu'il avance et *neutre* par rapport aux propositions de l'opposant. Il a donc un devoir de défense mais pas un droit inconditionnel d'attaque. La contre-attaque constitue en revanche une défense autorisée.
- L'opposant, de manière asymétrique, est *pour* ce qu'il avance et *contre* ce qu'avance le proposant. Il a le droit d'attaquer toute proposition ne figurant pas dans sa liste de propositions déjà concédées.

3.3.3 Le modèle mathématique du dialogue de Hamblin

Dans son article fondateur paru en 1971, Hamblin (Hamblin, 1971) propose d'étudier plus spécifiquement les arguments *formellement* fallacieux en étudiant la validité d'un argument (*i.e.,* si un argument est acceptable ou non), mais hors de son contexte d'utilisation. En effet, certains arguments sont fallacieux car utilisés de manière inadéquate au cours du dialogue, et non parce qu'ils sont intrinsèquement invalides.

Pour Hamblin, les dialogues argumentatifs sont des dialogues qui reposent sur des normes et des conventions. L'objectif est de contraindre les interactions entre les participants en garantissant la validité du raisonnement produit au cours du dialogue. Le modèle de Hamblin s'appuie sur un ensemble de règles qui régulent les « coups » pouvant être joués par les participants au dialogue. Chaque coup représente un acte de langage. Les règles sont de la forme « si *conditions* alors le coup c est interdit » ou alors « si *conditions* alors le coup c est autorisé ». Les conditions attachées aux règles peuvent faire référence à l'historique du dialogue. Pour conserver une trace des différentes propositions énoncées par les participants, Hamblin s'appuie sur une liste d'engagements notée CS, *« Commitment-Stores »*. La notion de *« Commitment-Stores »* est à rapprocher de celle de tableau de conversation introduite dans le paragraphe 3.2.3. Il permet de garantir la cohérence du dialogue en empêchant, par exemple, un joueur de se contredire ou de répéter une proposition.

Par la suite, (MacKenzie, 1979) a proposé une amélioration du système de Hamblin. Le système permet notamment de stocker les défis dans les CS et a été le système qui a été le plus utilisé hors de son contexte dialectique d'origine. Comme Hamblin, MacKenzie modélise une interaction de type argumentative entre deux opposants qui soutiennent des thèses contradictoires. Cet état de fait est implicite et préexiste à la conversation elle-même.

Le langage utilisé par les joueurs repose sur la logique propositionnelle. Les locutions sont constituées à partir de différentes fonctions communicatives appliquées à ces propositions. La liste des coups pouvant être énoncés par le proposant et l'opposant est donnée par le tableau 3.2.

Type de coup	Exemple	Notation
Affirmation	J'asserte que p	p
Retrait	Je retire que p	—
Question	Est-ce que p ?	—
Défi	Pourquoi p ?	$?p$
Résolution	Retire p ou $\neg p$!	—

TAB. 3.2: Coups du système proposé MacKenzie.

Le système introduit deux types de règles : *les règles de mise à jour*[2] (cf. tableau 3.3) qui précisent comment les listes d'engagements doivent être modifiées en fonction des coups ; et les *règles du dialogue* (cf. tableau 3.4) qui spécifient dans quels contextes les coups sont autorisés ou interdits.

Un coup joué par un joueur peut modifier non seulement sa liste d'engagements mais également la liste d'engagements de l'autre joueur. L'idée ici est qu'un joueur est engagé par défaut sur ce que l'autre lui dit. C'est à lui d'expliciter son désaccord (si désaccord il y a) et ainsi de limiter les acceptations explicites entre les participants.

CR.début : $CS_0(\mathcal{P}) = \emptyset$ et $CS_0(\mathcal{O}) = \emptyset$

CR.assertion(p) : $CS_i(\mathcal{P}) = CS_{i-1}(\mathcal{P}) \cup \{p\}$ et $CS_i(\mathcal{O}) = CS_{i-1}(\mathcal{O}) \cup \{p\}$

CR.défi(p) : $CS_i(\mathcal{P}) = CS_{i-1}(\mathcal{P}) \cup \{?p\} - \{p\}$ et $CS_i(\mathcal{O}) = CS_{i-1}(\mathcal{O})$

CR.assertion(p) comme défense sur q : (après un défi sur q)
$CS_i(\mathcal{P}) = CS_{i-1}(\mathcal{P}) \cup \{p, p \rightarrow q\}$ et $CS_i(\mathcal{O}) = CS_{i-1}(\mathcal{O}) \cup \{p, p \rightarrow q\}$

CR.retrait(p) : $CS_i(\mathcal{P}) = CS_{i-1}(\mathcal{P})$ et $CS_i(\mathcal{O}) = CS_{i-1}(\mathcal{O})$

CR.question(p) : $CS_i(\mathcal{P}) = CS_{i-1}(\mathcal{P})$ et $CS_i(\mathcal{O}) = CS_{i-1}(\mathcal{O})$

CR.résolution(p) : $CS_i(\mathcal{P}) = CS_{i-1}(\mathcal{P})$ et $CS_i(\mathcal{O}) = CS_{i-1}(\mathcal{O})$

TAB. 3.3: Règles de mise à jour des listes d'engagements.

Initialement, les listes d'engagements (CS) des deux joueurs $\mathcal{P}$ et $\mathcal{O}$ sont vides (**CR.début**). Si le joueur $\mathcal{P}$ affirme une proposition p (**CR.assertion(p)**) alors p est ajoutée aux engagements préalablement pris par $\mathcal{P}$ et $\mathcal{O}$ (engagement de facto). Si le joueur $\mathcal{P}$ défie une proposition p sur laquelle $\mathcal{O}$ est engagée (**CR.défi(p)**) alors $\mathcal{P}$ se désengage en supprimant p de sa liste d'engagements. Si le joueur $\mathcal{P}$ affirme une proposition p en réponse à un défi sur une proposition q (**CR.assertion(p) comme défense sur q**) alors $\mathcal{P}$ et $\mathcal{O}$ ajoutent à leur liste d'engagements l'implication $p \rightarrow q$. Finalement, si $\mathcal{P}$ retire (**CR.retrait(p)**), questionne (**CR.question(p)**)

[2]Ces règles sont exprimées selon le formalisme suivant : $CS_i(X)$ spécifie la liste d'engagements du joueur X après le coup 0, et à l'exception de la règle initiale (CR.début), toutes supposent que le dernier coup a été joué par le proposant P (Maudet, 2001).

ou demande la résolution (**CR.résolution(p)**) d'une proposition p alors les listes d'engagements de $\mathcal{P}$ et $\mathcal{O}$ restent inchangées.

DR.tours : chaque joueur joue à tour de rôle une locution autorisée.

DR.question : après une question sur p, l'autre joueur peut jouer (*i*) affirmer p, (*ii*) affirmer $\neg p$ ou (*iii*) défier p.

DR.assertion : ne peuvent être affirmer que les propositions qui ne sont pas déjà présentes dans le CS des deux joueurs.

DR.retrait : une implication dont la conséquence est une conséquence immédiate de son antécédent ne peut être ni retirée, ni défiée.

DR.défi : après un défi sur p, l'autre joueur peut (*i*) retirer p (*ii*) demander la résolution d'une implication dont la conclusion est p et dont l'antécédent est une conjonction de faits sur lesquels le participant qui a émis le défi est engagé, ou (*iii*) affirmer un fait non défié.

DR.résoudre : ne pas demander de résolution sur p sauf si (*i*) p est une conjonction de faits immédiatement incohérentes, ou si (*ii*) p est une implication pour laquelle la conclusion est une conséquence immédiate des prémisses, l'interlocuteur est engagé sur toutes les prémisses, et le coup précédent est un retrait ou un défi de la conclusion.

DR.résolution(p) : après une demande de résolution sur p, l'interlocuteur peut (*i*) retirer l'un des faits de p (ii) retirer l'une des prémisses de p ou (ii) affirmer p.

TAB. 3.4: Règles du dialogue proposées par MacKenzie.

Exemple 3.2 En guise d'illustration, nous reprenons ici l'exemple donné par (Maudet, 2001). Les propositions a, b, p et s représentent respectivement les énoncés suivants : *« la personne est d'accord »*, *« les journaux ont le droit de publier l'information »*, *« l'information est privée »* et *« l'information concerne la santé de la personne X »*, illustrées par le dialogue suivant :

$\mathcal{P}_1$:	Les journaux ne peuvent pas publier cette information !
$\mathcal{O}_1$:	Pourquoi ?
$\mathcal{P}_2$:	Parce qu'elle concerne la vie privée de X et qu'il n'est pas d'accord.
$\mathcal{O}_2$:	Pourquoi cette information concerne-t-elle sa vie privée ?
$\mathcal{P}_3$:	Parce que cela concerne sa santé.

Le tableau 3.5 représente l'évolution des listes d'engagements des deux joueurs au cours du dialogue présenté ci-dessus. Au départ, $\mathcal{P}$ affirme que les journaux ne doivent pas publier une information concernant un personnage X. En affirmant sa thèse, $\neg d$, $\mathcal{P}$ s'engage lui-même et engage son adversaire $\mathcal{O}$ sur ce fait. $\mathcal{O}$ se désengage de cette affirmation en la défiant. Il s'engage du même coup sur le défi et invite $\mathcal{P}$ à

justifier son affirmation. Il interdit ainsi une justification ultérieure reposant sur ce fait (cf. **DR.défi**). $\mathcal{P}$ fournit alors une justification, ce qui engage les deux joueurs sur cette affirmation et sur le fait que celle-ci implique logiquement le fait défié. Le dialogue se poursuit de manière similaire avec le défi d'un des éléments de la justification de $\mathcal{P}$, ce qui entraîne une nouvelle justification de $\mathcal{P}$.

Tour	**Joueur**	**Coups**	$CS(\mathcal{P})$	$CS(\mathcal{O})$
1	$\mathcal{P}$	*affirme*$(\neg d)$	$\neg d$	$\neg d$
2	$\mathcal{O}$	*défi*$(\neg d)$	$\neg d$	$?\neg d$
3	$\mathcal{P}$	*affirme*$(p \wedge \neg a)$	$\neg d, p \wedge \neg a$	$?\neg d, p \wedge \neg a$
			$p \wedge \neg a \rightarrow \neg d$	$p \wedge \neg a \rightarrow \neg d$
4	$\mathcal{O}$	*défi*(p)	$\neg d, p \wedge \neg a$	$?\neg d, ?p, \neg a$
			$p \wedge \neg a \rightarrow \neg d$	$p \wedge \neg a \rightarrow \neg d$
5	$\mathcal{P}$	*affirme*(s)	$\neg d, p \wedge \neg a, s$	$?\neg d, ?p, \neg a, s$
			$p \wedge \neg a \rightarrow \neg d$	$p \wedge \neg a \rightarrow \neg d$
			$s \rightarrow p$	$s \rightarrow p$

TAB. 3.5: Exemple d'évolution des listes d'engagements.

3.3.4 L'approche dialectique de Prakken

Le modèle dialectique de (Prakken, 1995; Prakken, 1997) présente l'avantage de bien décomposer les différentes problématiques abordées par l'argumentation. Son modèle s'appuie sur quatre couches :

1. *La couche logique* qui définit ce que sont les arguments (*i.e.,* le langage utilisé pour exprimer les arguments ainsi que les mécanismes pour les construire) ;
2. *La couche dialectique* qui traite les conflits entre les arguments en introduisant les notions d'*« attaque »*, de *« réfutation »* ou de *« défaite »*. Cette couche définit également les critères d'évaluation permettant de résoudre les conflits entre arguments ;
3. *La couche procédurale* qui régule les échanges entre les participants du dialogue. Elle introduit les règles du dialogue en définissant quand et comment un participant peut attaquer un argument ou en introduire un nouveau. Cette couche est garante de l'application des règles du dialogue entre les participants ;
4. *La couche stratégique ou heuristique* qui introduit la notion de stratégie dans le dialogue. Elle a pour rôle de déterminer l'argument le plus approprié en fonction des règles du dialogue décrites par les couches dialectique et procédurale.

Ces quatre couches sont illustrées par l'exemple suivant :

$\mathcal{P}_1$ illustre la couche procédurale : le proposant affirme une proposition pour débuter le dialogue que l'opposant peut soit accepter, soit rejeter. $\mathcal{O}_1$ rejette la proposition. $\mathcal{P}_2$ doit alors prouver son affirmation en construisant un argument à partir de la couche logique. La même couche intervient et produit le contre-argument

$\mathcal{P}_1$: J'affirme que John est coupable du meurtre.
$\mathcal{O}_1$: Je rejette ton affirmation.
$\mathcal{P}_2$: Les empreintes digitales de John ont été retrouvées sur le couteau près du corps de Bill. Si Bill a été poignardé, les empreintes de la personne sont sur le couteau, donc Bill a été poignardé par John. Si Bill a été poignardé par une personne, cette personne est coupable du meurtre. Donc John est coupable du meurtre de Bill.
$\mathcal{O}_2$: Je te concède les faits, toutefois, je suis en désaccord avec ton raisonnement : un témoin M. X dit que John a retiré le couteau du corps. Ce qui expliquerait le fait que les empreintes de John se trouve sur le couteau.
$\mathcal{P}_3$: Le témoignage de M. X est évidemment irrecevable, puisqu'il est anonyme. Par conséquent, mon affirmation est toujours valide.

$\mathcal{O}_2$ impliquant la couche dialectique pour déterminer la force appropriée du contre-argument. Même remarque pour $\mathcal{P}_3$, toutefois, $\mathcal{P}_3$ illustre la couche heuristique en utilisant le fait qu'une argument dont une prémisse est inconnue ne peut attaquer un fait.

La couche logique. Cette couche se focalise sur la description du langage qui permet de représenter l'information ainsi que les règles de production d'arguments. Les langages utilisés dans le cadre de l'argumentation sont le plus souvent basés sur des formalismes de la logique modale telle que la logique déontique qui permet d'exprimer des systèmes normatifs. Un système normatif modélise un ensemble d'agents interagissant dont le comportement est gouverné par des normes. Les normes prescrivent ce que les agents sont obligés de faire, ce qu'ils sont autorisés à faire et ce qui leur est interdit de faire (Nute, 1997).

La couche dialectique. Cette couche soulève trois principales questions : à quel moment considérer que des arguments sont conflictuels ? Comment les conflits entre arguments peuvent-ils être comparés ? Quels arguments résistent à un conflit ? La notion de conflit est utile pour déterminer si un argument en attaque un autre. La littérature distingue classiquement trois types de conflits entre arguments. Le premier type de conflit se produit lorsque deux arguments parviennent à des conclusions contradictoires, *i.e.,* réfutation de conclusions. Le second type de conflit se produit lorsque les prémisses[3] de deux arguments sont contradictoires. (Prakken and Vreeswijk, 2002) nomme ce conflit l'attaque de prémisses. La troisième et dernière forme de conflit, appelée conflit de règles, introduite par (Pollock, 1995), se produit lorsqu'un argument défie une règle d'inférence d'un autre argument. Ces trois types de conflits peuvent être directs ou indirects. Un argument est indirect s'il attaque une sous-conclusion d'un argument et direct dans le cas contraire.

[3]Nous utiliserons indifféremment les termes de prémisse, antécédent, ou support pour qualifier les conditions de l'application d'une règle.

La notion de *conflit* ou *d'attaque* entre deux arguments sous-tend une forme d'évaluation des arguments. En d'autres termes, la comparaison de deux arguments doit permettre d'évaluer la pertinence d'un argument vis-à-vis d'un autre, et ainsi déterminer si l'un des deux arguments invalide l'autre. Si l'attaque d'un argument a_1 invalide un argument a_2, alors a_2 est défait par a_1. La comparaison des arguments s'effectue en fonction des règles d'inférences mises en œuvre lors de la production de l'argument. Typiquement, un juriste peut invalider le raisonnement d'un collègue en s'appuyant sur des textes de loi de plus haut niveau hiérarchique. Par exemple, une loi constitutionnelle possède une portée plus limitée qu'une loi extraite d'un traité international. Cette hiérarchie dans les règles d'inférence permet de résoudre les conflits entre arguments. Toutefois, la comparaison hiérarchique des arguments peut elle-même être source de conflits. Dans ce cas, la hiérarchie de règles est également sujette à discussion. L'argumentation s'intéresse alors à déterminer si une règle est supérieure à une autre. Ce type d'argumentation doit alors posséder un langage, permettant d'exprimer les priorités entre les différentes règles d'inférences pour qu'elles puissent être considérées dans l'argumentaire.

La notion de *défaite* d'un argument ne peut être considérée comme la simple comparaison de deux arguments. L'état d'un argument dépend, en effet, de toutes les interactions entre les arguments. Ce problème est connu sous le terme de problème de rétablissement (Horty, 2000). Supposons que B invalide A et que B lui-même soit invalidé par un troisième argument C, alors A est toujours valide. $C \rightarrow B \rightarrow A$ est appelée la preuve de A (ou $C \rightarrow B$ exprime la relation d'attaque entre deux arguments et se lit C attaque B).

La notion de *preuve* est importante (Dung, 1995; Bondarenko et al., 1997). Elle permet de catégoriser les types de raisonnement mis en œuvre lors d'un débat. Un débat, dans lequel tous les arguments doivent être prouvés ou justifiés, s'appuie sur un raisonnement sceptique. Dans le cas contraire, on parle de raisonnement crédule. Il est à noter l'existence d'un autre type de raisonnement appelé par objection où tous les arguments attaqués doivent l'être par des arguments justifiés.

La notion d'*exception* est une notion naturelle dans tout système. La grammaire française n'en est pas exempte. Trois techniques différentes peuvent être mises en œuvre pour exprimer cette notion. Deux d'entre elles sont issues des recherches sur les raisonnements non-monotones tandis que la troisième provient plus spécifiquement des travaux sur l'argumentation.

Une première technique consiste à déclarer explicitement l'exception sous forme d'une *« clause d'exception »*, que l'on peut traduire par le sens commun de *« à moins que »*. De telles clauses sont représentées par des opérateurs qui ne peuvent être formalisés par des expressions issues des raisonnements non-monotones ou de la programmation logique. Dans certaines formalisations, l'exception est intrinsèquement associée à la règle. Par exemple, la règle *« si A alors B à moins que C »* s'exprime de la manière suivante (où $\sim$ représente l'opérateur d'exception) :

$$r_1 : A \wedge \sim C \rightarrow B$$

Une représentation plus abstraite et modulable est également possible en explicitant l'exception en tant que règle. Cette formalisation permet d'extraire l'exception de la règle.

$$r_1 : A \wedge \sim Exc(r_1) \rightarrow B$$
$$r_2 : C \rightarrow Exc(r_1)$$

Une variante de cette technique (Routen and Bench-Capon, 1991; Hage, 1996; Prakken and Santor, 1996), exprime la notion d'exception comme une méta-règle sans utiliser d'opérateur d'exception.

$$r_1 : A \rightarrow B$$
$$r_2 : C \rightarrow Exc(r_1)$$

Cette technique porte le nom d'approche par exclusion. Dans le cadre de l'argumentation, la règle r_2 est un contre-argument de l'application de r_1. Cette formalisation des exceptions est à rapprocher de la notion de conflit entre règles (Pollock, 1995).

Finalement, la troisième technique pour exprimer la notion d'exception repose sur une formalisation implicite de l'exception au travers des priorités entre les règles. Comme dans la seconde approche par exclusion, l'exception n'est pas explicitée, cependant, elle s'appuie sur le conflit entre deux règles et leurs préférences.

$$r_1 : A \rightarrow B$$
$$r_2 : C \rightarrow \neg B$$
$$r_1 < r_2$$

où $<$ définit la relation d'ordre entre les règles r_1 et r_2.

La couche procédurale. Cette couche régule les échanges entre les participants du dialogue. Elle définit la procédure par laquelle le proposant et l'opposant débattent d'une proposition. Cette couche revêt un caractère tout particulier dans le cadre des raisonnements juridiques dans lesquels le vice de procédure peut conduire à une annulation des poursuites ou à un non-lieu de l'accusation. Selon (Toulmin, 1958), un argument est valide s'il résiste aux critiques formulées et qu'il s'insère dans la procédure juridique. La procédure garantit ainsi la rationalité du raisonnement au même titre que la couche dialectique. En d'autres termes, le rôle de la couche procédurale est de spécifier les procédures qui doivent garantir la validité d'un raisonnement. Par exemple, l'une des procédures associée au raisonnement scientifique (Rescher, 1977; Brewka, 1994) édicte que toute proposition est considérée vraie tant qu'elle n'a pas été attaquée par un contre-argument valide.

Dans le cadre spécifique des dialogues argumentatifs dans lesquels la notion de persuasion est présente (Hamblin, 1971; McKenzie, 1990), la couche procédurale régule l'utilisation des actes de langage (Austin, 1962; Searle, 1977) de défi, de retrait, d'acception ou d'argument vis-à-vis d'une proposition. Le but du proposant est de faire concéder à l'opposant sa proposition et, *a contrario*, l'objectif de l'opposant est

de faire retirer la proposition du proposant. Le dialogue se termine lorsque l'un des participants au dialogue a atteint son but.

La plupart des systèmes argumentatifs s'appuient sur de tels mécanismes dialogaux (Bench-Capon, 1998; Lodder, 1999; Prakken, 2001a). Contrairement aux modèles théoriques classiques de l'argumentation composés de la couche logique et de la couche procédurale uniquement, l'ajout de la couche dialectique permet de dissocier les arguments liés aux aspects dialogiques et ceux liés à la procédure caractéristique d'un raisonnement. Une proposition peut, par conséquent, être attaquée par deux types d'arguments : des arguments relevant de l'approche traditionnelle (*i.e.,* attaque de conclusions, de prémisses et de règles) au niveau dialectique mais également par des arguments relevant d'un non respect de la procédure (Brewka, 2001; Prakken, 2000; Prakken, 2001b).

La couche heuristique. Cette couche traite les problèmes liés au choix des arguments à utiliser en déterminant notamment sur quelles prémisses s'appuyer pour construire les arguments et l'ordre de leur introduction dans le dialogue. Le rôle de la couche heuristique est de mettre en œuvre des stratégies efficaces afin de persuader les participants du dialogue de la thèse défendue. Par exemple, certains travaux issus de la négociation (Kraus et al., 1998; Parson et al., 1998) s'appuient sur un modèle des participants du dialogue pour évaluer leurs réponses possibles et ainsi être plus à même de les convaincre.

3.3.5 Un modèle de dialogue pour l'argumentation

Amgoud et de Maudet (Amgoud et al., 2000) ont proposé un modèle de dialogue reposant sur l'argumentation. Ces travaux qui ont été notamment repris par (Morge, 2005) dans le cadre d'un système dialectique pour l'aide à la concertation, s'intéresse à l'argumentation en tant que moteur de l'interaction entre agents rationnels. Chaque agent possède une base de croyances Σ pouvant être incohérente et non déductivement close. Σ contient des formules propositionnelles du langage $\mathcal{L}$ composées des symboles $\vdash$ traduisant l'inférence et $\equiv$ exprimant l'équivalence logique.

Définition. 3.2 (Argument)
Un argument est une paire (H, h) où h est une formule de $\mathcal{L}$ et H un sous-ensemble de Σ tel que (i) H est cohérent, (ii) $H \vdash h$ et (iii) H est minimal, par conséquent aucun sous-ensemble de H ne peut satisfaire (i) et (ii). H est appelé le support de l'argument et h sa conclusion.

À partir de la définition d'un argument, Amgoud introduit la notion d'attaque de prémisses[4] :

Définition. 3.3 (Attaque de prémisses)
Soit (H_1, h_1) et (H_2, h_2) deux arguments de l'ensemble des arguments $\mathcal{A}(\Sigma)$. (H_1, h_1) attaque les prémisses de (H_2, h_2) si et seulement si $\exists h \in H_2$ tel que $h \equiv \neg h_1$.

[4] L'attaque de prémisses est connu sous le terme anglais *« undercutting »*.

En d'autres termes, un argument est attaqué si et seulement si, il existe un argument pour la négation d'un élément de son support. Pour traduire différents niveaux de croyances chez les agents, leurs bases sont stratifiées en fonction d'un ensemble de préférences $Pref$. Un niveau de préférence est un ensemble non vide H de Σ, noté $level(H)$.

Définition. 3.4 (Préférence)
Soit (H_1, h_1) et (H_2, h_2) deux arguments de $\mathcal{A}(\Sigma)$. (H_1, h_1) est préféré à (H_2, h_2) selon $Pref$ si et seulement si $level(H_1) \leq level(H_2)$.

Finalement, Amgoud définit le système argumentatif d'un agent :

Définition. 3.5 (Système argumentatif)
Un système argumentatif est un triplet $\langle \mathcal{A}(\Sigma), Undercut, Pref \rangle$ tel que $\mathcal{A}(\Sigma)$ est un ensemble d'arguments construit à partir de Σ, $Undercut \subseteq \mathcal{A}(\Sigma) \times \mathcal{A}(\Sigma)$ et $Pref$ est un préordre[5] sur $\mathcal{A}(\Sigma) \times \mathcal{A}(\Sigma)$.

Au niveau du dialogue, les travaux d'Amgoud se sont inspirés de l'approche de (Pitt and Mamdanie, 1999) qui décomposent un système en deux niveaux : un niveau responsable du raisonnement et un niveau responsable de l'engagement dialectique des agents.

Le niveau du raisonnement. Le rôle de ce niveau est de définir les liens entre les tours de parole (*i.e.,* les actes de langage) et les états mentaux de ces participants. Ce niveau s'articule autour des attitudes que peuvent prendre les agents. Par exemple, les agents peuvent être *crédules*, *i.e.,* accepter une proposition p si p est soutenue par un argument, ou encore *attentifs*, *i.e.,* n'accepter que les propositions qu'ils peuvent eux-mêmes soutenir.

Le niveau dialectique. Les actes de langage du niveau dialectique sont définis en termes d'engagements (Singh, 2000; Singh, 1998). À chaque agent est associé un tableau de conversation qui garde une trace des engagements (*i.e.,* les propos) d'un participant au dialogue. Le tableau de conversation est régi par des règles qui permettent de définir comment les informations sont mises à jour. Amgoud note un tableau de conversation CS_{A_i} où A_i dénote l'agent auquel appartient le tableau. L'union de tous les tableaux de conversation à un instant donné constitue l'état du dialogue. Plus précisément, un tableau de conversation est noté :

$$CS_{A_i} = \langle \mathcal{S}, \mathcal{I}^{\mathcal{R}}, \mathcal{I}^{\mathcal{Q}}, \mathcal{I}^{\mathcal{C}} \rangle$$

avec :
- $\mathcal{S}$ est l'ensemble des affirmations et des promesses réalisées par l'agent A_i ;
- $\mathcal{I}^{\mathcal{R}}$ est l'ensemble des demandes effectuées par l'agent A_i ;
- $\mathcal{I}^{\mathcal{Q}}$ est l'ensemble des questions posées à l'agent A_i ;

[5] Un préordre est une relation binaire réflexive et transitive.

– $\mathcal{I}^{\mathcal{C}}$ est l'ensemble des défis réalisés par l'agent A_i.

La formalisation des actes de langage nécessaires à la mise en œuvre du dialogue argumentatif est définie par le langage minimal $\mathcal{X}$ constitué des actes suivants :

$$\mathcal{X} = \{Question, Challenge, Assert, Accept, Refuse, Request, Promise, Retract, Argue\}$$

Assert permet les échanges d'informations telles que *« il fait beau »* ou *« j'ai l'intention de faire quelque chose »*. *Request* est invoqué lorsque un agent ne peut ou préfère atteindre ses intentions seul. Les demandes diffèrent des affirmations car elles ne peuvent être associées à une valeur de vérité. L'acceptation d'une demande dépend des intentions de l'agent la recevant. Une promesse, *Promise*, est effectuée par un agent lorsqu'il a besoin de demander quelque chose à un autre agent et qu'il a quelque chose à donner en retour. En réponse à une affirmation, une demande ou une promesse, un agent peut accepter, refuser ou poser une question telle que *« est-il vrai que p est satisfait ? »*. Un *Challenge* permet à un agent de demander les raisons d'une affirmation ou bien de demander quelque chose (*e.g., « pourquoi ce journal a-t-il publié ces informations ? »*). La réponse à un défi doit être un argument. Finalement, *Argue*, permet à un agent d'argumenter en faveur d'une proposition et *Retract* lui permet de retirer une affirmation ou une demande précédemment effectuée.

Deux types d'actes de langage sont distingués : les *actes de langage initiateurs* qui permettent à un agent d'obtenir une réponse d'autres agents (*e.g., Question* ou *Challenge*) et les *actes de langage réactifs* qui permettent à un agent de répondre aux actes de langage initiatiques (cf. tableau 3.6).

Actes initiateurs	**Actes réactifs**
Assert, Request	Accept, Refuse, Promise
Question	Assert
Challenge	Argue, Retract

TAB. 3.6: Tableau des actes de langage et de leurs relations.

Pour chaque acte du tableau 3.6, Amgoud introduit une sémantique en s'appuyant sur les deux niveaux précédemment présentés. Considérons en guise d'illustration la description des actes *Question*, *Challenge*, *Assert*. Nous supposons qu'un agent A_i s'adresse à l'agent A_j :

1. **Question(p).** Le fait sur lequel porte la question est ajouté au tableau de conversation de l'agent récepteur. Ceci implique que l'agent récepteur s'engage à répondre à l'agent émetteur.
   Niveau raisonnement : $\emptyset$ (*aucune condition n'est requise).*
   Niveau dialectique : $CS_{A_j} \cdot \mathcal{I}^{\mathcal{Q}} = CS_{A_j} \cdot \mathcal{I}^{\mathcal{Q}} \cup \{p\}$.
2. **Challenge(p).** Un défi s'ajoute à l'ensemble des défis, à la condition que le fait défié soit un engagement de l'agent émetteur.

Niveau raisonnement : $\emptyset$.
Niveau dialectique : si $p \in CS_{A_j}$ alors $CS_{A_j} \cdot \mathcal{I}^{\mathcal{C}} = CS_{A_j} \cdot \mathcal{I}^{\mathcal{C}} \cup \{p\}$.

3. **Assert(p).** Un agent qui affirme une proposition p possède un argument en sa faveur. Par conséquent l'agent s'engage sur p dans le sens où si un autre agent attaque p, il est capable de justifier sa proposition. Une affirmation est ajoutée dans le tableau de conversation et dans le cas où l'affirmation est en réponse à une question, la question est supprimée du tableau.
   Niveau raisonnement : un agent A_i possède un argument en faveur de p.
   Niveau dialectique :
   - $CS_{A_i} = CS_{A_i} \cup \{p\}$.
   - Si $p \in CS_{A_i} \cdot \mathcal{I}^{\mathcal{Q}}$ alors $CS_{A_i} \cdot \mathcal{I}^{\mathcal{Q}} = CS_{A_i} \cdot \mathcal{I}^{\mathcal{Q}} - \{p\}$.

3.4 L'argumentation et ses applications

Ce paragraphe propose un bref aperçu des systèmes développés dans le cadre de l'argumentation au travers de trois réalisations : *pleadings game*, *persuader*, *trains system*.

3.4.1 Pleadings game

Pleadings game (Gordon, 1994; Gordon, 1995) est une tentative de modélisation d'une plaidoirie dans le cadre d'une procédure judiciaire. Durant cette phase, les parties échangent des arguments et des contre-arguments pour déterminer l'issue du procès. Ce système est à la fois implémenté et formalisé autour de la logique dialogique de (Lorenzen, 1987) et d'une théorie de raisonnement par défaut (Geffner, 1992).

La plaidoirie débute lorsque le plaignant énonce sa première affirmation. Par la suite, le dialogue est gouverné par deux règles structurelles. La première édicte qu'à chaque tour de parole les parties doivent répondre par des actes de langage autorisés et appropriés. Un acte de langage est approprié s'il concerne une affirmation dont il n'a pas encore été question au cours du dialogue. La seconde définit sous quelles conditions un acte de langage est autorisé. Par exemple, une affirmation d'une des parties peut être défiée si et seulement si cette affirmation n'a pas été déjà discutée et si elle n'a pas été précédemment énoncée pour se défendre. De plus, une affirmation défiée peut être défendue par un argument aussi longtemps que cette affirmation n'est pas discutée. Les prémisses de cet argument sont cohérentes avec les précédentes affirmations énoncées. Dans le cas contraire, elles doivent être concédées. Si aucune affirmation autorisée ne peut être énoncée, la plaidoirie se termine.

Le résultat d'une plaidoirie est composé d'une liste d'affirmations identifiées au cours du dialogue, sur lesquelles les parties ne sont pas d'accord et d'un gagnant (s'il existe). La notion de gagnant s'exprime relativement à la liste des affirmations conflictuelles du dialogue. Si la liste des affirmations sur lesquelles il existe un désaccord n'est pas vide, alors aucune des deux parties n'est gagnante et la cour doit prendre

la décision. Si toutes les affirmations conflictuelles ont été discutées et que l'enchaînement des arguments impliquant l'affirmation principale du plaidant construit une théorie valide et irréfutable alors, le plaidant remporte la plaidoirie. Dans le cas contraire, c'est le défenseur qui gagne.

Exemple 3.3 Nous illustrons le *pleadings game* avec un exemple extrait du droit allemand, opposant un plaignant $\mathcal{P}$ et un défenseur $\mathcal{O}$ au sujet de la validité ou de la non-validité d'un contrat. Le plaignant qui a formulé une offre au défenseur débute la plaidoirie en affirmant qu'un contrat les lie ($\mathcal{P}_1$). Le défenseur le défie de le prouver ($\mathcal{O}_1$). Le plaignant soutient son argumentaire en précisant que le défenseur a accepté son offre et que, par conséquent, ils sont liés par contrat ($\mathcal{P}_2$). Les propositions échangées sont numérotées (i).

$\mathcal{P}_1$: **Affirme**[(1) Contrat]
$\mathcal{O}_1$: **Réfute**[(1)]
$\mathcal{P}_2$: **Argumente**[(2) Offre, (3) Acceptation, (4) Offre $\wedge$ Acceptation $\Rightarrow$ (1)]

Par la suite ($\mathcal{O}_2$), le défenseur concède qu'il y a bien eu une offre et que si l'offre a été acceptée alors, ils sont liés par un contrat (4). Toutefois, il réfute le fait d'avoir accepté le contrat et se justifie en soulignant que l'acceptation a été envoyée après la date d'expiration de l'offre. Par conséquent, il n'y a pas d'acceptation au sens légal du terme.

$\mathcal{O}_2$: **Concède**[(2),(4)],
Réfute[(3)],
Argumente[(5) Accepter tardivement, (6) Accepter tardivement $\Rightarrow \neg$(3)]

Le plaignant contre-attaque ($\mathcal{P}_3$) avec un argument plus précis en énonçant que même si l'acceptation était tardive, l'acceptation a bien eu lieu et qu'il a immédiatement renvoyé un message confirmant son acceptation.

$\mathcal{P}_3$: **Concède**[(5)],
Réfute[(6)],
Argumente[(5) Accepter tardivement, (7) Acceptation reçue,
(8) Accepter tardivement $\wedge$ Acceptation reçue $\Rightarrow$ (3)]

Finalement, le défenseur concède l'argument (8) (le seul effet de cette concession est de pouvoir énoncer un contre-argument) et sa prémisse (5) et réfute (7). En effet, c'est la seule affirmation qui peut être encore réfutée puisque le défenseur a introduit la prémisse (5) en $\mathcal{O}_2$. Le plaignant réfute alors (7) et ne peut énoncer d'autre argument. La plaidoirie est alors terminée car aucun autre argument ne peut être formulé par l'une ou l'autre des parties.

$\mathcal{O}_3$: **Concède**[(5),(8)],
Réfute[(7)]
$\mathcal{P}_4$: **Réfute**[(7)]

Dans cette exemple, aucune des deux parties ne parvient à gagner la plaidoirie. Ceci est illustré par les résultats de la plaidoirie présentés ci-dessous. Le graphe dialectique d'arguments construit est le suivant :

$\mathcal{P}_1$: [2,3,4] en faveur de l'existence d'un contrat
$\mathcal{O}_1$: [5,6] en faveur d'une non acceptation de l'offre
$\mathcal{P}_2$: [5,7,8] en faveur de l'acceptation de l'offre

Les affirmations de ce graphe qui n'ont pas été concédées sont les suivantes :

(1) Contrat
(3) Acceptation
(6) Acceptation tardive $\Rightarrow \neg$ Acceptation
(7) Acceptation reçue

3.4.2 Persuader

Persuader est le nom d'un système argumentatif basé sur la négociation développé par Sycara (Sycara, 1989b; Sycara, 1989a; Sycara, 1990). *Persuader* s'applique à la négociation dans le monde du travail et implique trois types d'agents représentant respectivement les syndicats, une entreprise et un médiateur. Le système modélise les propositions et les contre-propositions, formulées de manière itérative, afin que les différentes parties puissent parvenir à un consensus. La négociation porte entre autres sur les salaires, les pensions, l'ancienneté des salariés ou encore la sous-traitance.

L'argumentation s'appuie sur les croyances des agents représentées sous la forme de buts et de relation entre les buts. Un exemple des croyances manipulées (extrait de (Sycara, 1989a) p. 130) est donné par la figure 3.4. Ces croyances, modélisées sous forme de graphe acyclique, représentent celles d'une entreprise dont le but de plus haut niveau est de maximiser son profit. Par exemple, une diminution des coûts (-) de production conduira à une augmentation des profits (+) ; une augmentation de la qualité (+) ou une baisse des prix (-) entraînera une augmentation des ventes (+), *etc.*

Dans (Sycara, 1989a), Sycara donne un exemple d'arguments pouvant être échangés avec les syndicats revendiquant une hausse des salaires : *« l'entreprise ne peut pas augmenter les salaires, car une hausse des salaires entraînera une diminution de la masse salariale »*. Pour engendrer cet argument, le système détermine les buts contradictoires avec la proposition et évalue une action de compensation. Dans l'exemple de la figure 3.4, l'action de compensation est la réduction de la masse salariale. À nouveau, l'action de compensation proposée par l'entreprise peut engendrer une contradiction avec les buts des syndicats et conduire à la production d'un autre argument.

Plus généralement, *Persuader* peut engendrer plus d'un argument à partir d'une proposition. Ces arguments sont classés en fonction de leur force. (Sycara, 1989a) ordonne les arguments, du plus faible au plus fort, selon le principe de (Gilkinson et al., 1954) : (1) principe universel ; (2) thème ; (3) autorité ; (4) *statu quo* ; (5) norme ; (6) habitude ; (7) contre-exemple et (8) menace.

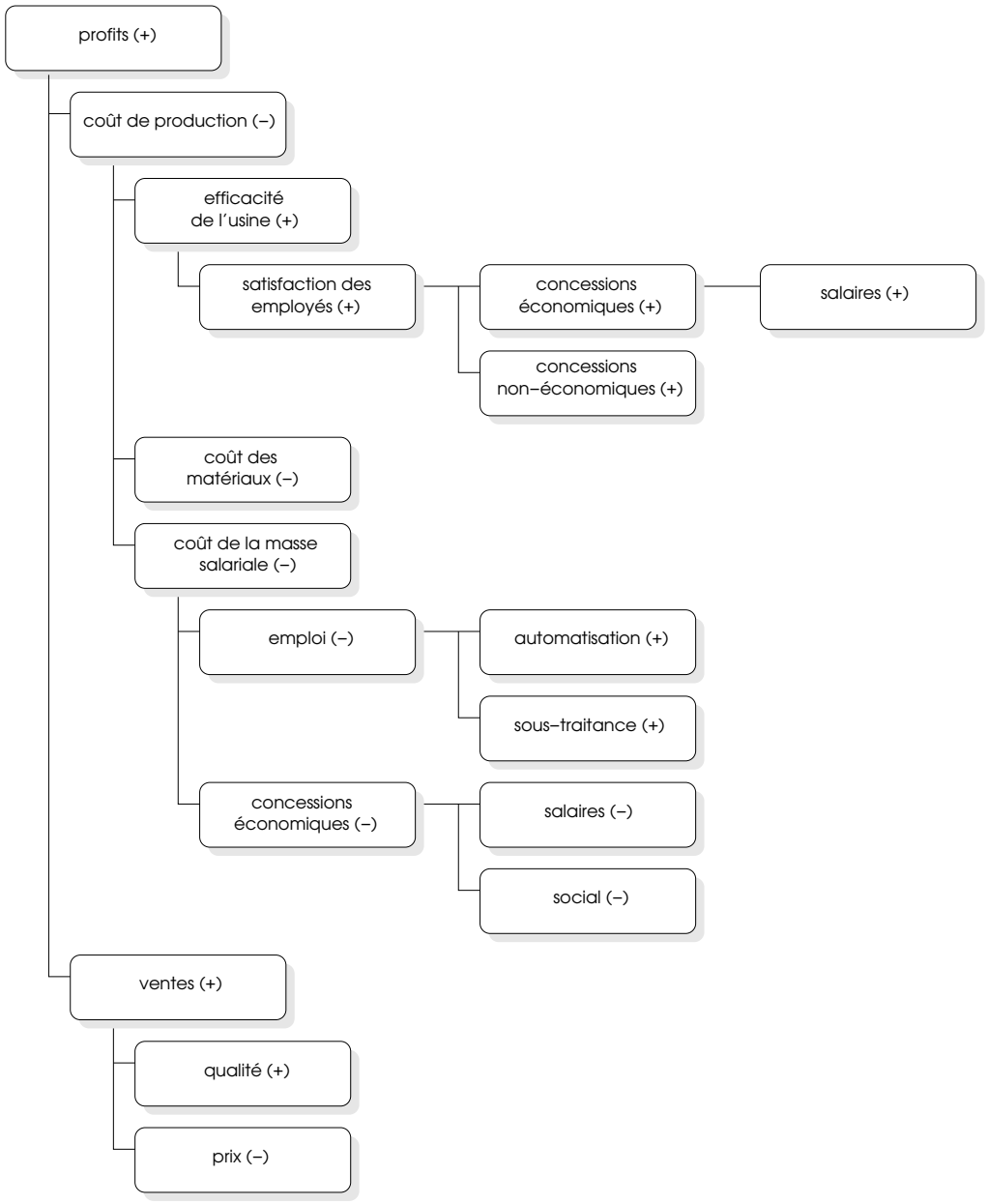


FIG. 3.4: Structure des arguments dans *persuader*.

3.4.3 Système Trains

Les travaux de Ferguson et Allen (Ferguson and Allen, 1994) se sont intéressés à la construction coopérative de plans entre un agent artificiel et un agent humain. Leurs travaux ont été appliqués à l'aide aux usagers du transport ferroviaire. Dans cette approche, les agents sont capables de proposer des plans, d'accepter une suggestion d'un autre agent ou encore de critiquer les plans produits. Pour Ferguson et Allen, la complexité des interactions impose une représentation adaptée de la notion de plan. Ceci les amène à proposer un modèle formel basé sur un système argumentatif[6].

Le raisonnement sur des plans s'exprime en termes de règles portant sur les actions (*i.e.,* les préconditions et leurs effets). Les arguments sont construits pour supporter des faits relatifs à l'exécution des actions et peuvent, comme dans tout système argumentatif, être réfutés. Les arguments ne pouvant plus être réfutés représentent alors des plans valides et exécutables.

La représentation des connaissances s'appuie sur une représentation événementielle et discrète du temps (Allen, 1984), à l'instant t, $n(t)$ représente le prochain pas de temps. Par exemple, le prédicat $Load(e_t)$, qui décrit un événement associé à une action de chargement est vrai si l'événement e_t se produit à l'instant t (*Load* est le type de e_t). Pour éviter la confusion entre la notion d'action et d'événement, Ferguson et Allen introduisent le prédicat $Try(a, t, e_t)$. Ce prédicat est satisfait si l'action a est exécutée à l'instant t entraînant le déclenchement de l'événement e_t.

Classiquement en planification (Finke and Nilsson, 1971), les actions sont représentées par un ensemble de préconditions qui spécifient les propriétés du monde qui doivent être vérifiées pour appliquer une action et un ensemble d'effets qui caractérisent les propriétés modifiées par son exécution. En s'appuyant sur le formalisme de Ferguson et Allen, une action peut se définir par :

$$Holds(preconds(a), t) \wedge Try(a, t, e_t) \supset Holds(effets(a), n(t))$$

En d'autres termes, si les préconditions d'une action a sont valides à un instant t et que nous tentons d'exécuter a, alors les effets de cette action seront vérifiés au prochain pas de temps. Ce type de formalisme pose le problème classique de *quantification*[7]. En effet, toutes les préconditions doivent être vérifiées pour garantir la bonne exécution d'une action.

Pour remédier à ce problème et tisser le lien entre le domaine de la planification et celui de l'argumentation, Ferguson et Allen propose une définition plus faible. Tout d'abord, ils introduisent la notion de règle dans la définition d'une action.

$$Holds(preconds(a), t) \wedge Try(a, t, e_t) \rightarrow Event(e_t)$$

[6]Pour plus de détails sur le système argumentatif utilisé par Ferguson et Allen, se référer à (Loui, 1987) et (Pollock, 1992; Pollock, 1987).

[7]Le problème de quantification (McCarthy, 1977) réside dans la spécification exhaustive des préconditions qui doivent être satisfaites pour qu'une action puissent être déclenchée.

Cette définition exprime que l'exécution d'une action implique l'apparition d'un événement sans dire quels sont les effets d'un tel événement et surtout sans garantir son succès. Une action est une règle d'inférence qui peut être attaquée. Les préconditions sont les prémisses de la règle et l'événement est sa conclusion. L'autre point important de cette définition est le découplage entre les effets d'une action et l'action elle-même, permettant, par exemple, à un agent de planifier une action sans connaître ses effets mais seulement l'événement généré par les actions de son plan. Ce découplage permet de faire le lien entre les connaissances partielles et distribuées des agents. La représentation des effets s'exprime classiquement de la manière suivante :

$$Event(e_t) \supset Holds(effets(e_t), n(t))$$

Ferguson et Allen cherchent également à affaiblir la notion de préconditions. En effet, certaines préconditions ne peuvent être connues que par certains agents ou un agent peut volontairement faire une hypothèse sur une précondition nécessaire au déclenchement d'une action. C'est pourquoi la règle suivante est rajoutée :

$$\{\neg Holds(preconds(a), t) \wedge Try(a, t, e_t)\} \rightarrow \neg Event(e_t)$$

Cette règle garantit simplement que si une condition nécessaire n'est pas satisfaite alors le déclenchement d'une action a ne pourra avoir lieu, mais autorise les agents à appliquer une action dont ils ne connaissent pas la validité des préconditions.

Supposons que ϕ_1 et ϕ_2 soient les préconditions d'une action a. Un plan qui ne prendrait en compte que la précondition ϕ_1 reste raisonnable dans la mesure où la valeur de ϕ_2 n'est pas connue, mais peut être renseignée par un autre agent. Il est alors possible d'établir un ordre partiel entre les différents arguments. Un argument qui ne formule aucune hypothèse quant à ses prémisses est par conséquent plus fort qu'un argument dont certaines sont hypothétiques car inconnues.

Exemple 3.4 Deux agents coopératifs A et B doivent construire un plan pour transporter des marchandises m à un lieu précis. Pour prendre ces marchandises, les agents doivent, dans un premier temps, transporter celles-ci au port P et les embarquer sur un cargo C. Un cargo quitte chaque jour le port entre 4h00 et 6h00. L'heure de départ du cargo est notée T_d, $4 \leq T_d \leq 6$. Si les marchandises voyagent en train jusqu'au port, elles arriveront à 5h00 au port. Si elles voyagent par camion, elles arriveront à 3h00. Toutefois, le transport par camion est trois fois plus cher que par train. Les deux modes de transport sont représentés respectivement par les actions $sendByTruck$ et $sendByTrain$. Ces deux actions ne possèdent pas de préconditions et définissent les règles suivantes où l représente le lieu destination :

$$\{Try(sendByTruck(m, l), t, e_t)\} \rightarrow Transport(e_t, m, l, 3)$$
$$\{Try(sendByTrain(m, l), t, e_t)\} \rightarrow Transport(e_t, m, l, 5)$$

Le prédicat d'événement $Transport$ est défini par :

$$Transport(e_t, m, d, n) \supset At(m, d, t + n)$$

L'action de charger dans le cargo C les marchandises M a pour préconditions que le cargo et les marchandises doivent être présents au port :

$$\{At(m,l,t), At(c,l,t), Try(Load(m,c,l),t,e_t)\} \rightarrow Load(e_t,m,c,l)$$

Le prédicat d'événement *Load* est défini par :

$$Load(e_t,m,c,l) \supset In(m,c,t+1)$$

Finalement, le système possède deux règles spécifiques du domaine :

$$\begin{aligned} T_d > t &\rightarrow At(C,P,t) \\ T_d \leq t &\rightarrow \neg At(C,P,t) \end{aligned}$$

L'agent A propose un plan initial dans lequel les marchandises voyagent par camion et sont chargées dans le cargo. Il ignore la précondition exprimant que le cargo doit être au port à l'heure du chargement.

$$\begin{aligned} Try(sendByTrain(M,P),0,e_0) &\rightarrow Transport(e_0,X,P,5) \\ Transport(e_0,X,P,5) &\rightarrow At(M,P,5) \\ At(M,P,5) \wedge Try(Load(M,C,P),5,e_5) &\rightarrow Load(e_5,M,C,P) \\ Load(e_5,M,C,P) &\rightarrow In(M,C,6) \end{aligned}$$

L'agent B indique que le chargement échouera si le cargo part à 4h00 :

$$\begin{aligned} T_d = 4 &\rightarrow \neg At(M,P,5) \\ \neg At(M,P,5) \wedge Try(Load(M,C,P),5,e_5) &\rightarrow \neg Load(e_5,M,C,P) \end{aligned}$$

L'agent A répond par un argument de même force. Par conséquent, l'argument initial de A est toujours valide et un plan solution a été trouvé :

$$\begin{aligned} T_d = 6 &\rightarrow \neg At(M,P,5) \\ At(M,P,5) \wedge Try(Load(M,C,P),5,e_5) &\rightarrow Load(e_5,M,C,P) \end{aligned}$$

L'argument équivalent dans le cadre du transport par camion aurait été :

$$\begin{aligned} Try(sendByTruk(M,P),0,e_0) &\rightarrow Transport(e_0,X,P,3) \\ Transport(e_0,X,P,3) &\rightarrow At(M,P,3) \\ At(M,P,3) \wedge Try(Load(M,C,P),3,e_3) &\rightarrow Load(e_3,M,C,P) \\ Load(e_3,M,C,P) &\rightarrow In(M,C,3) \end{aligned}$$

Conclusion

Ce chapitre reprend les principaux travaux dans le domaine de l'argumentation et des raisonnements non-monotones sous l'angle du dialogue. Ceci nous a permis de présenter les fondements de l'argumentation ainsi que les mécanismes à mettre en œuvre pour structurer et réguler les interactions entre agents artificiels dans un cadre argumentatif.

Les systèmes dialectiques, et le domaine de l'argumentation de manière plus générale, apparaissent comme des voies prometteuses pour la réalisation de systèmes multi-agents capables de raisonner conjointement. Les travaux récents dans ce domaine montrent que l'argumentation trouve naturellement sa place dans la problématique de la prise de décisions distribuée (notamment dans le cadre de la négociation). L'intérêt de cette étude pour les systèmes multi-agents est double : d'une part, elle donne un cadre formel à la modélisation des interactions entre agents, et d'autre part, elle permet de définir une certaine forme de rationalité à leurs interactions.

Deux constations s'imposent :

Tout d'abord, les systèmes dialectiques se limitent le plus souvent à des interactions impliquant seulement deux participants, *i.e.,* un proposant et un opposant. Ceci constitue une limitation forte dans un cadre multi-agent. En outre, il n'est pas toujours possible de se ramener à des interactions binaires comme l'a souligné (Traum, 2004). Cette limitation provient de la difficulté d'appréhender la notion de tour de parole dans un contexte impliquant plus de deux interlocuteurs.

Finalement, ce chapitre met en évidence que les systèmes dialectiques s'intéressent plus au résultat du processus de délibération, *e.g.,* qui conduit les agents à un consensus sur un état du monde ou à une prise de décisions, qu'à la démarche qui a permis de l'obtenir. Or, cette démarche est toute aussi importante que le résultat, puisqu'elle introduit la notion de preuve. Nous voyons ici se dessiner un lien avec la planification. En effet, un plan est une preuve particulière qui cherche à montrer qu'un but est réalisable. La question est alors la suivante : dans quelle mesure les mécanismes dialectiques peuvent-ils être adaptés à la synthèse de plans dans un contexte multi-agent ?

Chapitre 4

Concepts de la synthèse dialectique de plans

Introduction

La planification est un domaine de l'Intelligence Artificielle qui étudie les mécanismes permettant à un agent autonome de raisonner sur l'action. Ces mécanismes abstraits définissent comment choisir et comment organiser les actions à exécuter, en anticipant leurs conséquences sur le monde pour atteindre un but prédéterminé. En ce sens, la planification constitue un moteur important de la rationalité d'un agent. Mais qu'en est-il lorsque nous cherchons à définir une rationalité collective ? Quels sont les mécanismes qui permettent à des agents de raisonner conjointement sur un but commun à atteindre ? Dans ce manuscrit, nous défendons l'idée forte que la planification distribuée définit une certaine forme de rationalité collective. Dans cette perspective, l'étude des mécanismes nécessaires à la synthèse collaborative de plans constitue une étape importante de sa formalisation.

Dans ce chapitre, nous définissons les concepts de la synthèse dialectique de plans. Nous posons l'hypothèse qu'aucun des agents impliqués dans la construction d'un plan n'est capable de résoudre le but assigné seul et se trouve dans l'obligation d'interagir et de coopérer avec les autres agents afin de l'atteindre. Dans un premier temps, nous présentons les fondements de notre modèle. Puis, nous introduisons les notions préliminaires nécessaires à sa formalisation. Finalement, nous définissons les notions de plans et de plans solutions manipulées dans notre approche.

4.1 Le fondement de l'approche

De manière similaire à la construction d'une preuve mathématique, la production d'un plan peut s'assimiler à la construction automatique d'une preuve permettant de démontrer qu'un but est atteignable. En ce sens, un plan constitue une preuve particulière dans laquelle les règles de déduction sont définies en termes d'actions.

Pour Popper et Lakatos (Lakatos, 1976), il n'existe pas de preuve *correcte* de manière absolue :

> « Nous ne savons rien, nous pouvons seulement conjecturer. »

Autrement dit, toute preuve peut à tout moment être *réfutée* par une expérimentation ou un test. Si le test conduit à une réfutation de la preuve, celle-ci est remise en cause. Il faut alors la *réparer* afin de la rendre plus robuste. La nouvelle version de la preuve peut à son tour être testée et éventuellement réfutée puis réparée. Par conséquent, l'obtention d'une preuve peut se définir comme un processus itératif de propositions – réfutations – réparations. Le respect de cette démarche définit la rationalité scientifique et garantit la validité de la preuve produite.

Pour appliquer une telle approche au domaine de la planification distribuée, il nous faut tout d'abord introduire la notion de *conjecture*. Une conjecture peut se définir comme un *plan sous hypothèses*, *i.e.,* un plan *qui peut être exécuté si certaines conditions sont vérifiées*. Supposons un instant qu'une porte soit fermée. Si l'agent n'est pas capable de l'ouvrir, même s'il peut réaliser l'ensemble des actions nécessaires pour remplir son but derrière celle-ci, il ne proposera aucun plan. En effet, la planification classique (Finke and Nilsson, 1971) requiert que l'ensemble des propriétés du monde nécessaires à l'exécution d'un plan soient satisfaites avant sa réalisation. En revanche, si l'agent peut faire l'hypothèse que la porte est ouverte, il est fort possible qu'un autre agent sera capable de lui venir en aide en proposant un plan pour l'ouvrir. Nous voyons bien ici l'intérêt de considérer la notion de conjecture plutôt que celle de plan. Bien entendu, les hypothèses formulées par les agents doivent être vérifiées pour qu'une conjecture soit considérée comme un plan-solution (*i.e.,* exécutable au sens de la planification classique). C'est alors le rôle des autres agents de proposer leurs compétences pour les satisfaire. Par conséquent, les hypothèses deviennent des sous buts à atteindre par les autres agents. Un agent peut ainsi, même s'il ne possède pas toutes les compétences nécessaires, proposer une conjecture pour résoudre une partie du plan global.

Supposons maintenant que tous les agents soient capables de construire de telles conjectures à partir de leur base de croyances (*i.e.,* l'état du monde connu par un agent) et de leurs compétences (*i.e.,* un ensemble d'actions leur permettant d'agir) et qu'ils possèdent un but commun. Tour à tour, les agents peuvent prendre la parole pour raffiner les hypothèses de la conjecture, réfuter ou encore réparer la conjecture. Lorsque la réparation de la conjecture précédemment réfutée réussit, la conjecture devient plus robuste puisqu'elle résiste dorénavant à cette réfutation mais peut être encore réfutée. Lorsque la réparation échoue ou qu'il n'est plus possible de poursuivre le processus de raffinement, la conjecture doit être abandonnée et une nouvelle doit être explorée. En revanche, si une conjecture n'est pas réfutée, elle est considérée comme un plan acceptable et constitue la solution du système.

En poussant plus loin le rapprochement entre le concept de validité de la preuve de Lakatos et la planification distribuée, il est légitime de s'interroger sur les types de réfutations que l'on peut formuler à l'encontre d'une conjecture et de manière réciproque sur les mécanismes de réparation à mettre en œuvre pour les contrer.

Nous proposons pour répondre à cette question de s'inspirer de la littérature portant sur la coordination et plus spécifiquement sur les travaux traitant des interactions entre plans (Martial, 1992). Les chercheurs de ce domaine se sont attachés à décrire les différents conflits pouvant exister entre plans. Ces conflits peuvent être vus comme autant de réfutations identifiables par les agents. Ces travaux revêtent un intérêt majeur non seulement pour définir les mécanismes permettant d'identifier les conflits et donc les réfutations mais également pour formaliser leurs techniques de résolution.

Ceci nous amène au dernier point important de notre approche : comment organiser les interactions entre les agents pour garantir la validité du plan produit ? Pour répondre à cette question nous nous inspirons des travaux portant sur l'argumentation. Ces travaux fournissent un cadre formel aux interactions entre agents. Nous pensons ici aux travaux sur les dialogues argumentatifs (Prakken, 2001b) et sur les jeux de dialogue (Amgoud et al., 2002; Tang and Parson, 2005). L'intérêt de ces travaux réside dans leurs modèles de l'interaction. En effet, ils apportent une formalisation des mécanismes dialectiques nécessaires à un groupe d'interlocuteurs afin de démontrer la validité d'une proposition. Un dialogue argumentatif se focalise généralement sur deux interlocuteurs, l'opposant et le proposant. Ces deux interlocuteurs constituent deux rôles distincts. Le dialogue est structuré en tours de parole, chaque interlocuteur jouant à tour de rôle. Le dialogue se termine lorsque la thèse soutenue par l'un des deux protagonistes du dialogue est reconnue comme vraie par l'autre.

4.1.1 Un aperçu du modèle

Contrairement à la formalisation « classique » des dialogues argumentatifs (Amgoud et al., 2000), nous proposons un modèle du dialogue dans lequel les interactions entre les agents prennent la forme d'un débat, de manière similaire au type d'interactions que l'on peut trouver sur un forum. Ce modèle semble, en effet, mieux adapté à la mise en œuvre de dialogues dans lesquels le nombre d'agents est important et dont le rôle (*e.g.,* proposant, opposant, *etc.*) évolue rapidement et n'est pas défini a priori (Traum, 2004). Les agents interviennent dans le dialogue en réagissant aux propositions des autres agents, co-contruisant ainsi un plan-solution pour atteindre le but qui leur a été confié.

La vue générale de notre modèle est présentée à la figure 4.1. Les agents communiquent par l'intermédiaire d'actes de dialogue. Le contenu informationnel des actes de dialogue exprime soit des conjectures constituant une partie de la preuve globale construite par les agents, soit des réfutations en dénonçant la validité. Chaque agent possède une structure que nous nommons *tableau de démonstration*, dans laquelle il stocke les conjectures et les réfutations énoncées au cours du dialogue. Les mises à jour, réalisées en fonction des actes reçus ou émis, sont effectuées de manière totalement distribuée.

Le tableau de démonstration d'un agent définit l'état du raisonnement déjà produit par l'ensemble des agents. Chaque énonciation d'un acte, que ce soit pour

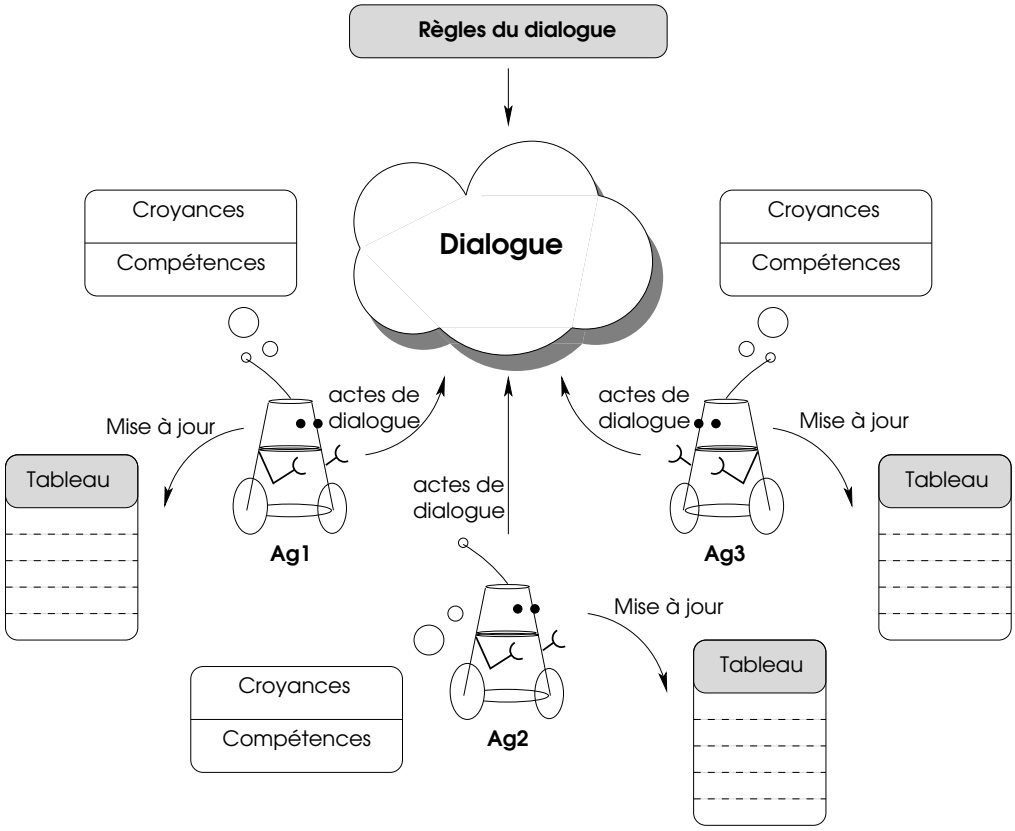


FIG. 4.1: Schématisation du modèle de construction dialectique de plans.

prouver la validité d'une hypothèse ou encore réfuter une conjecture, provoque une opération de mise à jour de celui-ci. Les actes de dialogue autorisés sont définis en fonction de l'état du tableau de démonstration. La mise à jour du tableau de démonstration doit respecter un certain nombre de règles pour garantir la validité du raisonnement global produit. C'est le respect de l'ensemble de ces règles qui fait que le plan est considéré comme un plan-solution valide et accepté par tous.

4.1.2 Un exemple introductif

Exemple 4.1 Considérons un exemple extrait du jeu des « settlers ». Cet exemple est composé de quatre agents : un fermier (farmer), un meunier (miller), un boulanger (baker) ainsi qu'un transporteur (conveyor). Le fermier est capable de produire du blé, le meunier de fabriquer de la farine à partir des sacs de blé vendus par le fermier et le boulanger le pain avec la farine du meunier. Le transporteur est en charge du transport des marchandises. Le but soumis à ces quatre agents est de fabriquer deux pains. Imaginons maintenant le dialogue qu'ils pourraient construire pour atteindre ce but :

baker_1 :	« Je vous propose de faire deux pains. Mais il me manque deux sacs de farine. Il faudrait que ces sacs me soient apportés au village1. »
conveyor_1 :	« Si tu veux, je me propose de transporter dans ma charrette cart1 les deux sacs de farine au village1. Mais je ne connais pas le village où je dois effectuer le chargement de la marchandise. »
miller_1 :	« En ce qui me concerne, je peux te vendre deux sacs de farine mais il faut que quelqu'un puisse te les apporter et que tu me paies la somme de 4 euros. Pour cela, je vais moudre un sac de blé mais il m'en manque encore un. Il faudrait que ce sac me soit apporté au village2. »
baker_2 :	« D'accord, je te paie donc 4 euros pour les deux sacs de farine. »
conveyor_2 :	« D'après ce que je comprends, les deux sacs de farine sont à charger au village2 où habite le meunier. »
farmer_1 :	« Je propose de vendre le sac de blé au meunier. Pour y parvenir je vais planter du blé et le récolter. Sous réserve que le meunier me paye 1 euros pour la marchandise et que cette marchandise puisse être disponible dans le village du meunier. »
miller_2 :	« Je te paie 1 euros pour le blé. ».
conveyor_3 :	« Je crois que vous avez encore besoin de moi pour le transport du sac de blé. Bon, je me propose de transporter dans ma charrette cart1 le sac de blé du village du fermier (village3) au village du meunier (village2). »
farmer_2 :	« Je te remercie mais comment peux-tu utiliser la même charrette pour le transport du blé et de la farine ? »
baker_3 :	« Le fermier a raison. Il faut utiliser une autre charrette pour le transport du blé. »
conveyor_4 :	« Je n'en ai pas d'autre malheureusement. »

miller_3 :	« Je propose que tu transportes le blé dans mon village puis que tu transportes la farine chez le boulanger. Ainsi, il n'est pas nécessaire d'utiliser une autre charrette. »
$\mathsf{conveyor}_5$:	« Ok, ça me va. »

Ce dialogue montre comment les raffinements traduisent les propositions formulées pour atteindre un sous-but (propositions : baker_1, $\mathsf{conveyor}_1$, miller_1, baker_2, farmer_1, $\mathsf{conveyor}_3$), comment les réfutations permettent d'invalider un plan (réfutations : baker_2, farmer_2) et finalement comment les réparations le remanient pour le rendre plus robuste (réparations : miller_2, $\mathsf{conveyor}_2$, miller_4). À la suite de ce dialogue, le plan-solution obtenu est le suivant : le boulanger fabriquera le pain en utilisant la farine que lui aura vendue le meunier. Pour satisfaire la commande du boulanger, le meunier moudra du blé qu'il achètera auprès du fermier. Le fermier plantera la quantité nécessaire de blé pour le récolter et le vendre au meunier. Finalement, le transporteur apportera la farine au boulanger et le blé au meunier.

4.2 Les notions préliminaires

Dans ce paragraphe, nous proposons de définir les notions préliminaires nécessaires à la formalisation du modèle dialectique de synthèse de plans.

4.2.1 Les états de croyance

La représentation des croyances repose sur une notation dérivée de la logique du premier ordre. En effet, la logique du premier ordre se prête bien à la description de propriétés générales sur le monde ; c'est donc elle que nous avons retenue comme base pour construire le langage $\mathcal{L}$ utilisé dans notre approche. Chaque terme de $\mathcal{L}$ est soit une variable soit une constante (nous ôtons les symboles de fonction du langage prédicatif). Les prédicats de la forme $P(t_1, \ldots, t_n)$ avec P un symbole de prédicat n-aire et $t_1, \ldots, t_n$ des termes de $\mathcal{L}$ codent les propriétés du monde manipulées par les agents. Finalement, les mots de $\mathcal{L}$ sont soit des termes soit des formules construites à partir des connecteurs classiques exprimant des propriétés secondaires du monde (Garcia, 1993), *e.g.,* une règle de non ubiquité.

Remarque. 4.1 Nous représenterons les prédicats et les constantes de $\mathcal{L}$ par des chaînes de caractères alphanumériques d'au moins deux caractères, *e.g.,* village1 ou cart2. Les symboles de variables seront toujours décrits par un unique caractère en italique pouvant être indicé, *e.g.,* x ou y_{11}.

Un *état de croyance* est un ensemble de prédicats instanciés de $\mathcal{L}$. Étant donné que notre langage $\mathcal{L}$ ne possède pas de fonctions et que la description des croyances des agents fait intervenir un nombre fini de constantes, l'ensemble des états de

croyance possibles est également fini. On peut alors traduire l'ensemble des formules de $\mathcal{L}$ sous forme d'un ensemble de propositions et utiliser les algorithmes classiques du calcul propositionnel. Nous disons qu'un prédicat p est vérifié dans un état de croyance s si et seulement si p peut être unifié avec un prédicat de s tel que $\sigma(p) \in s$, où σ est la substitution résultat de l'unifcation de p avec le prédicat de s. Dans le cas contraire, nous considérons la propriété du monde représentée par p comme étant inconnue. Par conséquent, il n'est plus possible de poser l'hypothèse du monde clos. Nous dirons alors qu'un prédicat p n'est pas vérifié dans un état s si et seulement si $\sigma(\neg p) \in s$.

Exemple 4.2 Supposons que nous voulions exprimer les croyances d'un agent transporteur. Ses croyances portent sur trois villages (village1, village2, village3), deux charrettes (cart1, cart2) et une caisse de marchandise (goods). L'état de croyance de l'agent est décrit par la figure 4.2.

L'unification[1] de deux prédicats p_1 et p_2 consiste à trouver, quand il existe, un ensemble de couples qui associe à une variable de p_1 un terme tel que p_1 et p_2 soient syntaxiquement égaux. Cet ensemble de couples est appelé une substitution.

Exemple 4.3 Considérons l'unification du prédicat $p = \mathsf{at}(c,v)$ avec l'état de croyance de l'agent transporteur (cf. figure 4.2). Il existe deux substitutions possibles, *i.e.,* deux interprétations possibles pour le prédicat p, définies par les substitutions suivantes :

$$\sigma_1 = \{c = \mathsf{cart1},\ v = \mathsf{village1}\ \}$$
$$\sigma_2 = \{c = \mathsf{cart2},\ v = \mathsf{village2}\ \}$$

Intuitivement, une substitution caractérise un ensemble de contraintes d'instanciation définissant pour chaque variable de p une valeur. Ainsi dans l'exemple ci-dessus σ_1 exprime les deux contraintes d'instanciation $c = \mathsf{cart1}$ et $v = \mathsf{village1}$ tandis que σ_2 exprime $c = \mathsf{cart2}$ et $v = \mathsf{village2}$. En substituant chaque occurrence des variables de p par leurs valeurs associées contenues dans σ_1 et σ_2 on obtient deux prédicats complètement instanciés qui appartiennent à l'état de croyance s.

4.2.2 Les opérateurs et les actions

En s'appuyant sur le modèle d'actions STRIPS (Finke and Nilsson, 1971), nous définissons une action par un opérateur de transformation. Le principe de la modélisation des actions par des opérateurs de transformation consiste à spécifier les propriétés du monde nécessaires à son application, *i.e.,* les préconditions de l'action, ainsi que les propriétés du monde engendrées par son exécution, *i.e.,* les effets de l'action.

[1] Nous renvoyons ici le lecteur sur les travaux fondateurs de (Robinson, 1971) dans le cadre général et de (Paterson and Wegman, 1976) dans le cadre de l'unification linéaire.

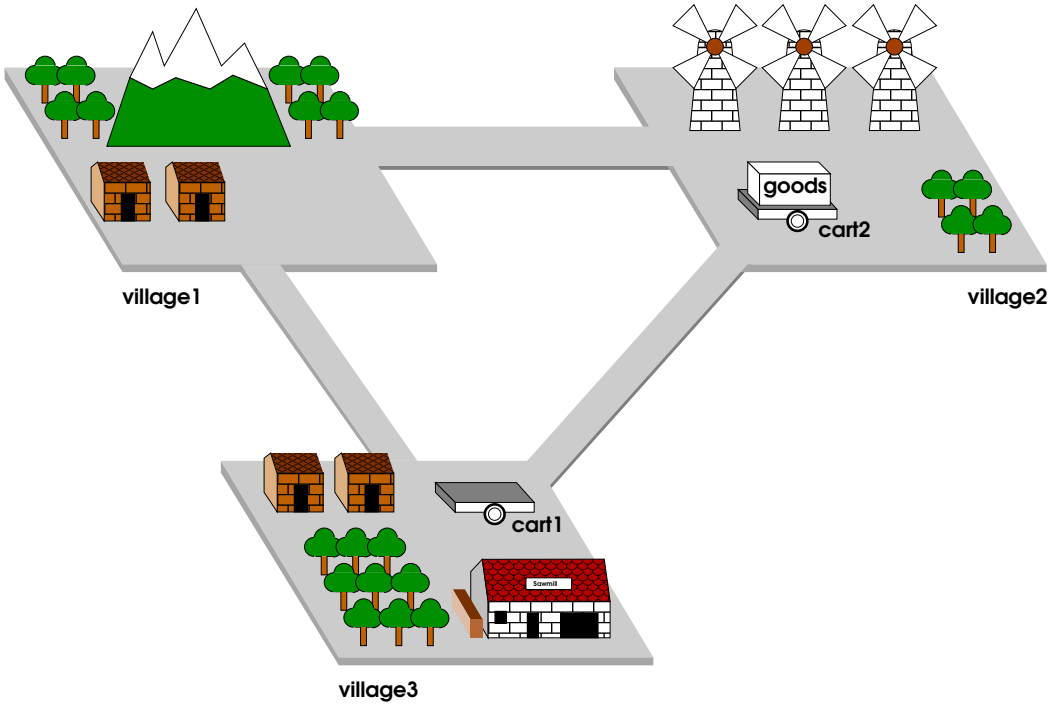


FIG. 4.2: L'état de croyance de l'agent transporteur est décrit par l'état :

$$s = \left\{ \begin{array}{l} \mathsf{connected}(\mathsf{village1}, \mathsf{village2}), \\ \mathsf{connected}(\mathsf{village2}, \mathsf{village1}), \\ \mathsf{connected}(\mathsf{village1}, \mathsf{village3}), \\ \mathsf{connected}(\mathsf{village3}, \mathsf{village1}), \\ \mathsf{connected}(\mathsf{village2}, \mathsf{village3}), \\ \mathsf{connected}(\mathsf{village3}, \mathsf{village2}), \\ \mathsf{at}(\mathsf{cart2}, \mathsf{village2}), \\ \mathsf{at}(\mathsf{cart1}, \mathsf{village3}), \\ \mathsf{in}(\mathsf{goods}, \mathsf{cart2}) \end{array} \right\}$$

Les préconditions d'un opérateur représentent le domaine de définition d'une action, *i.e.,* l'ensemble des états de croyance dans lesquels l'action peut être appliquée. Ce calcul peut être réalisé en unifiant les préconditions de l'opérateur avec l'état de croyance de l'agent. Classiquement, on s'arrange pour que le langage $\mathcal{L}$ contienne un petit nombre de prédicats permettant de définir simplement l'ensemble des domaines de définition des actions.

Les effets d'un opérateur spécifient les propriétés du monde modifiées par l'exécution d'une action. D'un point de vue formel, si une action est définie par un opérateur qui transforme un état de croyance s_i en un état s_{i+1}, les effets d'une action sont perceptibles au travers des différences qu'il existe entres les deux ensembles de prédicats s_i et s_{i+1}. La représentation des effets d'un opérateur de transformation peut donc se ramener au problème suivant : quelles propriétés doit-on ôter à s_i et quelles propriétés doit on ajouter à s_i pour définir s_{i+1} ?

La définition préliminaire[2] d'un opérateur de transformation est la suivante :

Définition. 4.1 (Opérateur)
Un *opérateur* peut être défini par le quadruplet

$$o = (name(o), precond(o), add(o), del(o))$$

- name(o), le nom de l'opérateur, est défini par une expression de la forme $n(x_1, \ldots, x_k)$ où n est un *symbole d'opérateur* et $x_1, \ldots, x_k$ représentent les paramètres de l'opérateur.
- *precond*(o) représentent les préconditions de l'opérateur o, *i.e.,* les propriétés du monde nécessaires à son exécution.
- *add*(o) et *del*(o) définissent deux ensembles de propriétés décrivant respectivement les faits à ajouter et les faits à supprimer de l'état du monde après l'exécution de o.

Le problème de persistance, *i.e.,* la détermination des propriétés du monde qui restent inchangées après l'exécution d'une action, est résolue simplement en conservant dans s_{i+1} toutes les propriétés non concernées par *add*(o) et *del*(o). Le problème de ramification, *i.e.,* la détermination des propriétés du monde qui sont modifiées par l'exécution d'une action, est résolue en explicitant entièrement dans *add*(o) et *del*(o) toutes les propriétés qui sont modifiées par l'opérateur o. Finalement, le problème de quantification, *i.e.,* la détermination des préconditions d'une action, est résolue en spécifiant dans *precond*(o) l'ensemble des conditions nécessaires à son exécution.

Exemple 4.4 Nous donnons ici les opérateurs associés à l'agent transporteur :

;; L'agent a déplace une charrette c d'un village v1 à un village v2
!move(*a*,*c*,*v1*,*v2*)
precond: connected(*v1*,*v2*), at(*c*,*v1*)
add: at(*c*,*v2*), ¬at(*c*,*v1*)
del: at(*c*,*v1*)

[2]Le concept d'opérateur de transformation sera enrichi dans le chapitre 6.

;; L'agent a charge une marchandise g dans une charrette c dans le village v
!load(a,c,v,g)
precond: at(c,v), is-available(g,v)
add: in(c,g), ¬at(c,v), ¬is-available(g,v)
del: at(c,v), is-available(g,v)

;; L'agent a décharge une marchandise g d'une charrette c dans le village v
!unload(a,c,v,g)
precond: at(c,v), in(c,g)
add: is-available(g,v), ¬in(c,g)
del: in(c,g)

Remarque. 4.2 L'absence de l'hypothèse du monde clos dans notre approche implique que la négation des propriétés contenues dans $del(o)$ soit spécifiée dans $add(o)$. Dans la suite, nous supposerons que $add(o)$ contient implicitement les propriétés niées de $del(o)$. Ceci nous permettra de ne pas surcharger l'écriture des opérateurs.

Considérons maintenant les caractéristiques des opérateurs de transformation mis en œuvre dans notre approche. Leur première caractéristique est leur déterminisme. Si une action est applicable dans un état s_i alors l'exécution de l'action transformera toujours s_i en un unique état s_{i+1}. Les opérateurs non déterministes peuvent se découper en deux grandes familles : les opérateurs conditionnels et les opérateurs probabilistes. Les premiers impliquent la prise en compte non plus d'un seul état résultant, mais de plusieurs états définissant autant d'alternatives possibles à l'exécution d'une action (Peot and Smith, 1992). La seconde famille d'opérateurs repose sur une représentation probabiliste de l'action (Kaelbling et al., 1998). À chaque action est associée une valeur qui caractérise la probabilité que l'action produise les effets escomptés par son exécution.

La seconde caractéristique importante des opérateurs choisis réside dans leur représentation implicite du temps. Les actions n'ont pas de durée et définissent des changements d'états instantanés. La représentation explicite du temps, au sein des opérateurs, ne se justifie que lorsqu'il est nécessaire de planifier en prenant en compte des références temporelles absolues. Dans la majorité des cas, un ensemble de contraintes d'ordre suffisent à spécifier des relations temporelles entre les actions (Allen, 1984). Dans un contexte multi-agent, la prise en compte de ces relations est intéressante. En effet, ces relations permettent d'exprimer la concurrence entre les différentes activités des agents.

Exemple 4.5 Supposons que l'état des croyances de l'agent transporteur décrit à la figure 4.2 soit le suivant :

$$s_0 = \left\{ \begin{array}{l} \mathsf{at(cart2, village2)}, \\ \mathsf{at(cart1, village1)}, \\ \mathsf{in(goods, cart2)}, \\ \mathsf{connected(village2, village1)}, \\ \mathsf{connected(village1, village2)} \end{array} \right\}$$

et que l'on veuille appliquer l'opérateur !move

!move(*c*,*v1*,*v2*)
precond: connected(*v1*,*v2*), at(*c*,*v1*)
add: at(*c*,*v2*)
del: at(*c*,*v1*)

L'unification des préconditions de l'opérateur !move avec l'état de croyance s_0 définit deux actions applicables à partir de s_0. La première action peut être décrite par :

;; Déplace la charrette cart1 dans le village village2
!move(cart1,village1,village2)
precond: connected(village1,village2), at(cart1,village1)
add: at(cart1,village2)
del: at(cart1,village1)

L'état résultant de l'application de move est alors :

$$s_1 = \left\{ \begin{array}{l} \mathsf{at}(\mathsf{cart2}, \mathsf{village2}), \\ \mathsf{at}(\mathsf{cart1}, \mathsf{village2}), \\ \neg\mathsf{at}(\mathsf{cart1}, \mathsf{village1}), \\ \mathsf{in}(\mathsf{goods}, \mathsf{cart2}), \\ \mathsf{connected}(\mathsf{village2}, \mathsf{village1}), \\ \mathsf{connected}(\mathsf{village1}, \mathsf{village2}) \end{array} \right\}$$

La seconde action applicable peut être décrite par :

;; Déplace la charrette cart2 dans le village village1
!move(cart2,village2,village1)
precond: connected(village2,village1), at(cart2,village2)
add: at(cart2,village1)
del: at(cart2,village2)

L'état résultant de l'application de move est alors :

$$s'_1 = \left\{ \begin{array}{l} \mathsf{at}(\mathsf{cart2}, \mathsf{village1}), \\ \mathsf{at}(\mathsf{cart1}, \mathsf{village1}), \\ \neg\mathsf{at}(\mathsf{cart2}, \mathsf{village2}), \\ \mathsf{in}(\mathsf{goods}, \mathsf{cart2}), \\ \mathsf{connected}(\mathsf{village2}, \mathsf{village1}), \\ \mathsf{connected}(\mathsf{village1}, \mathsf{village2}) \end{array} \right\}$$

4.2.3 Les notions d'agent, de domaine et de problème

Jusqu'à présent, nous avons introduit le langage sur lequel repose la description des croyances des agents ainsi que les opérateurs nécessaires à sa manipulation. Dans ce paragraphe, nous donnons notre définition des notions d'agent, de domaine et de problème habituellement utilisées en planification.

Définition. 4.2 (Agent)
Un *agent* est un tuple

$$ag = (name(ag), operators(ag), beliefs(ag))$$

- $name(ag)$ est le nom de l'agent ;
- $operators(ag)$ est un ensemble d'opérateurs, *i.e.,* les compétences de l'agent ;
- $beliefs(ag)$ définit un ensemble de propriétés du monde connues par l'agent. Nous supposons que l'état initial de croyance $beliefs(ag)$ définit un ensemble cohérent de propriétés.

Exemple 4.6 L'agent fermier f peut réaliser les actions décrites par les cinq opérateurs suivants :

1. !sow(f,wheat,q) : planter une quantité q de blé.
2. !harvest(f,wheat,q) : récolter une quantité de blé q.
3. !sell(f,ag,wheat,q) : vendre une quantité de blé q à un autre agent ag.
4. !pay(f,ag,g,p) : payer une marchandise g à un autre agent ag pour un prix déterminé p.
5. !settle(f,v) : s'installer dans un village v.

L'ensemble des propriétés décrivant l'état initial de ses croyances est le suivant :

$$beliefs(\text{farmer}) = \left\{ \begin{array}{l} \text{agent-name(famer)}, \\ \text{agent-village(village3)}, \\ \text{sowed-wheat(farmer, 0)}, \\ \text{has-goods(farmer, wheat, 0)}, \\ \text{has-cash(farmer, 2)}, \\ \text{price(farmer, wheat, 3)}, \\ \text{has-tool(farmer, scythe)}, \\ \text{at(cart1, village3)} \end{array} \right\}$$

En planification, un domaine définit classiquement l'ensemble des opérateurs qui peuvent s'appliquer sur le monde. Dans le cadre de notre approche, ces opérateurs sont contenus dans la description des agents eux-mêmes. Par conséquent, un domaine de planification multi-agent peut se définir simplement comme un ensemble d'agents.

Définition. 4.3 (Domaine de planification)
Un *domaine de planification* $\mathcal{D}$ est un ensemble d'agents de la forme $(name(ag), operators(ag), beliefs(ag))$ tel que

- $name(ag)$ identifie de manière unique l'agent ag dans le domaine $\mathcal{D}$;
- l'union des états initiaux des croyances représente un ensemble cohérent de propriétés.

Par conséquent, nous ne nous intéressons pas au problème de la gestion de croyances contradictoires. Un certain nombre de travaux apportent une réponse partielle à cette problématique en utilisant notamment des bases de croyances stratifiées, *e.g.,* (Gregoire, 1999; Amgoud and Cayrol, 2002), la mise en œuvre de systèmes d'inférences vérifiant la cohérence des croyances, *e.g.,* (Stanojevic et al., 1994; Huhns and Bridgeland, 1991) ou encore l'exécution d'actions spécifiques permettant de vérifier certaines propriétés du monde, *e.g.,* (Petrick and Bacchus, 2004).

La gestion de la cohérence des croyances est fondamentale dans les systèmes multi-agents. En revanche, en planification, l'état initial est toujours supposé cohérent car il est difficile de produire des plans corrects à partir d'un état incohérent. En effet, supposons qu'un agent doive atteindre un but et que sa réalisation nécessite l'ouverture d'une porte. Si les croyances des agents spécifient que la porte est ouverte et fermée, quelle propriété doit-on considérer comme vraie ? Une solution pourrait être de produire deux plans, un pour chaque sous état initial cohérent, mais encore faut-il que les agents soient capables de les construire. D'une manière générale, le calcul des sous états cohérents nécessite l'introduction de règles d'inférence permettant de déduire les contradictions entre les propriétés du monde, complexifiant ainsi de façon importante le processus de planification.

Exemple 4.7 Voici un domaine de planification permettant à un groupe d'agents de produire du pain. Un agent fermier fait pousser du blé. Lorsque le blé est arrivé à maturité, il est fauché puis transporté par un transporteur auprès du meunier qui le moud pour obtenir de la farine. Les sacs de farine peuvent alors être déposés chez le boulanger pour la fabrication du pain.

Dans cet exemple, le domaine $\mathcal{D}$ est composé de quatre agents. Par soucis de concision, nous ne donnons ici que leur nom et leur état initial de croyance.

$$\mathcal{D} = \{\mathsf{famer}, \mathsf{miller}, \mathsf{baker}, \mathsf{conveyor}\}$$

$$\mathit{beliefs}(\mathsf{farmer}) = \left\{ \begin{array}{l} \mathsf{agent\text{-}name}(\mathsf{famer}), \\ \mathsf{agent\text{-}village}(\mathsf{village3}), \\ \mathsf{sowed\text{-}wheat}(\mathsf{farmer}, 0), \\ \mathsf{has\text{-}goods}(\mathsf{farmer}, \mathsf{wheat}, 0), \\ \mathsf{has\text{-}cash}(\mathsf{farmer}, 2), \\ \mathsf{price}(\mathsf{farmer}, \mathsf{wheat}, 3), \\ \mathsf{has\text{-}tool}(\mathsf{farmer}, \mathsf{scythe}), \\ \mathsf{at}(\mathsf{cart1}, \mathsf{village3}) \end{array} \right\}$$

$$\mathit{beliefs}(\mathsf{miller}) = \left\{ \begin{array}{l} \mathsf{agent\text{-}name}(\mathsf{miller}), \\ \mathsf{agent\text{-}village}(\mathsf{village2}), \\ \mathsf{has\text{-}goods}(\mathsf{miller}, \mathsf{flour}, 0), \\ \mathsf{has\text{-}goods}(\mathsf{miller}, \mathsf{wheat}, 1), \\ \mathsf{has\text{-}cash}(\mathsf{miller}, 4), \\ \mathsf{price}(\mathsf{miller}, \mathsf{flour}, 1) \end{array} \right\}$$

$$beliefs(\mathsf{baker}) = \left\{ \begin{array}{l} \mathsf{agent\text{-}name(baker)}, \\ \mathsf{agent\text{-}village(village1)}, \\ \mathsf{has\text{-}goods(baker, flour, 0)}, \\ \mathsf{has\text{-}goods(baker, bread, 0)}, \\ \mathsf{has\text{-}cash(baker, 6)}, \\ \mathsf{price(baker, bread, 1)} \end{array} \right\}$$

$$beliefs(\mathsf{conveyor}) = \left\{ \begin{array}{l} \mathsf{agent\text{-}name(conveyor)}, \\ \mathsf{connected(village1, village2)}, \\ \mathsf{connected(village2, village1)}, \\ \mathsf{connected(village2, village3)}, \\ \mathsf{connected(village3, village2)}, \\ \mathsf{at(cart1, village3)} \end{array} \right\}$$

Étant donné un domaine, il nous reste maintenant à décrire un problème de planification. Un problème doit spécifier les états initiaux des croyances des agents, les opérateurs qu'ils peuvent appliquer ainsi que le but qu'ils doivent réaliser. Dans le cadre de la représentation logique des états de croyance, il apparaît naturel de représenter le but sous la forme d'un ensemble de prédicats instanciés décrivant les propriétés du monde qui doivent être vérifiées par les agents.

Définition. 4.4 (Problème de planification)
Un *problème de planification* pour un domaine $\mathcal{D}$ est un triplet

$$\mathcal{P} = (s_0, \mathcal{O}, g)$$

- s_0 et $\mathcal{O}$ représentent l'union des croyances et des opérateurs des agents de $\mathcal{D}$.
- g définit un ensemble cohérent de prédicats instanciés, *i.e.,* les propriétés du monde devant être atteintes.

Exemple 4.8 Un but possible pouvant être soumis aux agents de l'exemple 4.7 peut être le suivant :

$$g \quad = \quad \{\mathsf{has\text{-}goods(baker, bread, 2)}, \mathsf{sow\text{-}wheat(famer, 10)}\}$$

Ce but s'adresse à quatre agents : un fermier, un meunier, un boulanger ainsi qu'un transporteur. Ces quatre agents doivent être capables d'atteindre deux buts : le boulanger doit posséder deux pains et le fermier doit avoir planté dix parcelles de blé.

Remarque. 4.3 Il est possible que les buts décrivant un problème de planification soient incompatibles. Dans ce cas, il est nécessaire de résoudre les conflits entre les buts en appliquant notamment des mécanismes de négociation ou d'argumentation (Karunatillake et al., 2005; Tessier et al., 2000).

4.2.4 Les conjectures et les plans

Classiquement, un *plan* est un ensemble d'actions contenues dans une structure particulière exprimant des relations entre les actions. Dans le cas d'une séquence, la relation entre les actions est une relation d'ordre total. Le choix d'une telle structure semble trop restrictif pour s'appliquer dans un contexte multi-agent. En effet, elle ne permet pas de définir simplement la notion de conjectures (*i.e.,* de plans hypothétiques) mise en œuvre dans notre approche, ni de décrire des actions concurrentes. Ceci nous amène à retenir pour notre approche la notion de plan partiel utilisée par les algorithmes de planification dans un espace de plans (Ghallab et al., 2004).

Pour illuster tous les aspects d'une conjecture reprenons l'exemple 4.8 comme fil conducteur. Nous supposons qu'il existe une conjecture initiale constituée d'une seule action proposée par le boulanger qui permet d'atteindre le but has-goods(baker,bread,2) :

;; Le boulanger baker fabrique une quantité 2 de pain.
!make(baker,bread,2)
precond: has-goods(baker,flour,2)*, has-goods(baker,bread,0), is-available(flour,village1)*
add: has-goods(baker,flour,0), has-goods(baker,bread,2),
del: has-goods(baker,flour,2), has-goods(baker,bread,0), is-available(flour,village1)

Regardons comment la conjecture doit être raffinée par ajouts successifs d'actions[3] et de quelle manière s'effectue sa mise à jour. Cela nous permettra d'introduire de manière informelle la notion d'hypothèse ainsi que les quatre constituants d'une conjecture : un ensemble d'actions, un ensemble de contraintes d'ordre, un ensemble de contraintes d'instanciation et un ensemble de liens causaux.

Les actions. Pour l'instant rien ne garantit au sein de la conjecture que la farine nécessaire à la fabrication des pains soit disponible dans le village où habite le boulanger. Par conséquent, la propriété is-available(flour,village1), requise par les préconditions de l'action !make, est une hypothèse formulée par la conjecture initiale[4]. Pour vérifier cette hypothèse, le transporteur propose de raffiner cette conjecture en ajoutant la séquence suivante de trois actions :

;; Le transporteur conveyor charge la marchandise flour dans la charrette cart1 dans un village v
!load(conveyor,cart1,*v*,flour)
precond: at(cart1,*v*)*, is-available(flour,*v*)*
add: in(cart1,flour)
del: at(cart1,*v*), is-available(flour,*v*)

[3] Par abus de langage, nous utiliserons dans la suite de ce manuscrit le terme générique d'action pour caractériser à la fois une action en tant qu'instance d'un opérateur de transformation et l'opérateur lui-même.

[4] Les hypothèses sont identifiées par un astérisque.

;; Le transporteur conveyor déplace la charrette cart1 d'un village v au village village1
!move(conveyor,cart1,*v*,village1)
precond: connected(*v*,village1)*, at(cart1,*v*)*
add: at(cart1,*village1*)
del: at(cart1,*v*)

;; Le transporteur conveyor décharge la marchandise flour de la charrette cart1 dans le village village1
!unload(conveyor,cart1,village1,flour)
precond: at(cart1,village1), in(cart1,flour)
add: is-available(flour,village1)
del: in(village1,flour)

De la même manière, rien ne garantit que la farine, bien que disponible dans le village du boulanger, lui appartienne. La précondition de l'action !make, has-goods(baker,flour,2), est également une hypothèse formulée par la conjecture. Pour vérifier cette hypothèse, le meunier propose d'ajouter la séquence de deux actions suivantes :

;; Le meunier miller moud une quantité 1 de blé.
!grind(miller,wheat,1)
precond: has-goods(miller,wheat,1)*, has-goods(miller,flour,1),
is-available(wheat,village2)*
add: has-goods(miller,wheat,0), has-goods(miller,flour,2),
del: has-goods(miller,wheat,1), has-goods(miller,flour,1),
is-available(wheat,village2)

;; Le meunier miller vend au boulanger baker la quantité 2 de farine.
!sell(miller,baker,flour,2)
precond: has-goods(miller,flour,2), has-cash(miller,4),
receive-cash(miller,baker,flour,2)*
add: has-goods(miller,flour,0), has-cash(miller,6),
is-available(flour,village2)
del: has-goods(miller,flour,2), has-cash(miller,4)

Les contraintes d'ordre. L'action proposée par le transporteur (!unload), et celle proposée par le meunier (!sell), doivent être exécutées avant l'action !make pour satisfaire les hypothèses formulées par la conjecture initiale. En effet, rien n'indique pour l'instant l'ordre dans lequel ces actions doivent être exécutées. Par conséquent, il est nécessaire d'ajouter une contrainte d'ordre précisant que !unload et !sell doivent être réalisées avant !make.

En revanche, est-ce que la séquence d'actions !load, !move et !unload doit être exécutée avant ou après la séquence !grind et !sell ? Les deux options sont possibles. Dans l'état actuel de la conjecture rien n'oblige à trancher pour l'une ou l'autre des

solutions. Nous appliquons ici le principe de *moindre engagement*. L'ajout d'une contrainte n'a lieu que si elle est strictement nécessaire. Dans ce cas, les seules contraintes d'ordre nécessaires sont de réaliser !unload avant !make et !sell avant !make. Si aucune autre contrainte d'ordre n'est ajoutée à la conjecture au cours du processus de planification, alors les actions proposées par le transporteur et le meunier pourront être exécutées de manière concurrente.

Les liens causaux. Pour le moment, nous savons ajouter des actions et des contraintes d'ordre à une conjecture. Mais est-ce suffisant ? À cause de la représentation non explicite de la notion d'état courant (car distribué sur l'ensemble des agents), les contraintes d'ordre ne suffisent pas à garantir, par exemple, que la farine restera disponible dans le village village1 jusqu'à ce que l'action !make soit réalisée. En effet, au cours du processus de planification, les agents peuvent trouver d'autres raisons de déplacer la farine dans un autre village et oublier la raison qui les a fait la rendre disponible dans le village du boulanger. Par conséquent, il est nécessaire de coder explicitement au sein de la conjecture les raisons qui ont fait que les actions ont été ajoutées. Ainsi, dans notre exemple, il est nécessaire de spécifier que l'action !unload du transporteur a été ajoutée pour vérifier la précondition is-available(flour,village1) de l'action !make.

La relation entre les actions !make et !unload portant sur la propriété is-available(flour,village1) est appelée un *lien causal*. L'action !make est appelée le consommateur et l'action !unload le producteur. Un lien causal exprime qu'une propriété du monde nécessaire à l'exécution d'une action est satisfaite par les effets d'une autre action. En l'absence de lien causal, la précondition de l'action n'est pas vérifiée et sera considérée comme une hypothèse formulée par la conjecture. Les hypothèses doivent être assimilées à des *sous-buts* devant être réalisés par les autres agents.

Notons qu'une action qui supporte une hypothèse doit toujours être réalisée avant l'action qui la formule. Par conséquent, un lien causal est toujours associé à une relation d'ordre, mais il possible d'avoir une contrainte d'ordre sans lien causal. Toutefois, d'autres actions peuvent être intercalées entre les deux actions liées par un lien causal. Un lien causal n'est donc pas garant de l'absence de conflit entre deux actions.

Les contraintes d'instanciation. Il est nécessaire de préciser les contraintes d'instanciation relatives aux variables manipulées par les opérateurs de transformation décrivant les actions. En effet, chaque opérateur, comme présenté dans le paragraphe 4.2.2 décrit un ensemble d'actions. L'unification des préconditions d'un opérateur avec l'état de croyance d'un agent peut définir plusieurs actions applicables à partir d'un même état. Il faut donc garantir, par exemple, que le nouvel opérateur !unload concerne bien la même marchandise flour ainsi que le même lieu village1 utilisés par l'opérateur !make. Certaines variables peuvent ne pas être instanciées. C'est le cas de la variable v de l'opérateur !load proposée par le transporteur. Cette variable traduit le fait que le transporteur ne connaît pas, pour l'instant, le lieu exact où la

marchandise devra être chargée. Nous parlons alors d'action partiellement instanciée.

Pour résumer, nous avons ajouté à la conjecture des actions, des contraintes d'ordre, des liens causaux ainsi que des contraintes d'instanciation. Ces éléments constituent les éléments nécessaires à la formalisation de la notion de conjecture utilisée dans notre approche.

Définition. 4.5 (Conjecture)
Une *conjecture* est un tuple

$$\chi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$$

- $\mathcal{A} = \{a_0, \ldots, a_n\}$ est un ensemble d'actions ;
- $\prec$ est un ensemble de contraintes d'ordre sur les actions $\mathcal{A}$ de la forme $a_i \prec a_j$, *i.e.,* a_i précède a_j ;
- $\mathcal{I}$ est un ensemble de contraintes d'instanciation portant sur les variables des actions $\mathcal{A}$ de la forme $x = y$, $x \neq y$, ou $x = \mathsf{cst}$ tel que $\mathsf{cst} \in D_x$ et D_x est le domaine de x ;
- $\mathcal{C}$ est un ensemble de liens causaux de la forme $a_i \xrightarrow{p} a_j$ tels que a_i et a_j sont deux actions de $\mathcal{A}$, la contrainte d'ordre $a_i \prec a_j$ existe dans $\prec$, la propriété p est un effet de a_i et une précondition de a_j et finalement les contraintes d'instanciation qui lient les variables de a_i et de a_j portant sur la propriété p sont contenues dans $\mathcal{I}$.

Les hypothèses d'une conjecture sont représentées par les préconditions des actions qui ne sont pas supportées par un lien causal.

Définition. 4.6 (Hypothèse)
Soit une conjecture $\chi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$. Une *hypothèse* formulée par χ est définie comme une précondition p d'une action $a_j \in \mathcal{A}$ telle que pour toutes actions $a_i \in \mathcal{A}$, le lien causal $a_i \xrightarrow{p} a_j \notin \mathcal{C}$. Nous notons respectivement $assump(\chi)$ et $assump(a_j)$ l'ensemble des hypothèses formulées par χ et par a_j.

De plus, l'ordonnancement partiel des actions implique qu'une conjecture définit un ensemble de séquences d'actions totalement ordonnées respectant $\prec$.

Définition. 4.7 (Linéarisation)
Soit une conjecture $\chi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$.

1. On appelle *linéarisation* de χ toute conjecture $\lambda = (\mathcal{A}, <, \mathcal{I}, \mathcal{C})$, où $<$ est un ordre total sur $\mathcal{A}$ compatible avec $\prec$, qui définit une séquence de $n+1$ états $\langle s_0, \ldots, s_i, \ldots s_n \rangle$ avec

$$s_i \;=\; (((s_{i-1} \cup assump(a_{i-1})) - del(a_{i-1})) \cup add(a_{i-1}) \quad \text{pour} \quad 0 \leq i \leq n.$$

2. On appelle *complétion* de χ l'ensemble des linéarisations de χ, noté $completion(\chi)$.

Nous dirons que l'ensemble des contraintes d'ordre $\prec$ d'une conjecture χ est *cohérent* si *completion*(χ) est non vide. Cela signifie qu'il existe au moins une linéarisation possible de χ. Pour tester la cohérence des contraintes d'ordre $\prec$ d'une conjecture χ, il faut vérifier que $\prec$ n'exprime aucun cycle de dépendance entre les actions de χ. La figure 4.3 montre une conjecture possédant un tel cycle. La vérification de cette propriété s'effectue en calculant la fermeture transitive de la relation d'ordre définie par $\prec$. Le calcul de la fermeture transitive permet de déterminer pour chaque couple d'actions a_i et a_j s'il existe une relation d'ordre[5].

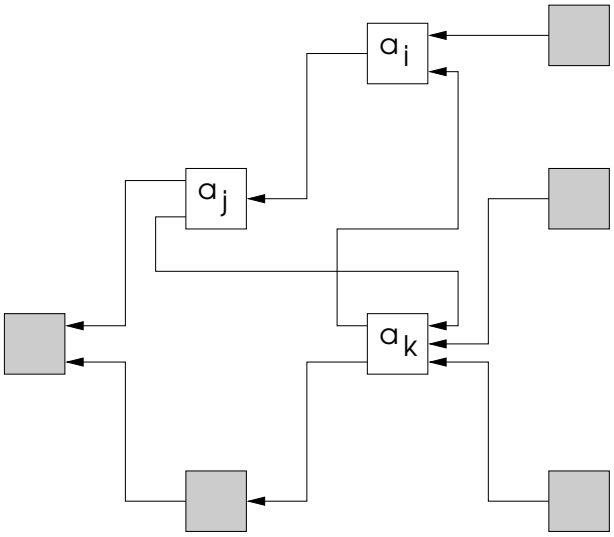


FIG. 4.3: Exemple de cycle de dépendance entre les trois actions a_i, a_j et a_k.

Finalement, l'absence de la notion d'état oblige à représenter les buts par une action particulière. Étant donné que les préconditions d'une action définissent les hypothèses potentielles pouvant être formulées par une action, les buts g sont représentés par une action fictive a_∞ qui ne possède pas d'effets. De manière similaire, la représentation de l'état initial nécessite l'introduction d'une action fictive a_0. Cette action ne possède pas de précondition mais des effets qui représentent l'état initial. Notons que l'état initial global n'est pas accessible directement puisqu'il est réparti sur l'ensemble des agents. Par conséquent, cet état est construit au cours du processus de synthèse de plans par ajout d'effets à a_0.

Une représentation graphique de la conjecture de l'exemple 4.8 est donnée à la figure 4.4. Les rectangles représentent les actions. Les préconditions de l'action sont indiquées au-dessus de l'action tandis que les effets sont indiqués en dessous. Les flèches pleines représentent les relations d'ordre entre les actions et les flèches en pointillés les liens causaux. Les hypothèses sont définies par les préconditions ne portant pas de liens causaux. Les contraintes d'instanciation sont implicitement définies par les paramètres des actions. Afin de ne pas surcharger la représentation, les

[5]Une relation d'ordre sur un ensemble E est une relation binaire dans E, à la fois réflexive, antisymétrique et transitive. Cette relation d'ordre est totale si deux éléments quelconques de E sont comparables sinon elle est partielle.

propriétés statiques du domaine, *i.e.,* agent-village(v), agent-name(n) et price(a,g,p) ne sont pas indiquées.

4.2.5 Les plans solutions et les réfutations

Classiquement, un *plan-solution* se définit comme un chemin dans un espace d'états. Le passage d'un état à l'autre s'effectue par l'application d'une action, *i.e.,* un opérateur complètement instancié, respectant la définition 4.7. Par conséquent, un plan-solution pour un problème de planification $\mathcal{P} = (s_0, \mathcal{O}, g)$ est une séquence d'actions décrivant un chemin d'un état initial s_0, représentant l'union des croyances des agents, à un état final à s_n tel que $g \subseteq s_n$. Or, dans notre approche, nous devons tenir compte du fait qu'une conjecture peut contenir des hypothèses et définit un ensemble de linéarisations. Par conséquent, toutes les linéarisations d'une conjecture doivent décrire un chemin de l'état s_0 à s_n pour que la conjecture soit un plan-solution.

En outre, il est clair que si une conjecture χ ne définit pas un ensemble de contraintes d'ordre $\prec$ cohérent, alors χ ne peut être un plan-solution. Ceci fournira un moyen d'éliminer des voies de recherche inutiles, en interdisant aux agents d'introduire des cycles de dépendances au sein des conjectures.

Rappelons que les contraintes d'instanciation utilisées dans notre modèle sont de trois types : les contraintes unaires de la forme $x = \mathsf{cst}$, $\mathsf{cst} \in D_x$ et les contraintes binaires de la forme $x = y$ et $x \neq y$. Il faut donc également garantir qu'aucune des contraintes d'instanciation de $\mathcal{I}$ n'exprime de contradiction, par exemple :

$$\mathcal{I} = \{\ x = \mathsf{c1}, x = y, y = \mathsf{c2}, z \neq x, z = \mathsf{c1}\ \}.$$

En conclusion, nous donnons la définition d'un plan-solution :

Définition. 4.8 (Plan-solution)
Une conjecture $\chi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ est un *plan-solution* pour un problème de planification $\mathcal{P} = (s_0, \mathcal{O}, g)$ si :

- l'ensemble des contraintes d'ordre $\prec$ et l'ensemble des contraintes d'instanciation $\mathcal{I}$ sont cohérents ;
- toutes les linéarisations $\lambda \in \mathit{completion}(\chi)$ définissent une séquence d'états cohérents $\langle s_0, \ldots, s_i, \ldots s_n \rangle$ pour $0 \leq i \leq n$ tels que
  - le but g est vérifié dans l'état s_n, *i.e.,* $g \subseteq s_n$;
  - λ ne formule aucune hypothèse, *i.e.,* $\mathit{assump}(\lambda) = \emptyset$.

Malheureusement, la seconde partie de la définition, qui consiste à tester systématiquement pour chaque linéarisation d'une conjecture si elle décrit une séquence d'états cohérents conduisant à un état but, ne définit pas une condition aisément calculable. Par conséquent, nous avons besoin de spécifier un ensemble de propriétés traduisant de façon pratique cette condition. Pour cela, nous la réexprimons en termes de *réfutations*.

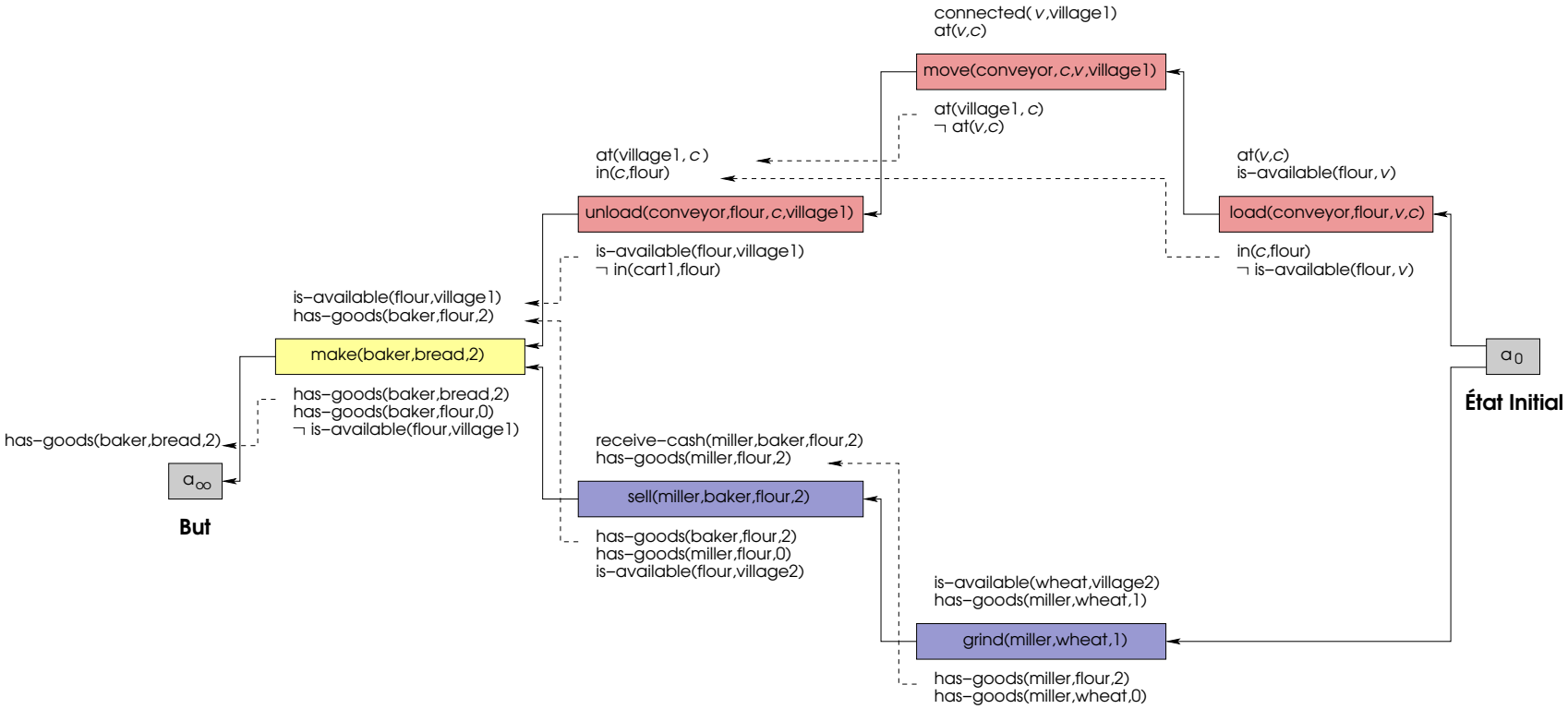


FIG. 4.4: Le but est défini par l'ensemble de propriétés $\{\mathsf{has\text{-}goods}(\mathsf{baker}, \mathsf{bread}, 2)\}$.

Bien que la conjecture ne formule plus d'hypothèse, elle peut ne pas être assez contrainte pour garantir que toutes les séquences d'actions possibles définies par $\prec$ soient exemptes de conflit. En effet, un lien causal $a_i \xrightarrow{p} a_j$ n'interdit pas que d'autres actions soient exécutées entre a_i et a_j. Pour s'en persuader, considérons la conjecture de la figure 4.5. Supposons que l'effet $\neg q$ soit produit par l'action a_k, et que q soit unifiable avec p. L'action a_k invalide potentiellement une précondition nécessaire à l'exécution de a_j. En l'absence de contrainte d'ordre entre a_k et les actions a_i et a_j, la conjecture définit au moins une sous séquence d'actions $\langle a_i, \ldots, a_k, \ldots, a_j \rangle$ invalide : la propriété du monde représentée par p n'est pas vérifiée dans l'état précédant l'exécution de a_j. On retrouve ici la notion d'exclusion mutuelle entre actions sur laquelle s'appuie les techniques de planification comme TWEAK (Chapman, 1987) ou GRAPHPLAN (Blum and Furst, 1997).

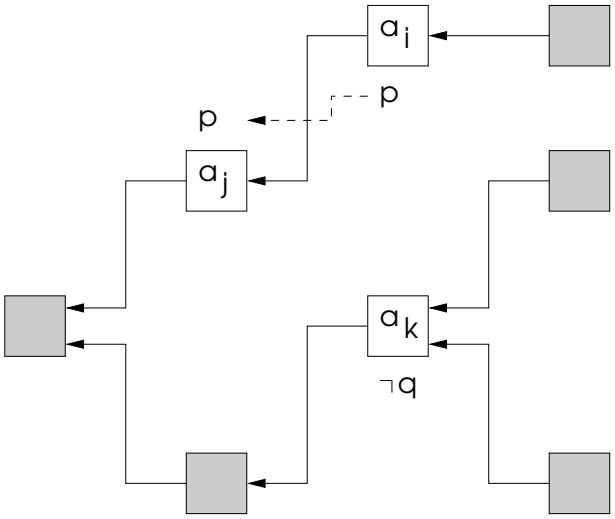


FIG. 4.5: Exemple de réfutation. p et q sont deux prédicats unifiables.

Pour capturer cette condition et ainsi supprimer les séquences d'actions non valides, nous définissons ce que nous appelons une *réfutation*.

Définition. 4.9 (Réfutation)
Une *réfutation* portant sur une conjecture $\chi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ est un couple $(a_k, a_i \xrightarrow{p} a_j)$ tel que :

- a_k a pour effet $\neg q$ avec p et q unifiables ;
- les contraintes d'ordre $a_i \prec a_k$ et $a_k \prec a_j$ sont cohérentes avec $\prec$,
- les contraintes d'instanciation résultant de l'unification de p et q sont cohérentes avec $\mathcal{I}$.

Dans la suite de ce manuscrit, nous utiliserons le terme de *menace* pour caractériser l'ensemble des hypothèses et des réfutations d'une conjecture.

Proposition. 4.1 Une conjecture $\chi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ est un plan-solution pour un problème de planification $\mathcal{P} = (s_0, \mathcal{O}, g)$ si :

- les ensembles de contraintes d'ordre $\prec$ et d'instanciation $\mathcal{I}$ sont cohérents ;
- χ ne contient aucune menace.

Lemme. 4.1 Soit $\chi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ une conjecture et un lien causal $(a_i \xrightarrow{p} a_n) \in \mathcal{C}$. Nécessairement $p \in s_n$ s'il n'exite pas de réfutation $(a_k, a_i \xrightarrow{p} a_n)$.

Preuve. 4.1 Preuve par induction sur la longueur de $\lambda \in \mathit{completion}(\chi)$:

Cas de base : soit $\chi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ avec $\mathcal{A} = \{a_0, a_\infty\}$. $\mathit{completion}(\chi) = \{\lambda\}$ et $\lambda = \langle a_0, a_\infty \rangle$. $s_0 = s_n$ et, par définition, il n'existe pas de réfutation possible $(\forall p \in s_0, p \in s_n)$.

Induction : Supposons que le lemme est vérifié pour χ ayant n actions. Montrons qu'il est également vrai pour χ composée de $n+1$ actions. Soit $\lambda \in \mathit{completion}(\chi)$ avec $\lambda = \langle a_0, \ldots, a_{n-1}, a_n \rangle$ $(a_n = a_\infty)$ et $\lambda' = \langle a_0, \ldots, a_{n-1} \rangle$. D'après l'hypothèse d'induction, $\forall (a_i \xrightarrow{p} a_{n-1})$ pour $0 \leq i < n-1$, $p \in s_{n-1}$ s'il n'existe pas de réfutation $(a_k, a_i \xrightarrow{p} a_{n-1})$ pour $0 \leq k < n-1$. Par définition, $s_n = ((s_{n-1} \cup \mathit{assump}(a_{n-1}) - \mathit{del}(a_{n-1})) \cup \mathit{add}(a_{n-1})$. Par conséquent, $\forall p \in s_n$, soit $p \in \mathit{add}(a_{n-1})$, soit $p \in s_{n-1} \cup \mathit{assump}(a_{n-1})$ et $p \notin \mathit{del}(a_{n-1})$. Dans le premier cas, p est produit par a_{n-1} et il n'y a pas de réfutation possible. Dans le second cas, p a été produit par λ' et n'est pas réfutée par a_{n-1}. Donc, dans tous les cas, le lemme est vérifié.

Preuve. 4.2 Soit $\chi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$. $\prec$ et $\mathcal{I}$ sont cohérents et il n'y a pas de menace dans χ. Donc, $\forall \lambda \in \mathit{completion}(\chi)$, λ définit une séquence d'états $\langle s_0, \ldots, s_n \rangle$ telle que $s_i = (s_{i-1} - \mathit{del}(a_{i-1})) \cup \mathit{add}(a_{i-1})$ car $\mathit{assump}(\chi) = \emptyset$. Comme il n'y a pas de réfutation dans χ, $g \subseteq s_n$. Cela se démontre par l'absurde. Supposons qu'il existe $p \in g$ et $p \notin s_n$. Comme p ne peut être une hypothèse $(\exists (a_i \xrightarrow{p} a_n))$, l'absence de p dans s_n est due à une réfutation d'après le lemme 4.1. Ceci est contradictoire avec l'absence de menace. Par conséquent, $g \subseteq s_n$ et χ est un plan-solution.

Exemple 4.9 Pour illustrer la notion de plan-solution, nous donnons à la figure 4.6 le plan-solution complet obtenu par les agents de l'exemple 4.8. Dans un soucis de lisibilité, seules les actions et les contraintes d'ordre sont représentées.

Conclusion

Ce chapitre nous a permis d'introduire le fondement de la synthèse dialectique de plans ainsi que ses concepts. À partir d'une conjecture initiale représentant le but à atteindre, les agents cherchent à raffiner récursivement les hypothèses formulées par les autres agents. Si un agent détecte que la conjecture n'est pas correcte, *i.e.,* contient une réfutation, il doit alors la réfuter. Finalement, une conjecture est considérée comme plan-solution lorsqu'elle ne peut plus être menacée.

La notion de conjecture introduite formellement dans ce chapitre permet de représenter des plans, non plus comme une simple séquence d'actions totalement instanciées, mais comme un ensemble d'actions partiellement ordonnées et partiellement instanciées. Ceci est un atout pour la synthèse de plans dans un contexte multi-agent. En effet, le choix d'une telle représentation permet aux agents de co-construire des plans pouvant être exécutés de manière concurrente, mais également des plans reflétant leurs croyances limitées.

Il faut maintenant spécifier formellement les mécanismes dialogaux qui permettent aux agents de co-construire un plan-solution. C'est ce que nous proposons dans le chapitre suivant.

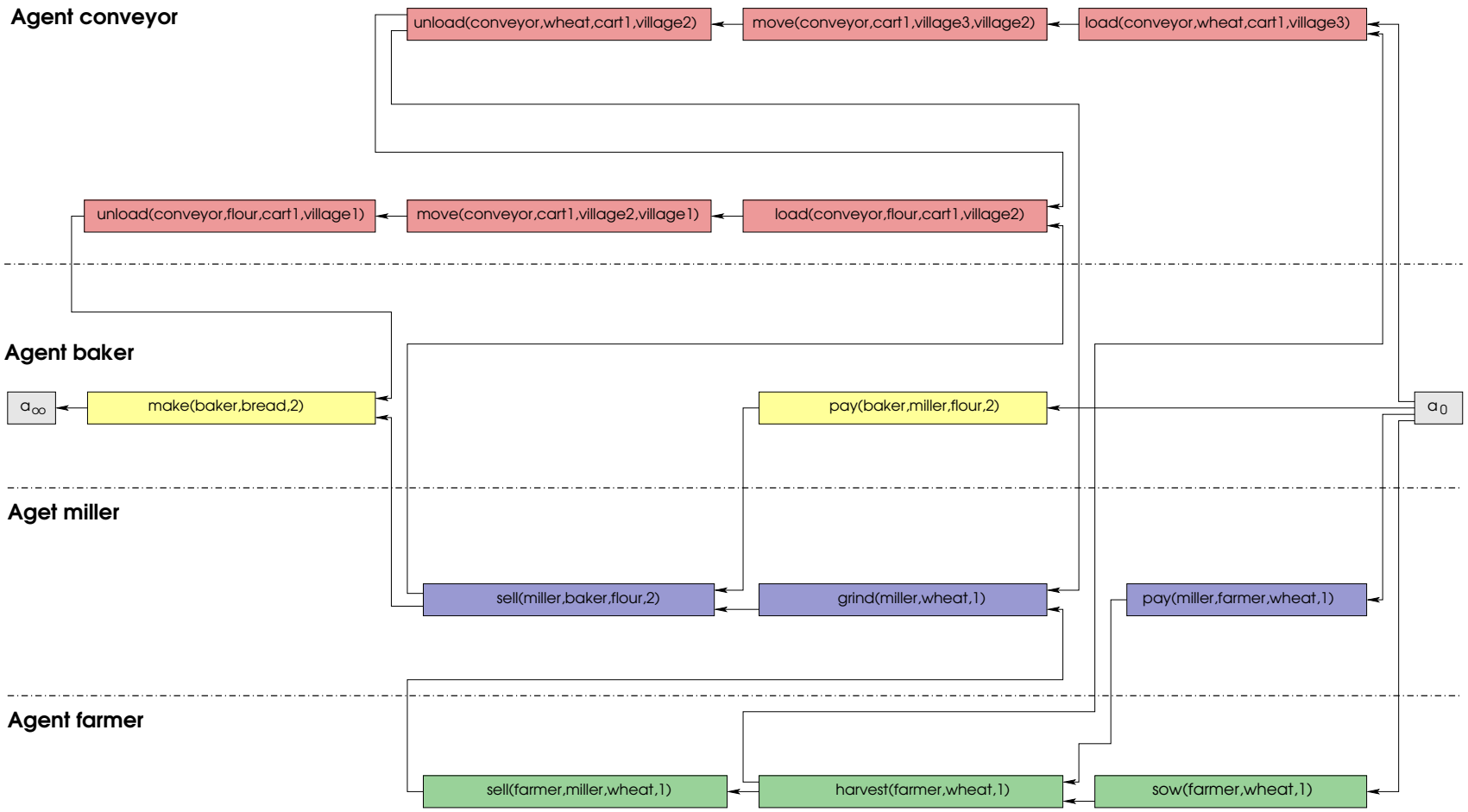


FIG. 4.6: Exemple de plan-solution.

Chapitre 5

Mécanismes dialectiques pour la synthèse de plans

Introduction

Le chapitre précédent a permis d'entrevoir les mécanismes nécessaires à la mise en œuvre de la synthèse dialectique de plans au travers des définitions de conjecture et de réfutation. Dans ce chapitre, il s'agit maintenant de formaliser ces mécanismes permettant à un groupe d'agents de raisonner conjointement sur l'élaboration d'un plan-solution. Ces mécanismes sont complexes. D'une part, les agents doivent être capables d'interagir en respectant un certain nombre de règles qui garantissent le bien fondé du raisonnement produit et, d'autre part, ils doivent être capables de démontrer la validité des hypothèses formulées par les autres agents, de réfuter les conjectures incorrectes ou encore de les réparer lorsque celles-ci ont été précédemment réfutées. Par conséquent, il faut différencier deux types de mécanismes : les règles du dialogue qui spécifient quand un acte de dialogue peut être énoncé et les mécanismes qui servent de support à la production du contenu de l'acte.

Dans un premier temps, nous présenterons le principe général de la synthèse dialectique de plans ainsi que les règles qui régissent le raisonnement produit par les agents. Puis, nous introduirons plus particulièrement la boucle de raisonnement par conjecture - réfutation qui guide les interactions des agents. Finalement, nous décrirons les différents mécanismes et algorithmes de raffinement, de réfutation et de réparation.

5.1 Raisonner par conjecture - réfutation

La synthèse dialectique de plans repose sur une approche conventionnelle du dialogue (Singh, 1998). En d'autres termes, tous les agents impliqués dans la synthèse doivent respecter un certain nombre de règles. Ces règles garantissent la validité du raisonnement ainsi que celle du plan-solution produit. Dans ce paragraphe nous commençons par énoncer le principe de la synthèse dialectique de plans puis nous

présentons les règles du raisonnement par conjecture - réfutation au travers des actes de dialogue utilisés dans notre approche.

5.1.1 Le principe

Chaque agent possède un *tableau de démonstration* (Hamblin, 1971) dans lequel il enregistre les propositions des autres agents. D'un point de vue algorithmique, le tableau de démonstration peut être vu comme un graphe orienté dont les nœuds représentent des conjectures. Chaque arête sortant d'un nœud χ est un *opérateur dialectique* qui transforme une conjecture χ en une conjecture successeur χ', traduisant ainsi les modifications proposées par les agents en termes de raffinement, de réfutation et de réparation. Par conséquent, le tableau de démonstration définit l'état du dialogue mais également l'espace de recherche co-construit par les agents.

Les différents actes de dialogue utilisés dans notre approche sont donnés par le tableau 5.1. Ils se regroupent en deux niveaux :

Les actes de niveau informationnel qui permettent aux agents d'échanger des informations sur les conjectures contenues dans le tableau de démonstration.

Les actes de niveau contextualisation qui permettent de modifier le contexte de l'interaction. C'est par l'intermédiaire de ces actes que les agents vont pouvoir débuter ou encore suspendre l'élaboration d'un plan-solution.

Niveau	**Actes**
Informationnel	*refine*, *refute*, *repair*, *failure*
Contextualisation	*prop.solve*, *prop.failure*, *prop.success*, *ack.failure*, *ack.success*

TAB. 5.1: Tableau des actes de dialogue classés par niveau.

De manière générale, un acte de dialogue se définit par un type, un locuteur et un contenu : le type caractérise le type de l'acte (*e.g.,* refine, refute, repair, *etc.*) ; le locuteur, l'agent qui a énoncé l'acte et le contenu, l'information à communiquer.

Classiquement, le tableau de démonstration de chaque agent est initialisé avec une conjecture χ_0 définie par :

- deux actions a_0, a_∞ telles que les préconditions de a_∞ représentent le but g soumis à l'ensemble des agents et les effets de a_0 l'état initial des croyances de l'agent ;
- une relation d'ordre $(a_0 \prec a_\infty)$;
- les contraintes d'instanciation relatives à la description des préconditions et des effets de a_0, a_∞ ;
- un ensemble de liens causaux vide.

Exemple 5.1 Si le but est is-available(flour,village1) et que l'état initial des croyances de l'agent transporteur est défini par :

$$beliefs(\text{conveyor}) = \left\{ \begin{array}{l} \text{connected}(\text{village1}, \text{village2}), \\ \text{connected}(\text{village2}, \text{village1}), \\ \text{is-available}(\text{flour}, \text{village2}), \\ \text{at}(\text{cart1}, \text{village2}) \end{array} \right\}$$

la conjecture initiale χ_0 contenue dans son tableau de démonstration peut être représentée par la figure suivante :

FIG. 5.1: Exemple de conjecture initiale.

Les tableaux de démonstration des agents ne contiennent que la partie publique du raisonnement produit par les autres agents. En d'autres termes, seules les hypothèses ainsi que les effets des actions liés par un lien causal sont communiqués.

Exemple 5.2 Reprenons le raffinement énoncé par le meunier dans l'exemple 4.8 pour vérifier l'hypothèse has-goods(baker,flour,2). Ce raffinement est donné à la figure 5.2. Les effets encadrés ne seront pas transmis aux autres agents. En effet, ils n'apportent pas d'information utile. Seul le tableau de démonstration du meunier possédera la conjecture complète.

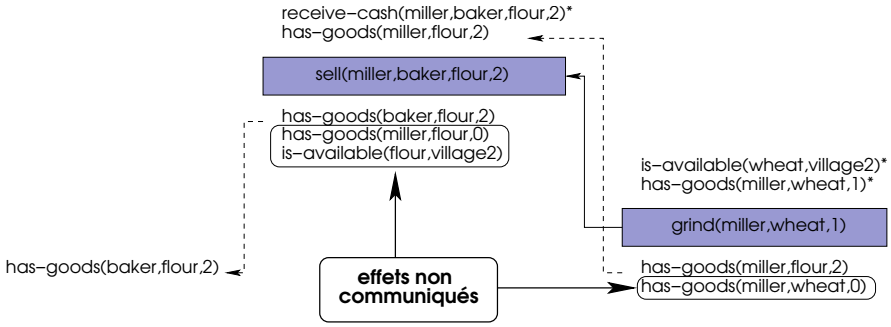


FIG. 5.2: Exemple de raffinement.

Avant de présenter plus en détail les règles du dialogue, nous souhaitons expliciter les propriétés du modèle de communication sous-jacent. Les agents communiquent par diffusion asynchrone de messages. Chaque message correspond à un acte de dialogue défini par le tableau de 5.1. Le temps de réception d'un message par un agent

est fini et tous les messages sont délivrés dans l'ordre causal de leur émission (Lamport, 1978), garantissant ainsi que le tableau de démonstration décrit toujours une vue causalement cohérente du dialogue.

5.1.2 Les règles de conjecture - réfutation

Nous associons à chaque opérateur dialectique un acte du niveau informationnel comme présenté par le tableau 5.1. Pour caractériser complètement les mécanismes du dialogue par cycles de conjecture - réfutation, nous proposons de les décrire en s'inspirant de (Maudet, 2001). Nous définissons ainsi pour chaque acte du niveau informationnel trois sous-ensembles de règles :

1. *les règles de rationalité* qui définissent les propriétés que doit vérifier le contenu informationnel de l'acte pour être valide au sens du dialogue.
2. *les règles du dialogue* qui spécifient les actes de dialogue autorisés pour garantir la cohérence du raisonnement produit par les agents.
3. *les règles de mise à jour* qui précisent comment le tableau de démonstration doit être modifié en fonction des actes de dialogue.

La formalisation de ces règles est donnée par le tableau 5.2. À la réception ou à l'émission de chaque acte de dialogue, les agents en vérifient la conformité.

5.1.3 Les règles de contextualisation

Les règles du niveau informationnel ne suffisent pas à définir totalement le raisonnement par conjecture - réfutation. Il reste encore à préciser comment les agents débutent et clôturent la synthèse d'un plan. Nous proposons de formaliser l'ensemble des règles de contextualisation par un automate à états finis (cf. figure 5.3). Les états de l'automate définissent les états du dialogue et les arêtes les différents actes de dialogues du tableau 5.1. À chaque réception ou émission d'un acte par un autre agent, l'automate maintient à jour l'état courant du dialogue. Nous utilisons la notion *?* et *!* devant un acte de dialogue pour indiquer que l'acte est respectivement reçu (*e.g.,* *?refine()*) ou émis (*e.g.,* *!refine()*) par l'agent.

Essayons maintenant d'étudier plus en détails l'automate de contextualisation. Initialement, les agents sont dans un état **IDLE**. Cet état correspond à une attente des agents d'un but à résoudre. À la réception ou à l'émission de l'acte *prop.solve*, le dialogue passe dans l'état **CoRe** dans lequel les agents échangent des raffinements, des réfutations ou encore des réparations qui sont stockés dans le tableau de démonstration en fonction des règles définies par le niveau informationnel (cf. tableau 5.2). Notons que l'acte *prop.solve* peut être énoncé, soit par un opérateur humain, soit par un agent qui souhaite soumettre à une société d'agents un but à résoudre. La phase d'ouverture du dialogue est équivalente dans les deux cas. Le contenu informationnel de l'acte *prop.solve* représente les buts soumis à la société d'agents. Lorsque le dialogue est ouvert, les agents sont dans l'état **CoRe**. La sortie de cet état peut se produire dans deux cas :

refine(ρ, p, χ) où ρ est un raffinement et p l'hypothèse raffinée de la conjecture χ.

Rationalité : l'agent doit vérifier que :
- χ est une conjecture du tableau de démonstration ;
- h est une hypothèse formulée par χ ;
- ρ n'a pas déjà été proposé comme raffinement de p ;
- la conjecture résultat χ' de l'application de l'opérateur de raffinement ρ est valide au sens où ses contraintes d'ordre et d'instanciation sont cohérentes.

Dialogue : les agents peuvent raffiner toutes les hypothèses de χ' ou réfuter χ'.

Mise à jour : ajouter χ' comme raffinement de l'hypothèse p de χ dans le tableau de démonstration.

refute(ϕ, χ) où ϕ est une réfutation et χ la conjecture réfutée.

Rationalité : l'agent doit vérifier que :
- χ est une conjecture du tableau de démonstration ;
- ϕ n'a pas déjà été proposée comme réfutation à l'encontre de χ.

Règles : les agents peuvent réparer χ.

Mise à jour : ajouter ϕ comme réfutation de χ dans le tableau de démonstration.

repair(ψ, ϕ, χ) où ψ est une réparation de χ en réponse à la réfutation ϕ.

Rationalité : l'agent doit vérifier que :
- χ est une conjecture du tableau de démonstration ;
- ϕ est une réfutation formulée à l'encontre de χ ;
- ψ n'a pas déjà été proposée comme réparation de χ en réponse à la réfutation ϕ ;
- la conjecture résultat χ' de l'application de ψ est valide au sens où ses contraintes d'ordre et d'instanciation sont cohérentes.

Règles : les autres agents peuvent raffiner toutes les hypothèses de χ' ou réfuter χ'.

Mise à jour : ajouter χ' comme réparation de la réfutation ϕ de χ.

failure(Φ,χ) où Φ est une menace, *i.e.,* une hypothèse ou une réfutation de χ.

Rationalité : l'agent doit vérifier que :
- χ est une conjecture du tableau de démonstration ;
- Φ est une hypothèse ou une réfutation formulée à l'encontre de χ ;

Règles : $\emptyset$

Mise à jour : marquer Φ comme ne pouvant être résolue par l'agent qui a énoncé l'acte.

TAB. 5.2: Règles du dialogue par conjecture - réfutation.

- ▷ **Sur proposition de l'agent.** Lorsque l'agent initie une proposition de sortie sur échec (resp. sur succès), l'agent énonce l'acte *prop.failure* (resp. *prop.success*). Le dialogue passe alors dans un état **Failure** (resp. **Success**) dans lequel l'agent attend les acquittements locaux des autres agents. Si tous les agents acquittent la proposition alors les conditions de terminaison sur échec (resp. sur succès) sont vérifiées.
- ▷ **Sur proposition d'un autre agent.** c'est-à-dire lorsqu'un agent reçoit d'une proposition de sortie sur échec, *prop.failure*, ou respectivement à la réception d'une proposition de sortie sur succès, *prop.success*. La fermeture du dialogue sur proposition d'un autre agent nécessite un découpage en deux phases.

  Une phase de vérification. Lorsqu'un de ces deux actes est reçu, le dialogue passe en instance de sortie, représenté par les états **IF** (Instance Failure) et **IS** (Instance Success). Ces deux états du dialogue correspondent à la vérification locale des propriétés de sortie de dialogue. Dans le cas d'une proposition de sortie sur échec, l'agent doit vérifier qu'il n'existe pas localement de solution au problème initialement soumis à la société d'agents et dans le cas d'une proposition de sortie sur succès que la conjecture constituant le contenu informationnel de l'acte *prop.success* vérifie bien localement les propriétés d'un plan-solution définies par la proposition 4.1.

  Une phase d'acquittement. Si les propriétés locales de terminaison sur échec (resp. succès) sont vérifiées alors l'agent acquitte la proposition de sortie en énonçant *ack.failure* (resp. *ack.success*), et le dialogue passe dans l'état **Failure** (resp. **Success**). Les états **Failure** et **Success** correspondent à l'attente par un agent des acquittements des autres agents. Notons que le non acquittement par un agent de la proposition de sortie ne se traduit pas par un acte spécifique du niveau de contextualisation. En effet, supposons qu'un agent soit dans l'état **IF** (*i.e.,* Instance Failure), si l'agent reçoit ou soumet un raffinement, une réfutation ou une réparation, cela signifie qu'un des agents est capable de poursuivre le processus de synthèse de plans. Les agents doivent par conséquent réévaluer le tableau de démonstration puisque celui-ci a été modifié. De manière similaire, lorsqu'un agent se trouve dans l'état **IS** (*i.e.,* Instance Success), si l'agent reçoit ou soumet une réfutation portant sur la conjecture proposée comme solution, la conjecture précédemment proposée comme solution est invalidée et l'agent doit à nouveau poursuivre le processus de synthèse de plans. Dans les deux cas, tous les agents impliqués dans le dialogue sont contraints à poursuivre le processus de synthèse de plans ce qui se traduit par un retour dans l'état **CoRe** de l'automate de contextualisation.

L'automate de contextualisation garantit que la terminaison du processus de synthèse de plans ne peut avoir lieu que si l'ensemble des agents sont dans l'incapacité de faire progresser la co-construction d'un plan ou alors qu'un plan-solution est présent dans le tableau de démonstration. Ceci revient à calculer un état global pour déterminer si l'une des conditions de sortie est vérifiée.

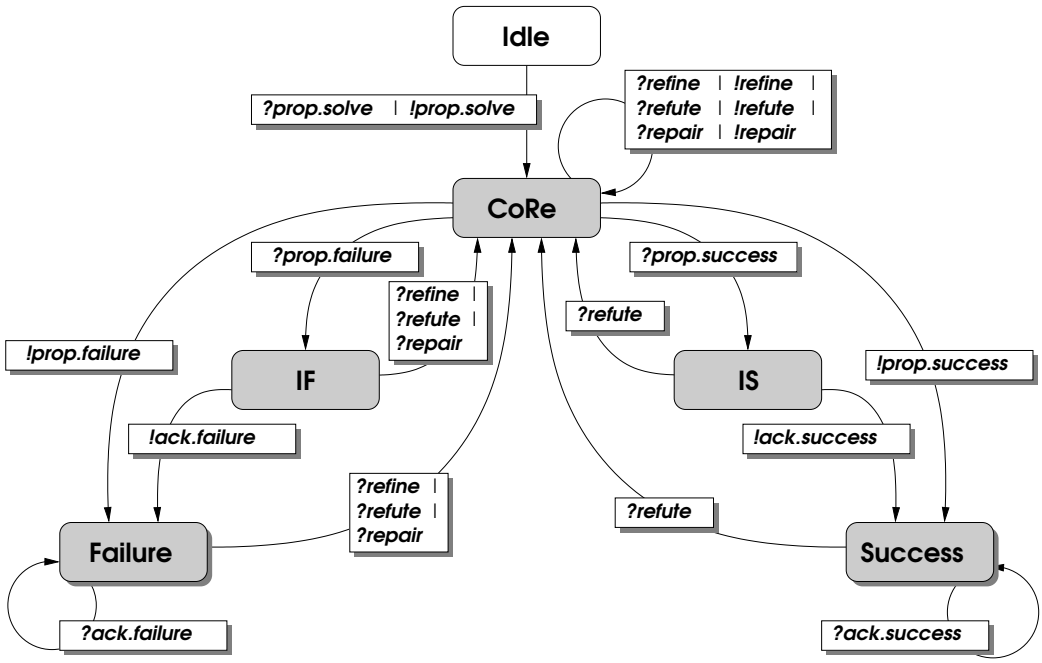


FIG. 5.3: Automate de contextualisation du dialogue par conjecture – réfutation.

5.2 Les cycles de conjecture - réfutation

La définition des règles, qui régissent le dialogue, est nécessaire mais ne spécifient pas les mécanismes qui guident l'interaction. Pour répondre à cette question, nous proposons, dans ce paragraphe, d'introduire la boucle de raisonnement par conjecture - réfutation. Cette boucle, qui s'inspire des algorithmes de recherche dans un espace de plans (Penberthy and Weld, 1992), dirige les propositions des agents. À chaque itération, un agent choisit une conjecture et cherche à lui appliquer un opérateur dialectique faisant ainsi progresser collectivement la recherche d'un plan-solution.

L'exploration de l'espace de conjectures, représenté par les tableaux de démonstration des agents, est obtenue en appliquant itérativement la procédure suivante :

1. Choix d'une conjecture χ non encore explorée ;
2. Choix de l'opérateur dialectique à appliquer à χ ;
3. Calcul des modifications à apporter à χ ;
4. Soumission des modifications aux autres agents.

L'exécution itérative de cette procédure définit ce nous appelons les *cycles de conjecture - réfutation*. Le raisonnement se poursuit tant que l'espace de recherche ne contient pas au moins une conjecture qui respecte les propriétés d'un plan-solution (terminaison sur succès cf. proposition 4.1) ou alors qu'aucun agent ne peut plus proposer d'opérateurs dialectiques pour faire progresser la synthèse de plans (terminaison sur échec). Lorsque l'une de ces deux conditions de terminaison est vérifiée alors les agents doivent mettre fin à leur raisonnement.

5.2.1 La boucle de raisonnement CoRe

La boucle de raisonnement qui décrit les cycles de conjecture - réfutation est donnée en pseudo-code par l'algorithme 5.1. La première opération effectuée par la boucle de raisonnement est de choisir une conjecture du tableau de démonstration. Cette sélection ne peut porter que sur les conjectures non terminales. Une conjecture est considérée comme terminale si l'agent ne peut plus proposer d'opérateur dialectique à lui appliquer. Si toutes les conjectures sont terminales (ligne 3), alors l'agent ne peut plus faire progresser la synthèse de plans. Par conséquent, l'agent doit soumettre aux autres agents une proposition de terminaison sur un échec et mettre fin à sa propre boucle de raisonnement (ligne 5). La proposition de terminaison est alors soumise aux autres agents pour acceptation.

Soulignons que le choix de la conjecture à explorer est important pour garantir de bonnes performances à l'algorithme de synthèse de plans. En effet, c'est de ce choix local que découle la politique globale d'exploration de l'espace de conjectures par les agents. Ce point précis sera discuté dans le paragraphe 5.2.2. Pour l'instant, considérons que ce choix est effectué de manière non déterministe.

Maintenant qu'une conjecture a été sélectionnée, l'agent doit déterminer si cette conjecture est un plan-solution. Rappelons qu'une conjecture χ est un plan-solution (ligne 8) si χ ne formule plus aucune hypothèse et ne peut pas être réfutée. La première propriété est vérifiée par la procédure `Assumptions(`χ`)` qui calcule l'ensemble des hypothèses présentes dans une conjecture. Cette procédure peut être implémentée de manière efficace en utilisant une table de hachage : à chaque ajout d'une nouvelle action a à la conjecture, les hypothèses de a sont ajoutées à la table et, à chaque ajout d'un lien causal à la conjecture, les hypothèses correspondantes sont supprimées de la table. La seconde propriété est vérifiée par la procédure `Refutations(`χ`)` qui calcule l'ensemble des réfutations pouvant être formulées à l'encontre de la conjecture (cf. § 5.3.2).

Lorsqu'une conjecture χ vérifie localement ces deux propriétés (ligne 9), l'agent doit proposer à l'ensemble des autres agents de terminer le processus de synthèse de plans en soumettant χ comme plan-solution. De manière symétrique aux mécanismes de terminaison sur échec, l'agent met fin à son raisonnement.

Si la conjecture χ ne vérifie pas les propriétés d'un plan-solution, l'agent doit poursuivre son raisonnement en proposant un nouvel opérateur dialectique. L'ordre dans lequel s'effectue la sélection des opérateurs dialectiques est important dans la prise en compte des performances de l'algorithme. En effet, même si ce choix ne guide pas directement la politique d'exploration de l'espace de conjectures co-construit par les agents, il détermine les performances de l'algorithme (cf. § 5.2.2).

Considérons tout d'abord qu'un agent décide d'appliquer un opérateur de raffinement sur une hypothèse de χ (ligne 14). L'agent calcule les raffinements possibles par l'intermédiaire de la procédure `Refine(`ϕ, χ`)` qui prend en paramètres l'hypothèse ϕ à raffiner ainsi que la conjecture χ. Si aucun raffinement ne peut être proposé, alors

ϕ est marquée comme ne pouvant être résolue et l'agent propage son échec. Sinon, l'agent choisit un raffinement et le soumet à la délibération des autres agents.

Le même mécanisme s'applique pour les réparations (ligne 24). Si ϕ est une réfutation qui a été précédemment formulée à l'encontre de la conjecture, χ doit être réparée. L'agent calcule alors les réparations possibles pour la réfutation grâce à la procédure `Repair`(ϕ, χ). Si aucune réparation n'a été produite, l'agent marque la réfutation comme ne pouvant être résolue et l'agent propage son échec. Dans le cas contraire, l'agent informe les autres agents des réparations produites.

Finalement, si ϕ est une réfutation (ligne 35), l'agent soumet simplement la réfutation aux autres agents afin de leur indiquer que dorénavant la conjecture χ n'est pas correcte et doit être réparée.

Proposition. 5.1 La procédure CoRe est correcte et complète : chaque fois qu'il existe, du point de vue d'un agent, une solution pour $\mathcal{P} = (s_0, \mathcal{O}, g)$, cet agent termine CoRe en faisant une proposition de succès avec un plan-solution.

Preuve. 5.1 (Correction) Par définition, la conjecture initiale $\chi_0 = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ est définie par $\mathcal{A} = \{a_0, a_\infty\}$, $\prec$ un ensemble de contraintes d'ordre cohérent et $\mathcal{I}$ un ensemble de contraintes d'instanciation cohérent. Lorsque CoRe fait une proposition de succès, la conjecture χ proposée ne contient plus de menace et tous les opérateurs de raffinements et de réparation maintiennent $\prec$ et $\mathcal{I}$ cohérents. Ceci est garanti par les règles du dialogue par conjecture - réfutation. Par conséquent, d'après la proposition 4.1, χ est un plan-solution.

La procédure CoRe est également correcte au niveau du groupe car le cycle de conjecture - réfutation ne s'arrête que si et seulement si un agent a initié une proposition de succès sur χ (χ vérifie localement les propriétés d'un plan-solution pour l'agent initiateur) et que χ n'est pas réfutable par les autres agents. Par conséquent, χ est un plan-solution pour $\mathcal{P}$ au niveau du groupe.

Remarque. 5.1 Chaque ajout d'une action au sein d'une conjecture entraîne le renommage systématique des variables. Ceci permet de garantir que $\mathcal{I}$ reste cohérent.

Preuve. 5.2 (Complétude) Il faut montrer qu'un moins une des traces d'exécution de CoRe renvoie un plan-solution d'un point de vue d'un agent lorsqu'il existe. Preuve par induction, sur la longueur k d'un plan-solution.

Cas de base ($k = 0$) *:* le plan vide χ est solution de $\mathcal{P}$. Par conséquent, χ ne comporte pas de menace et CoRe fait immédiatement une proposition de succès.

Induction : supposons que CoRe est complet pour un problème nécessitant une solution de longueur k. Soit $\mathcal{P} = (s_0, \mathcal{O}, g)$ nécessitant un plan-solution $\langle a_0, \ldots, a_k \rangle$ de longueur $k + 1$. Comme a_k est nécessaire pour démontrer le but g, il existe au moins un agent qui proposera de manière non déterministe un opérateur dialectique à appliquer à la conjecture initiale χ_0 contenant a_k. Soit

5.1 : Boucle de raisonnement CoRe

```
1  while raisonne = true do
2  |  conjectures ← les conjectures non terminales du tableau ;
3  |  if conjectures est vide then
4  |  |  Soumettre une proposition d'échec ;
5  |  |  raisonne ← false ;
6  |  else
7  |  |  Sélectionner une conjecture χ ∈ conjectures ;
8  |  |  menaces ← Assumptions(χ) ∪ Refutations(χ) ;
9  |  |  if menaces est vide then
10 |  |  |  Soumettre une proposition de succès portant sur χ ;
11 |  |  |  raisonne ← false ;
12 |  |  else
13 |  |  |  Sélectionner une menace φ ∈ menaces ;
14 |  |  |  if φ est une hypothèse then
15 |  |  |  |  raffinements ← Refine(φ, χ) ;
16 |  |  |  |  if raffinements est vide then
17 |  |  |  |  |  Marquer φ comme ne pouvant être résolue ;
18 |  |  |  |  |  Soumettre l'échec de réparation portant sur φ ;
19 |  |  |  |  else
20 |  |  |  |  |  Sélectionner un raffinement ρ ∈ raffinements ;
21 |  |  |  |  |  Soumettre ρ comme raffinement φ de χ;
22 |  |  |  |  end
23 |  |  |  end
24 |  |  |  else if φ est une réfutation déjà formulée à l'encontre de χ then
25 |  |  |  |  reparations ← Repair(φ, χ) ;
26 |  |  |  |  if reparations est vide then
27 |  |  |  |  |  Marquer φ comme ne pouvant être résolue ;
28 |  |  |  |  |  Soumettre l'échec de réparation portant sur φ ;
29 |  |  |  |  else
30 |  |  |  |  |  Sélectionner une réparation ψ ∈ reparations ;
31 |  |  |  |  |  Soumettre ψ comme réparation de φ de χ;
32 |  |  |  |  end
33 |  |  |  end
34 |  |  |  else
35 |  |  |  |  Soumettre φ comme nouvelle réfutation de χ ;
36 |  |  |  end
37 |  |  end
38 |  end
39 end
```

$s_{k+1} = ((s_k \cup assump(a_k)) - del(a_k)) \cup add(a_k)$, l'état suivant l'exécution de a_k. Au prochain appel récursif de CoRe par les agents la conjecture χ_1 résultat de l'opérateur dialectique contenant a_k est constitué d'au moins de trois actions

$\{a_0, a_k, a_\infty\}$. χ_1 est équivalent à la conjecture initiale d'un problème défini par l'état s_0 et $g = assump(a_k) \cup assump(a_\infty)$. Ce problème admet une solution $\langle a_0, \ldots, a_j \rangle$ de longueur $j \leq k$. Par hypothèse d'induction, les appels récursifs de CoRe sur la conjecture χ_1 produisent une trace qui trouve le plan-solution $\langle a_0, \ldots, a_j \rangle$ et CoRe propose de sortir sur succès.

5.2.2 Les heuristiques pour la synthèse de plans

La formalisation de la boucle de raisonnement a permis de souligner l'importance que revêtent les choix de la conjecture à explorer et de l'opérateur dialectique à appliquer. Dans ce paragraphe, nous proposons d'aller plus loin en étudiant les différentes heuristiques guidant ces choix. L'intérêt de cette étude est double : d'une part, de bonnes heuristiques peuvent améliorer de manière significative les performances globales du système et, d'autre part, participer à la spécification plus fine de la rationalité du raisonnement produit par les agents.

Les heuristiques pour guider l'exploration

Le tableau de démonstration définit un graphe orienté. Par conséquent, il est possible de s'appuyer sur les algorithmes de recherche dans les graphes. Nous proposons d'implanter dans le mécanisme dialectique de conjecture – réfutation une fonction d'évaluation inspirée de l'algorithme de recherche A* (Pearl, 1984). Ceci nous permettra de guider le raisonnement des agents et de définir une borne d'exploration. L'implantation d'un tel algorithme de recherche dans un contexte distribué pose deux problèmes : tout d'abord, A* n'est pas un algorithme distribué et les interactions entre les agents sont asynchrones ; de plus, A* s'appuie sur une fonction évaluation f pour guider la recherche. Dans notre cas, quelles sont les éléments permettant de caractériser cette fonction ?

Considérons tout d'abord le premier problème. Une réponse possible est de mettre en place un protocole de délibération permettant aux agents de décider conjointement de la conjecture à choisir. Cette solution a pour avantage de coller au plus près aux algorithmes classiques de recherche. En effet, une seule conjecture est explorée par l'ensemble des agents au cours d'un cycle de conjecture - réfutation. En dépit de cette considération, cette première solution n'est pas satisfaisante car, outre le temps nécessaire à la délibération, elle ne tire pas parti de l'asynchronisme des agents. Supposons que tous les agents travaillent à un instant donné sur la même conjecture, les agents ne pouvant pas proposer leur aide sont alors bloqués tant qu'une nouvelle conjecture n'a pas été choisie.

Ces raisons nous poussent à envisager une autre approche. Supposons que nous définissions une fonction d'évaluation commune à tous les agents. Nous obtenons l'avantage d'une recherche guidée au sens de A* au niveau des agents. En outre, rien n'interdit aux agents qui ne peuvent apporter leur aide d'explorer les autres conjectures du tableau de démonstration. Les agents choisissent alors la conjecture

suivante en fonction des critères définis par la fonction d'évaluation commune. Au prochain cycle de conjecture - réfutation, les tableaux de démonstration contiendront les nouvelles conjectures proposées par les agents. Par conséquent, le calcul de la fonction d'évaluation tiendra compte de ces modifications, permettant ainsi d'obtenir une recherche globale dirigée par l'heuristique. Finalement, en terme de performance, cette solution est plus avantageuse. En effet, le nombre de messages envoyés par les agents est minimisé puisque les agents n'ont pas à délibérer pour déterminer la conjecture à explorer. En revanche, les agents doivent partager la même fonction d'évaluation.

Intéressons nous maintenant au second problème qui consiste à déterminer la fonction d'évaluation. L'algorithme A* s'appuie sur une fonction $f(\chi)$ qui représente une estimation du coût du calcul de la solution. $f(\chi)$ est définie comme la somme de deux fonctions : $g(\chi)$ qui donne le coût de la conjecture courante χ et $h(\chi)$ qui donne le coût restant pour atteindre le meilleur plan-solution à partir de χ. Considérons que $f(\chi)$ est une mesure de la *« complexité »* d'une conjecture, *i.e.,* les bonnes conjectures sont des conjectures simples (avec une complexité faible). Il semble raisonnable d'explorer les conjectures qui sont les plus proches d'un plan-solution (définissant une valeur faible pour $h(\chi)$). Par conséquent, les agents doivent préférer explorer une conjecture χ qui semble plus simple à être démontrée à une conjecture χ' dont la complexité est supérieure à la complexité estimée de χ.

Précisons deux points sur A* avant de donner les deux fonctions $g(\chi)$ et $h(\chi)$ guidant la recherche des agents. Tout d'abord, pour garantir que A* trouve un plan-solution optimal, il faut s'assurer que $h(\chi)$ ne surestime pas le coût restant pour le construire (condition pour que $h(\chi)$ soit « admissible », (Nilsson, 1980)). Toutefois, si le but n'est pas de trouver obligatoirement un plan-solution optimal mais de trouver une solution satisfaisante le plus *rapidement* possible, alors $h(\chi)$ peut surestimer légèrement le coût restant pour atteindre la solution. C'est cette seconde option que nous retenons dans notre approche.

En gardant ces considérations à l'esprit, étudions les différentes métriques nous permettant de définir $g(\chi)$ et $h(\chi)$. La littérature traitant des planificateurs par recherche dans un espace de plans donnent des indices sur les métriques à utiliser (Penberthy and Weld, 1992) :

Nombre d'actions. Intuitivement, une conjecture comportant un nombre d'actions important est plus complexe qu'une conjecture dont le nombre d'actions à exécuter est faible. Afin de prendre en compte une évaluation plus précise du coût des actions pour un domaine précis, il est possible d'associer à chaque action un poids différent. Cette métrique peut naturellement permettre de définir $g(\chi)$.

Nombre de liens causaux. Le nombre de liens causaux apparaît être une alternative pour le calcul de $g(\chi)$. Plus le nombre de liens causaux est important, plus la conjecture est complexe. Le nombre des liens causaux dépend du nombre d'actions de la conjecture. En effet, une action est connectée par au moins un lien causal à une

autre action traduisant la dépendance qui les lie. L'utilisation du nombre de liens causaux pour définir $g(\chi)$ pénalise les conjectures dont plusieurs hypothèses sont raffinées par une seule action, ce qui n'est pas souhaitable.

Nombre d'hypothèses. Puisque les hypothèses définissent des sous buts à atteindre pour les autres agents, le nombre d'hypothèses formulées par une conjecture peut être utilisé pour définir $h(\chi)$. Notons que le nombre d'hypothèses ne définit pas toujours une métrique minorante. Comme nous allons le montrer dans le paragraphe 5.3, le raffinement d'une hypothèse n'entraîne pas forcément l'ajout d'une sous-conjecture, *i.e.,* au moins une action. Par conséquent, $h(\chi)$ peut surestimer le nombre d'actions à ajouter pour atteindre un plan-solution. En dépit de cet inconvénient (Gerevini and Schubert, 1996), cette métrique semble offrir un critère intéressant pour caractériser $h(\chi)$ dans la mesure où la possibilité que $h(\chi)$ soit surestimé est faible. En effet, la majorité des raffinements implique l'ajout d'une sous-conjecture. Toutefois, il est important de noter que le choix de cette métrique fait perdre l'optimalité.

Nombre de réfutations. Intuitivement, il est clair que le nombre de réfutations associées à une conjecture ne permet pas de définir la complexité d'une conjecture. En revanche, cette métrique peut être utile pour caractériser $h(\chi)$. Un argument en faveur de l'utilisation du nombre de réfutations dans le calcul de $h(\chi)$ est le suivant : plus une conjecture est réfutée, plus il sera nécessaire de la réparer. Par conséquent, la distance qui la sépare d'un plan-solution est importante. D'un autre côté, il faut garder à l'esprit que certaines réfutations peuvent se résoudre d'elles-mêmes au cours des raffinements successifs, *e.g.,* lorsque des contraintes d'instanciation sont ajoutées. Le fait d'inclure le nombre de réfutations peut surestimer le coût pour atteindre un plan-solution.

En conclusion, nous proposons de considérer l'heuristique $f(\chi) = g(\chi) + h(\chi)$ avec $g(\chi)$ représentant le nombre d'actions contenues dans la conjecture χ et $h(\chi)$ représentant le nombre d'hypothèses formulées, *i.e.,* l'estimation du nombre d'actions à ajouter à la conjecture pour obtenir un plan-solution.

Les heuristiques pour la sélection de l'opérateur dialectique

Étudions maintenant les heuristiques qui guident les agents dans le choix de l'opérateur dialectique à appliquer. Pour bien comprendre les heuristiques que nous proposons, revenons un instant sur la nature du tableau de démonstration. Le tableau de démonstration des agents peut être vu comme un arbre ET/OU (cf. figure 5.4). Les hypothèses et les réfutations correspondent à des branches ET. En effet, toutes les hypothèses et toutes les réfutations doivent être vérifiées soit en proposant un raffinement soit en proposant une réparation pour qu'un plan-solution soit trouvé. Lorsqu'il existe plusieurs raffinements pour une même hypothèse ou bien plusieurs réparations pour une même réfutation, on a alors une branche OU. Une seule solution est nécessaire pour trouver un plan-solution.

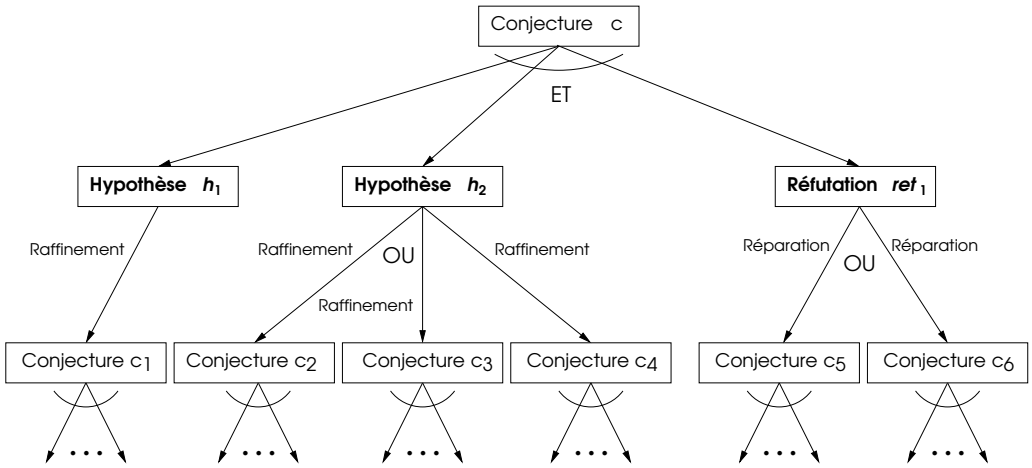


FIG. 5.4: Tableau de démonstration : les hypothèses et les réfutations correspondent aux branches ET tandis que les raffinements et les réparations proposées correspondent aux branches OU.

Cette représentation met en évidence plusieurs règles pour guider le choix de l'opérateur. Supposons tout d'abord que la politique qui dirige le choix des agents soit toujours d'appliquer les opérateurs dialectiques de réfutation avant de proposer ceux de réparation ou de raffinement. Les agents peuvent savoir au plus tôt quelles sont les conjectures qui sont réfutées et, par conséquent, celles qui nécessitent d'être réparées. Si maintenant aucun des agents ne peut réparer une conjecture précédemment réfutée, alors la conjecture doit être abandonnée. Les agents élaguent ainsi plus rapidement les branches du tableau de démonstration. Par conséquent, nous choisissons de privilégier l'opérateur de réfutation avant tout autre opérateur.

À ce stade, il reste à choisir entre l'application d'un opérateur de raffinement et d'un opérateur de réparation. Intuitivement, rien ne sert de raffiner les hypothèses d'une conjecture si elle n'a pas été préalablement réparée. Toutefois, ce choix n'est pas si simple. En effet, le raffinement des hypothèses d'une conjecture peut conduire à la suppression de réfutations précédemment formulées à son encontre (cf. §5.3). Malgré cette restriction, nous faisons le choix de réparer les conjectures réfutées avant de les raffiner.

Intéressons nous maintenant au choix des hypothèses à raffiner et au choix des réfutations à réparer. Les mêmes mécanismes s'appliquent dans les deux cas. Prenons le cas des hypothèses. Il semble préférable de choisir prioritairement les hypothèses pour lesquelles un agent n'est pas capable de produire de raffinement. En effet, lorsqu'une hypothèse ne peut être raffinée, l'agent propage son échec aux autres agents. Étant donné que toutes les hypothèses doivent être raffinées, il suffit qu'une seule hypothèse ne puisse pas l'être par l'ensemble des agents pour invalider la conjecture toute entière. Si c'est le cas, il ne sert à rien de raffiner les autres hypothèses puisque la conjecture ne pourra jamais être un plan-solution. Par conséquent, ce choix permet de couper au plus tôt les conjectures invalides et de limiter le nombre de raffinements produits par les agents.

En conclusion, la stratégie retenue dans notre approche est la suivante :

1. chercher à réfuter ;
2. chercher à réparer en privilégiant les réfutations pour lesquelles il n'existe pas encore de réparation ;
3. chercher à raffiner en privilégiant les hypothèses dont il n'existe pas encore de raffinement.

5.3 Les opérateurs dialectiques

Ce paragraphe présente les mécanismes nécessaires au calcul des opérateurs dialectiques. Nous décrirons successivement les différentes techniques de raffinement, de réparation et de réfutation.

5.3.1 Les mécanismes de raffinement

L'opérateur dialectique de raffinement est utile pour démontrer qu'une hypothèse peut être vérifiée. Comme dans le cadre d'une démonstration mathématique, la preuve qui doit être produite dépend du type de l'hypothèse. Considérons qu'une hypothèse soit définie par un prédicat dont certaines de ses variables ne sont pas instanciées, une technique de raffinement possible est de proposer une instanciation pour ces variables. L'agent démontre ainsi qu'il existe une valeur possible pour les variables laissées libres. Au contraire si toutes les variables sont déjà instanciées, il est nécessaire de proposer une sous-conjecture pour démontrer que cette propriété du monde est atteignable. Pour répondre à cette problématique nous distinguons deux mécanismes de raffinement : les raffinements par ajout d'un lien causal et les raffinements par ajout d'une sous-conjecture.

Les raffinements par ajout d'un lien causal

Le premier mécanisme de raffinement consiste à ajouter un lien causal. Cette technique de raffinement est la plus simple. S'il existe déjà au sein de la conjecture χ une action a_i qui a pour effet p alors il suffit d'ajouter à la conjecture χ un lien causal $(a_i \xrightarrow{p} a_j)$ indiquant dorénavant que l'action a_i démontre l'hypothèse p formulée par l'action a_j. Un exemple de ce type de raffinement est donné à la figure 5.5. Le raffinement solution est un tuple constitué d'un lien causal, d'une contrainte d'ordre, $(a_i \prec a_j)$, ainsi que des contraintes d'instanciation σ nécessaires à l'unification de p avec les effets de a_i.

L'intérêt de ce type de raffinement est de minimiser l'ajout de nouvelles actions à la conjecture et ainsi de réduire le nombre d'hypothèses introduites. Si l'on fait un rapprochement avec les travaux sur les interactions entre plans de (Martial, 1992), ce mécanisme peut être vu comme une technique de prise en compte des relations favorables[1] entre les actions proposées par les différents agents. En effet, la conjecture

[1] Dans la mesure où la ressource est partageable et non consommable.

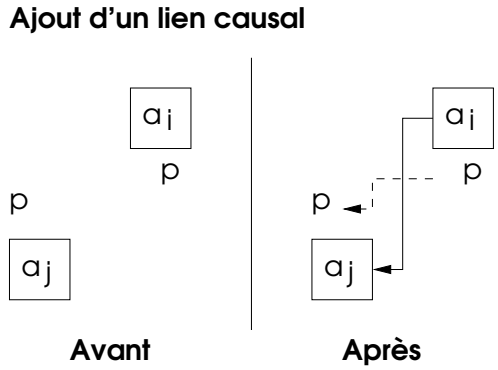


FIG. 5.5: Exemple de raffinement lorsqu'une action déjà contenue dans la conjecture permet de démontrer une hypothèse p.

construite est plus efficace puisque le raffinement conduit à la réutilisation d'une action déjà présente. On évite ainsi l'ajout d'actions redondantes au cours du processus de synthèse de plans.

La procédure de raffinement est donnée par l'algorithme 5.2. Pour chaque action a_i de la conjecture χ, l'agent teste si p peut être unifiée avec un effet produit par a_i. Dans ce cas, un raffinement par ajout de lien causal existe et un nouveau raffinement est ajouté à l'ensemble des raffinements précédemment calculés.

5.2 : Refine(p, χ)

```
1  RefSet ← est un ensemble vide de raffinements ;
2  forall a_i ∈ A do
3  |  forall q ∈ add(a_i) do
4  |  |  σ ← Unify(p, q, I) ;
5  |  |  if σ != Failure then
6  |  |  |  RefSet ← RefSet ∪ {(a_i ≺ a_j), (a_i --p--> a_j), σ} ;
7  |  |  end
8  |  end
9  end
10 return RefSet ;
```

N'oublions pas que l'état initial est défini par les effets d'une action particulière que nous avons notée a_0. Initialement, chaque agent initialise les effets de a_0 avec ses propres croyances (l'état initial global n'est pas accessible car distribué). Par conséquent, les agents ne possèdent pas tous la même action a_0. Cependant, comme a_0 est une action, le mécanisme présenté s'applique. Si un agent possède dans ses croyances initiales, *i.e.,* les effets de a_0, une proposition qui vérifie une hypothèse, alors il peut la proposer en tant que raffinement. Cette proposition doit alors être ajoutée dans les effets de a_0 des autres agents construisant ainsi l'état initial global nécessaire à l'exécution de la conjecture. Ce type de raffinement est intéressant car il

fournit un mécanisme uniforme pour partager non seulement les croyances initiales mais également les croyances déduites de l'exécution des actions.

Exemple 5.3 Considérons un exemple de conjecture représenté à la figure 5.6. Un agent transporteur propose de déplacer de la farine d'un village qu'il suppose connu v au village1. Ceci l'amène à formuler trois hypothèses : at($c1$,v), connected(v,village1) et is-available(wheat,v). Par la suite, le fermier propose de vendre du blé.

À cet instant, l'hypothèse formulée par le transporteur is-available(wheat,v) peut être raffinée par ajout d'un lien causal. En effet, en proposant de vendre du blé, le fermier a produit une action dont l'un des effets est de rendre la farine disponible au lieu village3. Pour traduire cette connaissance au sein de la conjecture, le fermier peut ajouter un lien causal $(a_i \xrightarrow{p} a_j)$ où a_i représente l'action !sell, a_j l'action !load et p la proposition is-available(wheat,village3) ($\sigma = \{v = \text{village3}\}$).

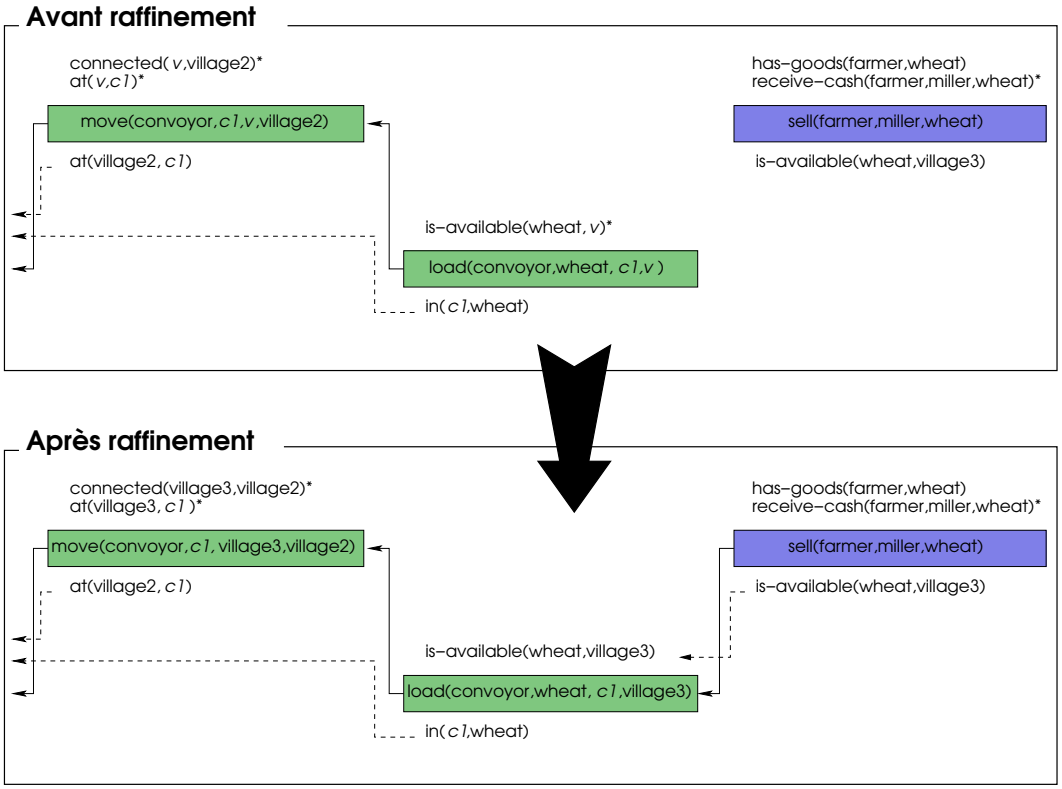


FIG. 5.6: Exemple de raffinement par ajout d'un lien causal : les hypothèses sont marquées par un astérisque.

Les raffinements par ajout d'une sous-conjecture

Le second mécanisme de raffinement consiste à ajouter une sous-conjecture pour démontrer la validité d'une hypothèse. Rappelons qu'une hypothèse p est considé-

rée comme un sous but à atteindre pour les autres agents. Le raffinement solution contient tous les éléments de la sous-conjecture produite χ', un lien causal, $(a_i \xrightarrow{p} a_j)$ permettant de spécifier quelle action a_i de χ' démontre l'hypothèse p, et une contrainte d'ordre $(a_i \prec a_j)$. La figure 5.7 montre un exemple de raffinement par ajout d'une sous-conjecture. Ce mécanisme repose sur la production d'un plan particulier pouvant lui-même contenir des hypothèses. Nous ne détaillerons pas ce mécanisme ici. Il fait l'objet du chapitre 6, *« Planifier sous hypothèses »*.

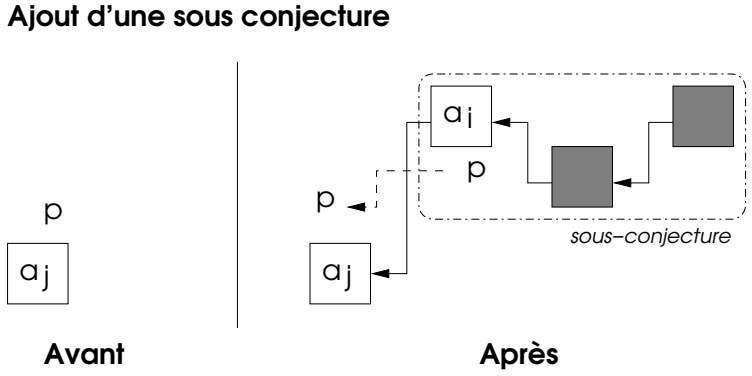


FIG. 5.7: Exemple de raffinement par ajout d'une sous-conjecture.

5.3.2 Les mécanismes de réfutation

En toute généralité, les mécanismes de réfutation cherchent à démontrer qu'une conjecture est incorrecte. Dans notre approche, une conjecture est incorrecte si elle contient des séquences d'actions non valides. Nous avons défini une réfutation (cf. définition 4.9) comme un couple $(a_k, a_i \xrightarrow{p} a_j)$ indiquant que l'action a_k à pour effet q tel que q supprime une précondition p nécessaire à l'exécution de a_j. Autrement dit, le calcul des réfutations consiste à déterminer les actions a_k qui invalident un lien causal $(a_i \xrightarrow{p} a_j)$ (cf. algorithme 5.3).

Exemple 5.4 Considérons à nouveau un extrait d'une conjecture présentée par la figure 5.8. Les actions a_i, a_j et a_k sont définies par :

- a_i = !load(conveyor,wheat,*c1*,*v*)
- a_j = !unload(conveyor,wheat,*c1*,village2)
- a_k = !unload(conveyor,wheat,*c2*,*v*)

Il existe un lien de causalité $(a_i \xrightarrow{p} a_j)$ tel que p représente le prédicat in(*c1*,*g1*) et a_k produit l'effet q, $\neg$in(*c2*,*g2*). Les seules contraintes d'instanciation de la conjecture portant sur p et q sont définies par $\mathcal{I} = \{g1 = \text{wheat}, g2 = \text{wheat}\}$. Par conséquent, p et q sont unifiables.

Remarque. 5.2 Nous ne considérons ici qu'un seul type de réfutation. Toutefois, le mécanisme de réfutation est générique et peut être étendu en s'appuyant notamment sur les travaux de (Prakken, 1997; Dung et al., 2003).

5.3 : Refute(χ)

```
1  RetSet ← est un ensemble vide de réfutations ;
2  forall a_k ∈ A do
3  |  forall (a_i -p-> a_j) ∈ C do
4  |  |  forall q ∈ add(a_k) do
5  |  |  |  σ ← Unify(¬q, p, I) ;
6  |  |  |  if σ != Failure et (a_i ≺ a_k), (a_k ≺ a_j) cohérentes avec ≺ then
7  |  |  |  |  RetSet ← RetSet ∪ {(a_k, a_i -p-> a_j)} ;
8  |  |  |  end
9  |  |  end
10 |  end
11 end
12 return RetSet ;
```

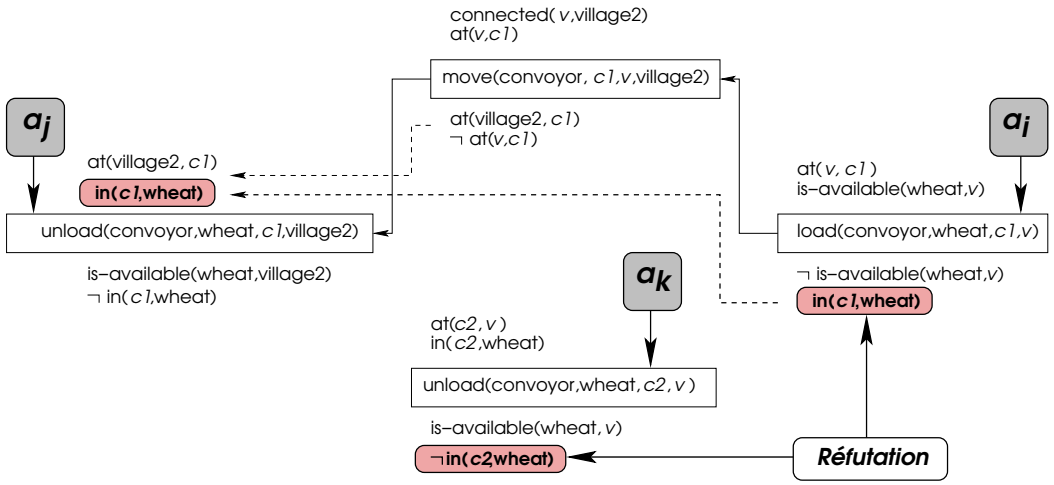


FIG. 5.8: Exemple de réfutation.

5.3.3 Les mécanismes de réparation

Les mécanismes de réparation calculent les modifications à apporter aux conjectures lorsque celles-ci ont été préalablement réfutées. Nous distinguons les réparations par ajout de contraintes d'ordre, par modification des contraintes d'instanciation et par ajout d'une conjecture.

Les réparations par ajout de contraintes d'ordre

Le premier type de réparation consiste à introduire une contrainte d'ordre entre a_k et les actions a_i et a_j. En effet, si une réfutation a été formulée à l'encontre de la conjecture, c'est parce que a_k supprime une propriété du monde nécessaire à l'exécution de a_j. Il faut donc s'assurer que a_k ne puisse pas être exécutée entre les actions a_i et a_j, *i.e.*, $(a_i \prec a_k \prec a_j)$. Pour garantir que cela ne peut pas se produire,

il faut ajouter soit une contrainte d'ordre ($a_j \prec a_k$), imposant que l'exécution de a_k soit réalisée après a_j (cf. figure 5.9 de gauche), soit une contrainte d'ordre ($a_k \prec a_i$), imposant que a_k s'exécute avant a_i (cf. figure 5.9 de droite).

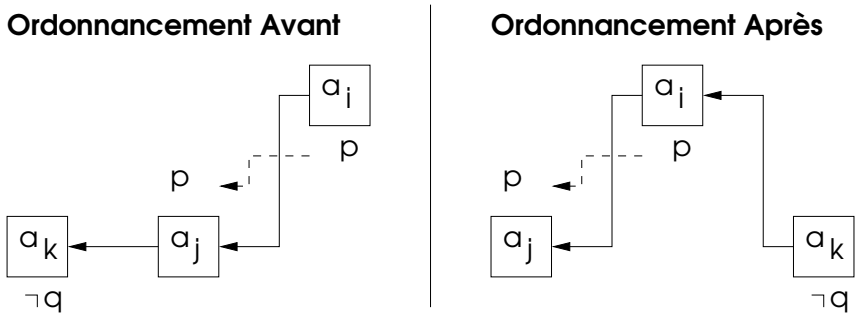


FIG. 5.9: Exemple de réparation par ajout de contraintes d'ordre. La figure de gauche ajoute la contrainte $a_j \prec a_k$ et la figure de droite $a_k \prec a_i$.

L'existence préalable de contraintes d'ordre dans la conjecture entre l'action a_k et les actions a_j et a_i contraignent les choix possibles. Le tableau 5.3 récapitule les réparations par ajout de contraintes d'ordre en fonction des contraintes qui lient a_k avec les autres actions.

Contraintes existantes sur a_k	**Contraintes à ajouter**
$\emptyset$	$a_j \prec a_k$ ou $a_k \prec a_i$
$a_k \prec a_j$	$a_k \prec a_i$
$a_i \prec a_k$	$a_j \prec a_k$
$a_i \prec a_k$ et $a_k \prec a_j$	pas de solution

TAB. 5.3: Taxonomie des réparations par ajout d'une contrainte d'ordre.

Exemple 5.5 Dans l'exemple de réfutation 5.4, aucune contrainte d'ordre n'est définie entre l'action a_k, !unload(conveyor,wheat,*c2*,*v*) et les actions a_i, !load(conveyor, wheat,*c1*,*v*), et a_j, !unload(conveyor,wheat,*c2*,*v*). Par conséquent, il existe deux réparations possibles consistant à ajouter les contraintes d'ordre suivantes :

1. !unload(conveyor,wheat,*c1*,village2) $\prec$!unload(conveyor,wheat,*c2*,*v*)
2. !unload(conveyor,wheat,*c2*,*v*) $\prec$!load(conveyor,wheat,*c1*,*v*)

La procédure calculant les réparations par ajout de contraintes d'ordre, donnée par l'algorithme 5.4, n'est qu'une transcription du tableau 5.3.

Les réparations par modification des contraintes d'instanciation

Le deuxième type de réparation possible consiste à rendre p et q non unifiables. Pour cela, il convient de modifier les contraintes d'instanciation tout en garantissant qu'elles restent cohérentes.

5.4 : Repair(ϕ, χ)

```
1  RepSet ← est un ensemble vide de réparations ;
2  if (a_k ≺ a_j) ∉ ≺ et (a_i ≺ a_k) ∉ ≺ then
3  |  RepSet ← RepSet ∪ {a_j ≺ a_k} ∪ {a_k ≺ a_i} ;
4  end
5  else if (a_k ≺ a_j) ∉ ≺ et (a_i ≺ a_k) ∈ ≺ then
6  |  RepSet ← RepSet ∪ {a_j ≺ a_k} ;
7  end
8  else if (a_k ≺ a_j) ∈ ≺ et (a_i ≺ a_k) ∉ ≺ then
9  |  RepSet ← RepSet ∪ {a_k ≺ a_i} ;
10 end
11 return RepSet ;
```

Exemple 5.6 Considérons l'exemple de réfutation de la figure 5.8. La substitution σ résultant de l'unification de p et q est la suivante :

$$\sigma = \{c1 = c2; g1 = g2; g1 = \mathsf{wheat}; g2 = \mathsf{wheat}\}$$

Deux cas sont possibles pour rendre p et q non unifiables : en ajoutant la contrainte $c1 \neq c2$ ou en ajoutant la contrainte $g1 \neq g2$. Dans le premier cas, l'ajout de la contrainte ne pose pas de problème puisque les variables *c1* et *c2* n'ont pas encore été instanciées. p et q ne peuvent plus être unifiées et nous garantissons ainsi que l'action a_k ne produira jamais comme effet $\neg q$. En revanche, dans le second cas, *g1* et *g2* ont déjà été instanciées et rendre $g1 \neq g2$ implique de modifier la valeur de l'instanciation de *g1* ou de *g2*.

Supposons que l'on veuille remplacer la contrainte $g1 = \mathsf{wheat}$ par $g1 = \mathsf{flour}$. De quelle manière la conjecture en est-elle affectée ? Étant donné qu'il existe un lien de causalité entre les actions a_i (!load) et a_j (!unload) portant sur in(*c1*,*g1*), la propagation de la contraintes $g1 = \mathsf{flour}$ au sein de la conjecture implique que a_j ne produit plus dorénavant l'effet is-available(wheat,village2) mais is-available(flour,village2). Si le but était de démontrer cette propriété du monde, a_j ne permet plus de le faire. Il faut alors supprimer a_j de la conjecture ainsi que l'ensemble des actions qui servaient de support à ses préconditions.

Au travers de cet exemple, nous voyons que ce mécanisme de réparation peut affecter de manière importante la conjecture en invalidant une partie plus ou moins grande de la conjecture déjà co-construite. Par conséquent, lorsque ce type de réparation a lieu, les agents doivent toujours considérer, dans un premier temps, celles affectant le moins la conjecture à réparer.

La procédure de réparation est donnée par l'algorithme 5.5. Pour chaque contrainte d'instanciation de χ impliquant l'une des variables de p ou de q, la procédure de réparation cherche à propager une contrainte rendant p et q non unifiables :

1. Si la contrainte est de la forme $x = y$ alors la réparation consiste à propager $x \neq y$. La fonction `Spread()` diffuse alors cette contrainte au sein de χ et retourne les modifications à apporter à χ ;
2. Sinon, la contrainte est de la forme $x = \mathsf{cst}$. Dans ce cas, l'agent essaie d'unifier p avec l'une de ses croyances pour trouver une autre instanciation pour x. Si l'unification réussie, alors la fonction `Spread()` propage les nouvelles contraintes d'instanciation et renvoie les modifications à appliquer à χ.

5.5 : Repair(ϕ, χ)

```
1  RepSet ← est un ensemble vide de réparation ;
2  forall c ∈ I impliquant une variable de p ou de q do
3  |  if c est de la forme x = y then
4  |  |  RepSet ← RepSet ∪ Spread(x ≠ y, χ) ;
5  |  end
6  |  if c est de la forme x = cst then
7  |  |  forall b ∈ beliefs(ag) do
8  |  |  |  σ ← Unify (b, p, ∅) ;
9  |  |  |  if σ ≠ Failure then
10 |  |  |  |  RepSet ← RepSet ∪ Spread(σ, χ) ;
11 |  |  |  end
12 |  |  end
13 |  end
14 end
15 return RepSet ;
```

Il est important de noter que la propagation des nouvelles contraintes au sein de la conjecture peut conduire à la suppression d'actions (cf. exemple 5.6). Regardons, au travers d'un exemple, comment la procédure `Spread()` détermine les actions à supprimer.

Exemple 5.7 Considérons la figure 5.10 qui décrit le cas d'une conjecture marquée par une réfutation et supposons que la propagation des nouvelles contraintes d'instanciation implique la suppression de l'action a_k. Les actions supportant ses préconditions doivent être également supprimées. Notons que la suppression de a_k n'entraîne pas celle de a_1 car a_1 supporte a_j.

Les réparations par ajout d'une conjecture

Étant donné que l'action a_k supprime une propriété p nécessaire à l'exécution de l'action a_j, la réparation consiste à ajouter une conjecture permettant de produire p après l'exécution de a_k. En ce sens, les réparations par ajout de conjectures peuvent être vues comme le raffinement d'une hypothèse implicite formulée par l'action a_k. En effet, en proposant l'action a_k, un agent suppose qu'un autre agent sera capable

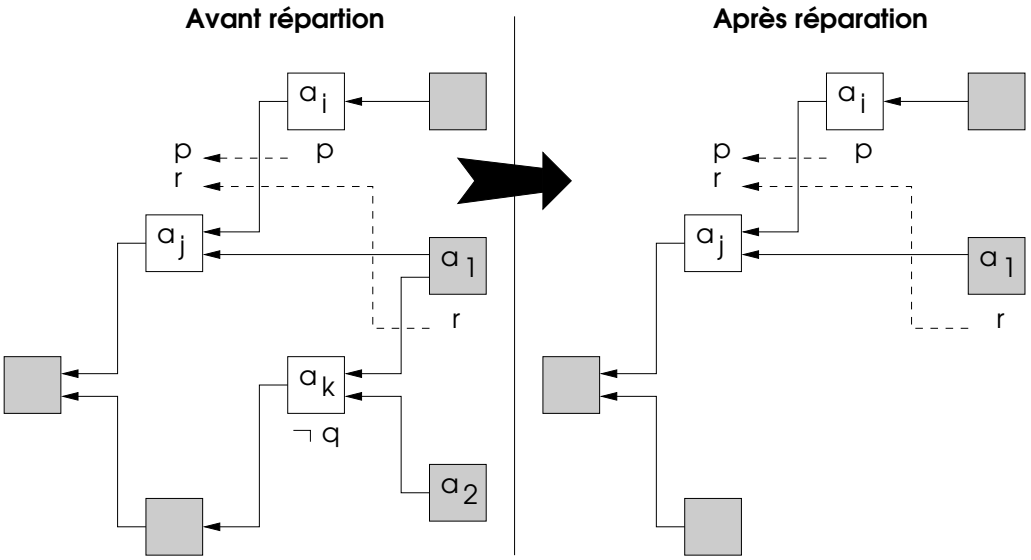


FIG. 5.10: Exemple de réparation qui conduit à la suppression d'actions.

de recréer la propriété q. Par conséquent, ce type de réparation qui repose sur la production de raffinements vérifiant p (cf. chapitre 6) participe à la mise en œuvre de la coopération entre les agents.

Exemple 5.8 Considérons l'exemple donné par la figure 5.11. L'action a_k réfute le lien causal $(a_i \xrightarrow{p} a_j)$ car l'effet p est unifiable avec q. L'ajout de la conjecture (ne contenant que l'action a_l) produit l'effet q après sa suppression par a_k. Par conséquent, q est vérifiée dans l'état précédant a_j (a_j peut être exécutée) et a_k ne réfute plus $(a_i \xrightarrow{p} a_j)$.

Remarque. 5.3 Ce mécanisme de réparation complète celui par ajout de contraintes d'ordre. En effet, nous avons souligné (cf. tableau 5.3) que lorsque les contraintes d'ordre imposaient que a_k soit exécutée entre les actions a_j et a_i, aucun ordonnancement possible de ces actions ne permettait de résoudre la réfutation. En produisant une conjecture vérifiant la propriété q, le mécanisme de réparation par ajout de conjecture pallie à cette limitation.

Conclusion

Dans ce chapitre, nous avons présenté la synthèse dialectique de plans comme un processus collectif de délibération permettant à un groupe d'agents de raisonner sur un but à atteindre. La description de ce processus de délibération a permis de mettre en évidence plusieurs types de mécanismes. Nous avons, tout d'abord, introduit les règles du raisonnement par conjecture - réfutation. Puis, nous avons détaillé

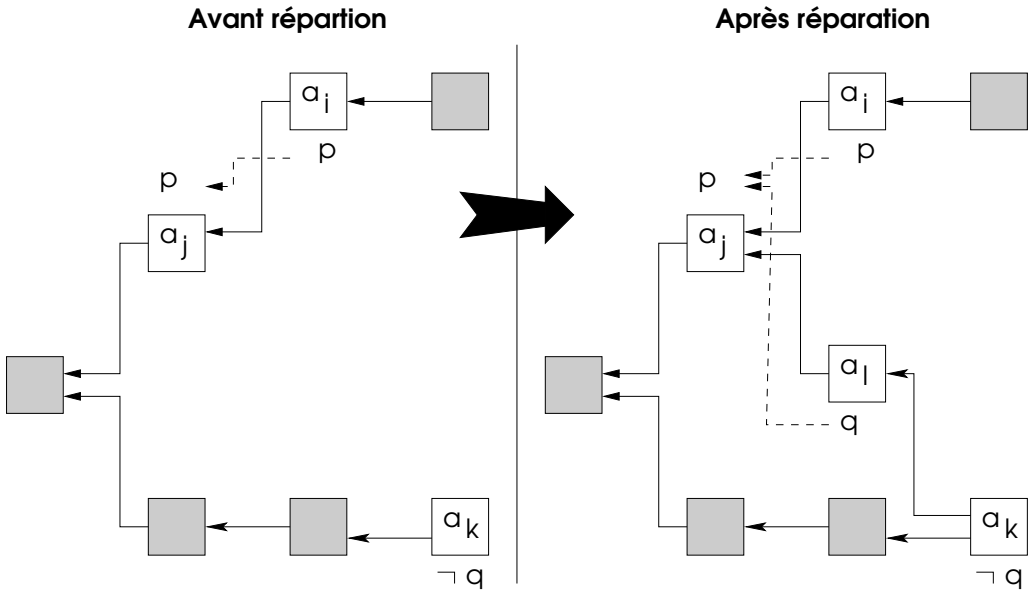


FIG. 5.11: Exemple de réparation par ajout d'une conjecture.

les mécanismes qui guident les interactions des agents : les agents appliquent itérativement des opérateurs dialectiques de raffinement, de réfutation et de réparation tant qu'un plan-solution n'a pas été trouvé. Finalement, nous avons formalisé les mécanismes de raffinement, de réfutation et de réparation sur lesquels s'appuient les agents pour produire un plan-solution.

Une caractéristique importante du raisonnement par conjecture - réfutation, outre la capacité des agents à débattre et à interagir, repose sur leur capacité à raisonner en s'appuyant sur des hypothèses par opposition au raisonnement se fondant sur des propriétés certaines du monde. C'est de cette caractéristique que naît la coopération des agents. Ce point particulier a été laissé en suspend : comment les agents sont-ils capables de produire les sous-conjectures, *i.e.,* des plans sous hypothèses, pour démontrer qu'un but est atteignable ? En effet, les algorithmes classiques de planification ne permettent pas la génération de plans dans lesquels certaines propriétés du monde ne sont pas encore vérifiées. Dans quelle mesure ces algorithmes peuvent être adaptés pour répondre à cette problématique ? Et quelles sont les hypothèses les plus « raisonnables » que peuvent formuler les agents ? Le chapitre suivant propose de répondre à ces questions.

Chapitre 6

Planifier sous hypothèses

Introduction

La synthèse dialectique de plans repose sur la capacité des agents à produire des conjectures, *i.e.,* des plans qui peuvent être exécutés si certaines propriétés du monde sont vérifiées. C'est de cette capacité à formuler les propriétés du monde manquantes pour l'exécution d'un plan que naît la collaboration entre les agents. En effet, comme nous l'avons déjà souligné, les propriétés manquantes apparaissent alors comme des sous buts à atteindre pour les autres agents. Les algorithmes classiques de planification n'autorise pas la production de plans avec de telles caractéristiques. Quelles sont alors les adaptations à leur apporter pour répondre à cette problématique et quelles sont les hypothèses les plus « raisonnables » que peuvent formuler les agents ?

Dans ce chapitre, nous présentons le modèle de planification sous hypothèses qui produit les raffinements nécessaires à la démonstration des propriétés du monde supposées par les autres agents. Puis, nous présenterons l'algorithme de planification proprement dit ainsi que ses heuristiques de recherche.

6.1 Le modèle de la planification sous hypothèses

6.1.1 La problématique

Le modèle de planification sous hypothèses, sur lequel est fondé la production des raffinements, repose sur une technique de planification dite hiérarchique. Cette technique s'appuie, comme les modèles classiques de planification, sur la recherche dans un espace d'états. Le passage d'un état à l'autre s'effectue par l'application d'une action. Cependant, la planification hiérarchique diffère dans la manière de produire les plans.

En planification hiérarchique, l'objectif n'est pas d'atteindre un ensemble de buts mais de réaliser un ensemble d'*actions complexes*. Le planificateur prend en entrée un ensemble d'opérateurs, de la même façon que les planificateurs classiques

STRIPS, mais également un ensemble de *méthodes*. Chaque méthode décrit comment une action complexe peut être décomposée en un ensemble de sous-actions. La production d'un plan peut alors s'exprimer comme la décomposition récursive d'actions complexes en sous-actions jusqu'à ce que toutes les actions soient des actions *primitives*, *i.e.,* pouvant être exécutées en appliquant un opérateur de transformation (cf. définition 4.1).

Maintenant que nous avons présenté succintement les mécanismes de la planification hiérarchique, intéressons nous aux adaptations à leur apporter pour répondre à notre problématique. La planification hiérarchique emploie un modèle d'actions qui suppose que toutes les préconditions des opérateurs o sont satisfaites dans l'état s précédant son application. Par définition, ce modèle ne permet pas le déclenchement d'opérateurs dont certaines préconditions ne sont pas vérifiées. Or, dans le cadre de la planification sous hypothèses, ce sont justement ces préconditions particulières qu'il est intéressant de considérer, car elles expriment les propriétés du monde manquantes. Pour produire des conjectures, il est donc nécessaire de relaxer ce modèle d'actions classiquement utilisé en planification. Nous considérons, dans ce chapitre, que les opérateurs[1] sont toujours applicables à partir d'un état de croyances s. Nous présenterons plus en détails les mécanismes calculant les hypothèses nécessaires au déclenchement des opérateurs dans le paragraphe 6.1.2.

En posant ce modèle, nous nous exposons à une explosion combinatoire. En pratique, cette combinatoire peut être maîtrisée. En effet, il n'est pas intéressant de considérer toutes les conjectures car seuls les raffinements formulant un petit nombre d'hypothèses ont une chance d'aboutir à un plan-solution. Par conséquent, planifier sous hypothèses consiste à construire un plan tout en minimisant le nombre d'hypothèses introduites. Cette minimisation présente un autre avantage dans un contexte multi-agent. En limitant le nombre d'hypothèses, nous réduisons le nombre d'opérateurs de raffinement à appliquer et, par effet de bord, le coefficient de branchement de l'espace de recherche au niveau global. Ceci améliore de manière significative les performances globales.

En outre, il n'est pas nécessairement utile aux agents de formuler des hypothèses sur toutes les propriétés du monde. Supposons qu'un agent veuille déplacer une charrette d'un village à un autre. Pour s'exécuter, les deux villages doivent être connectés par une route. Est-il « raisonnable » de faire l'hypothèse que les deux villages sont connectés ? Ceci dépend de la capacité des autres agents à proposer un raffinement pour démontrer cette propriété. Dans l'absolu, il n'est pas possible de dire si cette hypothèse pourra être ou non raffinée. En revanche, il est possible de définir pour un agent donné les propriétés du monde sur lesquelles il peut formuler des hypothèses. Ceci limite l'explosion combinatoire et participe à une description plus fine du raisonnement hypothétique que peuvent produire les agents.

[1] Nous utilisons le termes d'opérateurs au sens large, *i.e.,* opérateurs ou méthodes. Les méthodes sont formellement définies par la définition 6.1.

6.1.2 Les opérateurs et les méthodes

Nous ajoutons à notre langage un nouveau type d'opérateurs de transformation qui décrit des actions complexes.

Définition. 6.1 (Méthode)
Une *méthode* est un triplet

$$m = (name(m), precond(m), reduction(m))$$

- $name(m)$ est une expression de la forme $n(x_1, \ldots, x_k)$, où n représente le nom de la méthode et $x_1, \ldots, x_k$ ses paramètres.
- $precond(m)$ représente les préconditions (*i.e.,* un ensemble de prédicats) devant être vérifiées dans l'état des croyances de l'agent pour que m soit appliquée.
- $reduction(m)$ définit la séquence d'opérateurs ou de méthodes à accomplir pour réaliser m.

Remarque. 6.1 Dans la suite de ce manuscrit, les noms des méthodes ne seront pas précédés par le symbole !. Par exemple, la méthode produce-wheat(f,q) caractérise un ensemble d'actions complexes permettant à un agent fermier f de produire une quantité q de blé.

L'ajout de ce nouveau type d'opérateurs nous oblige à enrichir également la définition que nous avions donnée d'un agent (cf. définition 4.2) :

Définition. 6.2 (Agent)
Un *agent* est un quadruplet

$$ag = (name(ag), operators(ag), methods(ag), beliefs(ag))$$

- $name(ag)$ est le nom de l'agent ;
- $operators(ag)$ est un ensemble d'opérateurs, *i.e.,* les actions qu'un agent peut appliquer sur le monde ;
- $methods(ag)$ est un ensemble de méthodes, *i.e.,* les recettes qu'un agent peut appliquer pour réaliser une action complexe ;
- $beliefs(ag)$ définit un ensemble de propositions définissant les propriétés du monde connues par l'agent.

Exemple 6.1 Considérons un extrait des opérateurs et des méthodes de l'agent fermier.

;; Le fermier f plante une quantité de blé q.
!sow(f,q)
precond: sowed-wheat(f,q_{sw})
add: sowed-wheat(f,(q_{sw}+q))
del: sowed-wheat(f,q_{sw})

;; Le fermier f récolte une quantité de blé q.
!harvest(f,q)
precond: sowed-wheat(f,q_{sw}), has-goods(f,wheat,q_w)
add: sowed-wheat(f,($q_{sw}-q$)), has-goods(f,wheat,(q_w+q))
del: sowed-wheat(f,q_{sw}), has-goods(f,wheat,q_w)

;; Le fermier f vend à l'agent a la quantité q de blé.
!sell(f,a,wheat,q)
precond: has-goods(f,wheat,q_w), agent-village(v),
price(f,wheat,p), has-cash(f,q_c)
add: has-goods(f,wheat,(q_w-q)), has-cash(f,(q_c+q)),
is-available(wheat,v)
del: has-goods(f,wheat,q_w,v), has-cash(f,q_c)

;; Le fermier f récolte une quantité de blé q.
harvest(f,q)
precond: agent-village(v),
$\sim$has-tool(f,scythe), $\sim$is-available(scythe,v),
$\sim$sowed-wheat(f,q_{sw}), ($q \leq q_{sw}$)
reduction: !harvest-wheat(f,q)

;; Le fermier f produit une quantité de blé q.
produce(f,q)
precond: has-goods(f,wheat,q_w), sowed-wheat(f,q_{sw})
reduction: !sow-wheat(f,($q-(q_w+q_{sw})$)), harvest(f,($q-(q_w+q_{sw})$))

;; Le fermier f vend une quantité q de blé à l'agent a.
sell(f,a,wheat,q)
precond: has-goods(f,wheat,q_w), ($q_w \geq q$),
price(f,wheat,p), $\sim$receive-cash(f,a,($q*p$),wheat)
reduction: !sell(f,a,wheat,($q*p$))

;; Le fermier f vend une quantité q de blé à l'agent a.
sell(f,a,wheat,q)
precond: has-goods(f,wheat,q_w), ($q_w < q$),
price(f,wheat,p), $\sim$receive-cash(f,a,($q*p$),wheat)
reduction: produce(f,q), !sell(f,a,wheat,($q*p$))

Cet exemple nous amène également à introduire un certain nombre de compléments que nous apportons à notre langage initial. Tout d'abord, les préconditions précédées par le symbole $\sim$ définissent les propriétés du monde sur lesquelles les agents peuvent formuler des hypothèses. Nous présenterons dans le paragraphe 6.1.3 comment ces propriétés hypothétiques sont calculées.

Nous ajoutons des symboles de fonctions afin de manipuler des expressions numériques sur les entiers, par l'intermédiaire des symboles $+$, $-$, $*$ et $/$. De plus, nous étendons notre formalisme aux fonctions booléennes $<$, $>$, $\leq$, $\geq$, min et max.

Finalement, nous augmentons notre langage en introduisant le concept d'*axiome*. La mise en œuvre des mécanismes d'inférence s'appuie sur un démonstrateur de théorème implémenté comme une sous-routine de la procédure de planification présentée au paragraphe 6.2. Il est important de noter que nous nous limitons aux axiomes sous forme de clauses de Horn. La syntaxe complète utilisée et implémentée est donnée en annexe A.

6.1.3 La génération des hypothèses

La génération des hypothèses repose sur le fait qu'il n'est pas nécessaire que l'ensemble des préconditions soient vérifiées dans l'état initial défini par les croyances d'un agent pour qu'un opérateur ou une méthode soit applicable. Les préconditions non vérifiées forment alors un ensemble de propriétés du monde que nous nommons des *hypothèses* afin de les distinguer des croyances de l'agent.

Plus formellement, le calcul des hypothèses repose sur l'identification de l'ensemble des substitutions possibles pour appliquer un opérateur ou une méthode (cf. algorithme 6.1). Soit *precond* les préconditions d'un opérateur ou d'une méthode, s les croyances de l'agent et σ une substitution définissant un ensemble de contraintes d'instanciation, l'algorithme (algorithme 6.1) calcule les substitutions possibles à partir de s. Si *precond* est vide la procédure renvoie l'ensemble vide (ligne 2). Sinon, l'algorithme cherche récursivement à déterminer les substitutions possibles pour unifier *precond* avec s.

Dans ce cas, l'algorithme dépile alors la première précondition p contenue dans *precond* (ligne 5) et cherche à unifier p pour chaque croyance contenue dans s (ligne 7). Si l'unification est possible alors il existe une substitution qui unifie p et une croyance p' de s. L'algorithme est alors appliqué récursivement sur les préconditions non encore unifiées $\mathcal{R}$, en prenant en compte les variables précédemment instanciées (ligne 16).

Remarque. 6.2 Les substitutions résultantes de l'algorithme peuvent être partielles (*i.e.,* ne pas contenir de couple $(x_i = t_i)$ pour chaque variable x_i contenue dans les préconditions d'une action). Par conséquent, certaines préconditions peuvent être partiellement instanciées après l'application de la substitution. Un agent peut donc produire des raffinements partiellement instanciées.

Finalement, la procédure qui produit les hypothèses nécessaires à l'application d'une action est décrite par l'algorithme (algorithme 6.2). Soient *precond* les préconditions d'un opérateur ou d'une méthode, une substitution σ et un état s, l'algorithme applique pour chaque précondition $p \in$ *precond* la substitution σ et teste si $\sigma(p)$ appartient à s (ligne 3). Si $\sigma(p)$ appartient à s alors $\sigma(p)$ n'est pas une hypothèse (ligne 5). On se ramène alors dans le cas de la planification classique. Sinon $\sigma(p)$ est une hypothèse (ligne 8).

6.1 : FindSubstitutions($precond, s, \sigma$)

```
1  result ← un ensemble vide de substitutions;
2  if precond est vide then
3  |  return {σ} ;
4  end
5  p ← le premier prédicat de precond;
6  R ← le reste des prédicats de precond;
7  foreach prédicat p' ∈ s do
8  |  θ ← Unify(p, p', σ);
9  |  if θ == Failure then
10 |  |  if p est une propriété hypothétique then
11 |  |  |  continue ;
12 |  |  else
13 |  |  |  return Failure ;
14 |  |  end
15 |  else
16 |  |  Σ ← FindSubstitutions(R, s, σ(θ));
17 |  |  foreach substitutions γ ∈ Σ do
18 |  |  |  ajouter la substitution γ à result ;
19 |  |  end
20 |  end
21 end
22 return result ;
```

6.2 : Assumptions($precond, \sigma, s$)

```
1  result ← un ensemble vide d'hypothèses ;
2  foreach précondition p ∈ precond do
3  |  if σ(p) ∈ s then
4  |  |  continue ;
5  |  else
6  |  |  ajouter p à result ;
7  |  end
8  end
9  return result ;
```

Exemple 6.2 Supposons que l'état des croyances de l'agent fermier soit décrit par l'état s_0 suivant :

$$s_0 = \left\{ \begin{array}{l} \mathsf{agent\text{-}village}(\mathsf{village3}), \\ \mathsf{sowed\text{-}wheat}(\mathsf{farmer}, 3), \\ \mathsf{has\text{-}tool}(\mathsf{farmer}, \mathsf{scythe}), \\ \mathsf{has\text{-}goods}(\mathsf{farmer}, \mathsf{wheat}, 0) \end{array} \right\}$$

et que l'on veuille appliquer la méthode :

harvest(farmer,2)
precond: agent-village(village3),
∼has-tool(farmer,scythe), ∼is-available(scythe,village3),
∼sowed-wheat(farmer,3), $(2 \leq 3)$
reduction: !harvest-wheat(farmer,2)

Aucune proposition n'indique dans s_0 que la faux soit disponible dans son village. Dans ce cas, le déclenchement de la méthode harvest nécessite de formuler une hypothèse portant sur la précondition is-available(scythe,village3). Cette précondition est précédée par le symbole $\sim$. L'agent peut donc raisonner de manière hypothétique sur cette propriété et modifier l'état s_0 en conséquence afin d'appliquer la méthode :

$$s'_0 = \left\{ \begin{array}{l} \text{agent-village(village3)}, \\ \text{sowed-wheat(farmer, 3)}, \\ \text{has-tool(farmer, scythe)}, \\ \text{has-goods(farmer, wheat, 0)}, \\ \text{is-available(scythe, village3)}* \end{array} \right\}$$

L'hypothèses is-available(scythe,village3) devra alors être raffinée par un autre agent et l'opérateur !harvest-wheat(farmer,2) devra être réalisé.

Cet exemple met en évidence un premier type d'hypothèses que nous nommons des *suppositions*. Les suppositions définissent des propriétés absentes des croyances d'un agent mais qui doivent être vérifiées pour qu'une action puisse être réalisée. Ces propriétés sont considérées comme inconnues et non comme fausses.

Exemple 6.3 Supposons maintenant que l'état initial s_0 soit décrit de la manière suivante :

$$s_0 = \left\{ \begin{array}{l} \text{agent-village(village3)}, \\ \text{sowed-wheat(farmer, 3)}, \\ \text{has-tool(farmer, scythe)}, \\ \text{has-goods(farmer, wheat, 0)}, \\ \neg\text{is-available(scythe, village3)} \end{array} \right\}$$

et que l'on veuille appliquer la méthode harvest(farmer,2). L'agent fermier sait cette fois que la faux n'est pas disponible dans son village. Dans ce cas, l'hypothèse is-available(scythe,village3) est en contradiction avec ses croyances. La propriété supposée est ajoutée à l'ensemble des hypothèses formulées par l'agent et sa négation retirée de l'état courant des croyances. L'état s'_0 résultant est le suivant :

$$s'_0 = \left\{ \begin{array}{l} \text{agent-village(village3)}, \\ \text{sowed-wheat(farmer, 3)}, \\ \text{has-tool(farmer, scythe)}, \\ \text{has-goods(farmer, wheat, 0)}, \\ \text{is-available(scythe, village3)}* \end{array} \right\}$$

Ce type d'hypothèses est appelé une *dénégation*. Dans ce cas l'agent raisonne par contradiction et fait le pari que les autres agents seront capables d'atteindre ce but. La distinction entre ces deux types d'hypothèses est intéressante. Supposons qu'un agent puisse formuler une hypothèse sur la nature du temps, *i.e.,* beau ou pluvieux. En l'absence de croyances, l'agent peut raisonnablement conjecturer qu'il fait beau. En revanche, imager qu'il fait beau sachant pertinemment que le temps est pluvieux n'a pas de sens. En effet, quel agent sera capable de modifier cette propriété ? La résolution de ce problème passe par la formalisation d'une taxonomie des hypothèses, afin que le concepteur de l'agent puisse préciser plus finement les propriétés du monde sur lesquelles l'agent peut raisonner.

Considérons maintenant le dernier type d'hypothèses que nous appelons *les violations de contraintes*. Il caractérise des contraintes absentes ou contradictoires avec les croyances d'un agent. Les contraintes supposées sont alors ajoutées à l'ensemble des hypothèses formulées par l'agent. Cette classe d'hypothèses permet aux agents de raisonner sur des quantités de ressources.

Exemple 6.4 Supposons maintenant que l'état initial s_0 soit décrit de la manière suivante :

$$s_0 = \left\{ \begin{array}{l} \mathsf{agent\text{-}village(village3)}, \\ \mathsf{sowed\text{-}wheat(farmer, 1)}, \\ \mathsf{has\text{-}tool(farmer, scythe)}, \\ \mathsf{has\text{-}goods(farmer, wheat, 0)}, \\ \mathsf{is\text{-}available(scythe, village3)} \end{array} \right\}$$

et que l'on veuille toujours appliquer la méthode harvest(farmer,2). L'application de cette méthode nécessite que l'agent ait planté une quantité suffisante de blé. Or, dans l'état s_0 la quantité de blé déjà plantée n'est que de 1. Il est donc nécessaire de formuler l'hypothèse sowed-wheat(2). L'état s'_0 dans lequel peut être réalisé la méthode peut s'écrire :

$$s'_0 = \left\{ \begin{array}{l} \mathsf{agent\text{-}village(village3)}, \\ \mathsf{sowed\text{-}wheat(farmer, 2)}*, \\ \mathsf{has\text{-}tool(farmer, scythe)}, \\ \mathsf{has\text{-}goods(farmer, wheat, 0)}, \\ \mathsf{is\text{-}available(scythe, village3)} \end{array} \right\}$$

6.2 La production des raffinements

L'algorithme mis en œuvre pour produire les raffinements, *i.e.,* les sous-conjecture démontrant la validité d'une ou plusieurs hypothèses, repose sur les travaux de (Nau et al., 2003).

6.2.1 Le principe

Le but à atteindre n'est plus un ensemble de propriétés du monde qui doivent être vérifiées mais une séquence d'actions primitives ou complexes à réaliser à partir de l'état initial de croyance des agents. Pour passer de la représentation d'un but défini comme un ensemble de propositions à celle d'une actions à réaliser, il suffit de décrire une méthode particulière qui prend en paramètre le but à atteindre.

Exemple 6.5 La méthode achieve, donnée ci-dessous, permet au fermier de démontrer les propriétés du monde représentées par le prédicat has-good(a,wheat,q) :

:achieve(has-goods(a,wheat,q))
precond: agent-name(f)
reduction: sell-wheat(f,q,a)

Ceci nous conduit à généraliser la notion de problème de planification précédemment introduite (cf. § 4.2.3) :

Définition. 6.3 (Problème de planification)
Un *problème de planification* pour un domaine $\mathcal{D}$ est un couple

$$\mathcal{P} = (s_0, \mathcal{O}, \mathcal{M}, \langle \alpha_0, \ldots, \alpha_n \rangle)$$

- s_0 représentent l'union des croyances des agents de $\mathcal{D}$;
- $\mathcal{O}$ et $\mathcal{M}$ définissent l'union des opérateurs et des méthodes des agents de $\mathcal{D}$;
- $\langle \alpha_0, \ldots, \alpha_n \rangle$ est une séquence d'actions primitives ou complexes devant être réalisées.

La définition d'un raffinement peut alors être généralisée de la manière suivante :

Définition. 6.4 (Raffinement)
Soit un problème de planification $\mathcal{P} = (s_0, \mathcal{O}, \mathcal{M}, \langle \alpha_0, \ldots, \alpha_n \rangle)$. Une conjecture χ est un raffinement de $\mathcal{P}$ si toutes les linéarisations $\lambda \in \mathit{completion}(\chi)$ raffinent $\mathcal{P}$. Soit $\lambda = \langle a_0, \ldots, a_k \rangle$ une séquence d'actions primitives, λ raffine $\mathcal{P}$ si l'une des conditions suivantes est vérifiée :

Cas 1 : la séquence d'actions à réaliser est vide alors λ est la séquence vide ;

Cas 2 : α_0 est primitive : α_0 est identique à a_0, α_0 est applicable à partir de s_0 et $\lambda = \langle a_1, \ldots, a_k \rangle$ raffine $(s_1, \mathcal{O}, \mathcal{M}, \langle \alpha_1, \ldots, \alpha_n \rangle)$;

Cas 3 : α_0 est complexe : il existe une réduction $\langle r_1, \ldots, r_j \rangle$ applicable pour réaliser α_0 à partir de s_0 et λ raffine $(s'_0, \mathcal{O}, \mathcal{M}, \langle r_1, \ldots, r_j, \alpha_1, \ldots, \alpha_n \rangle)$ avec $s'_0 = s_0 \cup \mathit{assump}(\alpha_0)$.

Remarque. 6.3 Un raffinement ne contient pas de réfutation. Cependant, une action du raffinement peut réfuter un lien causal de la conjecture globale.

Exemple 6.6 Supposons que l'agent fermier veuille raffiner l'hypothèse has-goods(miller,wheat,3) en appliquant la méthode achieve décrite à l'exemple 6.5. L'état initial de ses croyances est défini par :

$$\mathit{beliefs}(\mathsf{farmer}) = \left\{ \begin{array}{l} \mathsf{agent\text{-}name(famer)}, \\ \mathsf{agent\text{-}village(village3)}, \\ \mathsf{sowed\text{-}wheat(farmer, 2)}, \\ \mathsf{has\text{-}goods(farmer, wheat, 0)}, \\ \mathsf{has\text{-}cash(farmer, 2)}, \\ \mathsf{price(farmer, wheat, 1)} \end{array} \right\}$$

Un raffinement possible peut être représenté par un arbre (cf. figure 6.1). Les actions grisées définissent des actions primitives, les flèches verticales les sous-actions des méthodes, et les flèches horizontales les contraintes d'ordre entre les sous-actions. Finalement, les hypothèses sont mentionnées au-dessus des actions.

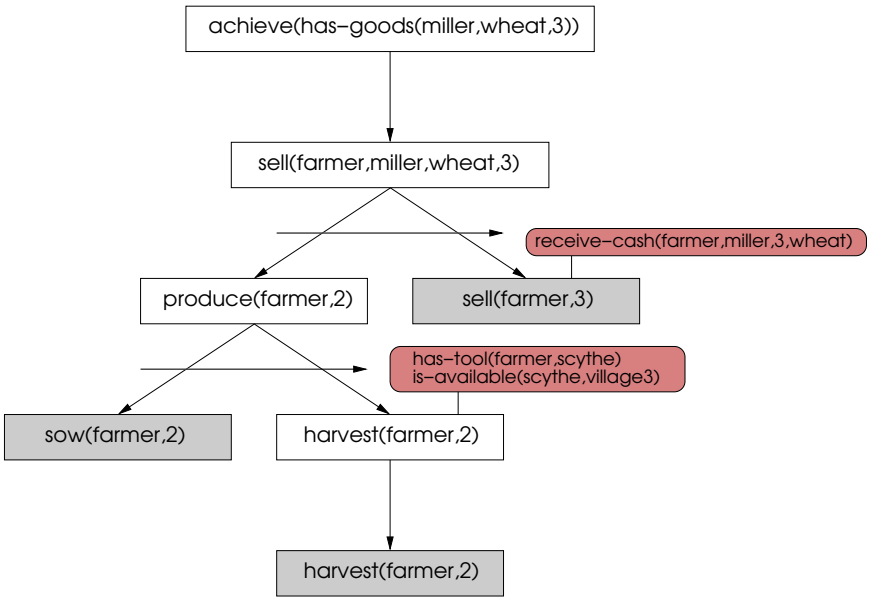


FIG. 6.1: Représentation d'un raffinement sous forme d'arbre.

6.2.2 L'algorithme

L'algorithme de planification sous hypothèses que nous proposons s'appuie sur une recherche dans l'espace d'états. Cet espace d'états est stocké dans un arbre appelé *arbre de raffinements*. Chaque nœud de l'arbre représente un état du monde atteignable après l'application d'une action. Un nœud n est défini par le tuple $(s, \langle \alpha_0, \ldots, \alpha_n \rangle)$ tel que s est un état du monde et $\langle \alpha_0, \ldots, \alpha_n \rangle$ la séquences d'actions restant à réaliser à cette étape de la décompostion. Les arêtes représentent les transitions possibles entre les différents états du monde. Une transition se caractérise par

l'opérateur ou la méthode appliqué (accompagné de ses contraintes d'instanciation) ainsi que les éventuelles hypothèses nécessaires à son exécution.

La procédure qui produit les raffinements se découpe en deux phases : l'expansion de l'arbre de raffinements (détaillée dans la suite de ce paragraphe) et l'extraction de la conjecture qui consiste à parcourir une branche de l'arbre d'une feuille solution au nœud racine en initialisant les actions, les contraintes d'ordre, les contraintes d'instanciation et les liens causaux du raffinement.

La construction de l'arbre de raffinements consiste à décomposer récursivement les actions complexes contenues dans les nœuds de l'arbre jusqu'à ce qu'une feuille soit produite, *i.e.,* un nœud possédant une séquence d'actions vide. Cette feuille représente l'état du monde qui sera atteint après l'exécution du raffinement.

L'algorithme d'expansion de l'arbre de raffinements s'appuie sur la définition 6.4 (cf. algorithme 6.3). Initialement, la procédure d'expansion de l'arbre est exécutée avec le nœud représentant l'état initial des croyances de l'agent s_0 et la séquence des actions à réaliser $\langle \alpha_0, \ldots, \alpha_n \rangle$. La procédure commence par tester si $\langle \alpha_0, \ldots, \alpha_n \rangle$ est la séquence vide (ligne 2). Dans ce cas, une feuille de l'arbre est atteinte et le nœud n est retourné comme nœud solution (cf. cas 1 définition 6.4). Sinon la procédure essaie de réaliser la première action α_0. Deux cas sont à envisager selon que α_0 est :

Une action primitive. Pour chaque opérateur contenu dans $\mathcal{O}$, la procédure teste s'il peut réaliser l'action α_0 (ligne 3). Si c'est le cas, la procédure calcule les substitutions unifiant les préconditions de l'opérateur avec l'état courant s contenu dans n (ligne 7). Finalement, pour chaque substitution σ, l'algorithme ajoute un nœud fils au nœud n (cf. cas 2 définition 6.4).

Une action complexe. La procédure repose sur le même principe que pour une action primitive. Cependant elle teste cette fois, non plus les opérateurs mais les méthodes qui permettent de réaliser α_0 et qui sont applicables dans s (ligne 17). Pour chacune de ces méthodes un nouveau nœud fils est ajouté au nœud n (cf. cas 3 définition 6.4).

Finalement, la procédure choisit de manière non déterministe[2] un nœud n_c parmi les nœuds fils de n (ligne 32) et s'appelle récursivement avec comme paramètre le nouveau nœud sélectionné.

L'algorithme de planification sous hypothèses proposé dans ce paragraphe est complet et correct. Ceci peut être démontré à partir de la définition 6.4.

Proposition. 6.1 (Correction) Soit une trace non déterministe $\lambda = \langle a_0, \ldots, a_k \rangle$. Si `ExpandTree`$(n, \mathcal{O}, \mathcal{M})$ retourne un nœud $(s_k, \langle\rangle)$ alors λ raffine $\mathcal{P} = (s_0, \mathcal{O}, \mathcal{M}, \langle \alpha_0, \ldots, \alpha_n \rangle)$.

Preuve. 6.1 Prouvons cette proposition par induction sur m le nombre d'appels à la procédure `ExpandTree` :

[2]Nous présenterons dans le paragraphe 6.2.3 quelles heuristiques guident la recherche.

6.3 : ExpandTree(n, $\mathcal{O}$, $\mathcal{M}$)

```
1  Soit n = (s, ⟨α_0, ..., α_n⟩) ;
2  if ⟨α_0, ..., α_n⟩ est la séquence vide then return n ;
3  else if α_0 est primitive then
4      foreach opérateurs o ∈ O do
5          θ ← Unify(name(α_0), name(o)) ;
6          if θ ≠ Failure then
7              Σ ← FindSubstitutions(precond(o), s, θ) ;
8              if Σ est vide then return Failure ;
9              foreach substitution σ ∈ Σ do
10                 H ← Assumptions(precond(o), σ, s) ;
11                 ajouter un fils à n avec a_0 = σ(o) tel que
                   (((s ∪ H) − del(a_0)) ∪ add(a_0)), ⟨α_1, ..., α_n⟩) ;
12             end
13         else
14             return Failure ;
15         end
16     end
17 else if α_0 est composée then
18     foreach méthode m ∈ M do
19         θ ← Unify(name(α_0), name(m)) ;
20         if θ ≠ Failure then
21             Σ ← FindSubstitutions(precond(m), s, θ) ;
22             if Σ est vide then return Failure ;
23             foreach substitution σ ∈ Σ do
24                 H ← Assumptions(precond(m), σ, s) ;
25                 ajouter un fils à n avec a_0 = σ(m) tel que
                   ((s ∪ H),Append(reduction(a_0), ⟨α_1, ..., α_n⟩)) ;
26             end
27         else
28             return Failure ;
29         end
30     end
31 end
32 Choix non déterministe d'un nœud n_c parmi les fils de n ;
33 ExpandTree(n_c, O, M) ;
```

Cas de base ($m = 1$) *:* il n'y a qu'un seul appel à `ExpandTree`. Par conséquent, la procédure s'arrête ligne 2 en renvoyant le nœud $(s_0, \langle\rangle)$. Le nœud retourné est le nœud racine, $\lambda = \langle\rangle$ et λ raffine $\mathcal{P} = (s_0, \mathcal{O}, \mathcal{M}, \langle\rangle)$.

Induction : soit $m > 1$. Supposons que la proposition est vérifiée pour $m < m'$. Deux cas sont à envisager :

Cas 1 : α_0 est primitive. Soit α_0 la première action de $n = (s_0, \langle\alpha_0, \ldots, \alpha_n\rangle)$ et $\lambda = \langle a_1, \ldots, a_k\rangle$ la trace produite par la procédure à la ligne

33 lancée avec le nœud $n_c = (s_1, \langle \alpha_1, \ldots, \alpha_n \rangle)$ et $s_1 = ((s_0 \cup \mathit{assump}(\alpha_0)) - \mathit{del}(\alpha_0)) \cup \mathit{add}(\alpha_0)$. Par hypothèse d'induction, λ raffine $\mathcal{P} = (s_1, \mathcal{O}, \mathcal{M}, \langle \alpha_1, \ldots, \alpha_n \rangle)$. Or, α_0 est primitive et réalise a_0 à partir de s_0. Donc, d'après le cas 2 de la définition 6.4, $\lambda = \langle a_0, \ldots, a_k \rangle$ raffine $\mathcal{P} = (s_0, \mathcal{O}, \mathcal{M}, \langle \alpha_0, \ldots, \alpha_n \rangle)$.

Cas 2 : α_0 est complexe. Soit α_0 la première action de $n = (s_0, \langle \alpha_0, \ldots, \alpha_n \rangle)$ et λ la trace produite par la procédure à la ligne 33. Le nœud choisi pour étendre l'arbre est $n_c = (s'_0, \langle \mathit{reduction}(\alpha_0), \alpha_1, \ldots, \alpha_k \rangle)$ avec $s'_0 = s_0 \cup \mathit{assump}(\alpha_0)$ et $\mathit{reduction}(\alpha_0) = \langle r_1, \ldots, r_j \rangle$ (ligne 25). Par hypothèse d'induction, λ raffine $\mathcal{P} = (s'_0, \mathcal{O}, \mathcal{M}, \langle r_1, \ldots, r_j, \alpha_1, \ldots, \alpha_n \rangle)$. Or, α_0 est complexe et sa réduction réalise a_0 à partir de s_0. Donc, d'après le cas 3 de la définition 6.4, $\lambda = \langle a_0, \ldots, a_k \rangle$ raffine $\mathcal{P} = (s_0, \mathcal{O}, \mathcal{M}, \langle \alpha_0, \ldots, \alpha_n \rangle)$.

Proposition. 6.2 (Complétude) Soit $\mathcal{P} = (s_0, \mathcal{O}, \mathcal{M}, \langle \alpha_0, \ldots, \alpha_n \rangle)$ un problème pour lequel il existe un raffinement. Il existe au moins une trace non déterministe $\lambda = \langle a_0, \ldots, a_k \rangle$ de `ExpandTree`$(n, \mathcal{O}, \mathcal{M})$ qui retourne un nœud $(s_k, \langle\rangle)$ tel que λ raffine $\mathcal{P}$.

Preuve. 6.2 (Complétude) La procédure `ExpandTree` explore un arbre. Or, le coefficient de branchement de l'arbre de raffinements construit par la procédure est fini. En effet, il existe un nombre limité d'opérateurs ou de méthodes appartenant à $\mathcal{P}$ applicables pour réaliser une action particulière. Cette limitation provient du fait qu'une action a_0 n'est déclenchée pour réaliser une action α_0 que si $\mathit{name}(a_0)$ peut être unifié avec $\mathit{name}(\alpha_0)$. Par conséquent, la seule façon pour que la procédure ne soit pas complète est qu'il existe dans l'arbre de raffinements une branche de longueur infinie. Or, le choix non déterministe du nœud (ligne 33) garantit que s'il existe un chemin de longueur fini alors la procédure le trouvera. Donc, la procédure `ExpandTree` est complète. De façon plus formelle, il est possible de faire une preuve par induction sur la longueur du raffinement λ.

Cas de base $(m = 0)$ *:* $\mathcal{P} = (s_0, \mathcal{O}, \mathcal{M}, \langle\rangle)$ a pour raffinement $\lambda = \langle\rangle$. Ce raffinement est immédiatement donné par la ligne 2.

Induction : admettons la complétude d'un raffinement de longueur inférieure à m. Soit $\mathcal{P} = (s_0, \mathcal{O}, \mathcal{M}, \langle \alpha_0, \ldots, \alpha_n \rangle)$. Deux cas sont possibles :

Cas 1 : α_0 est primitive. Pour que $\mathcal{P}$ ait un raffinement de longueur m, il faut que la ligne 33 renvoie un raffinement de longueur $j < m$ pour $\mathcal{P} = (s_0, \mathcal{O}, \mathcal{M}, \langle \alpha_1, \ldots, \alpha_n \rangle)$. Par induction, cette appel récursif la trouve et, par conséquent, l'appel initial retourne une solution de longueur m par ajout à gauche de a_0 à la trace retournée à la ligne 33. En effet, le raffinement de $\langle \alpha_1, \ldots, \alpha_n \rangle$ est nécessairement plus petit que celui de $\langle \alpha_0, \ldots, \alpha_n \rangle$.

Cas 2 : α_0 est complexe. Le raisonnement est identique.

6.2.3 Les heuristiques et le contrôle

Notons que, dans le cas d'une implémentation par définition déterministe de la procédure d'expansion de l'arbre, la complétude n'est pas nécessairement garantie.

Exemple 6.7 En effet, supposons qu'un agent veuille réaliser la séquence d'actions composée d'une unique action method et que l'état initial de ses croyances soit vide. Les méthodes et les opérateurs disponibles sont décrits ci-dessous :

method()
    precond: (aucune précondition)
    reduction: !op1(), method(), !op2()

method()
    precond: (aucune précondition)
    reduction: !do-nothing

!op1()
    precond: (aucune précondition)
    add: (aucun effet à ajouter)
    del: (aucun effet à supprimer)

!op2()
    precond: (aucune précondition)
    add: (aucun effet à ajouter)
    del: (aucun effet à supprimer)

!do-nothing()
    precond: (aucune précondition)
    add: (aucun effet à ajouter)
    del: (aucun effet à supprimer)

L'expansion de l'arbre de raffinements peut conduire à la construction d'une infinité de raffinements solution :

$$\chi_0 = \langle \text{!do-nothing()} \rangle$$
$$\chi_1 = \langle \text{!op1()}, \text{!do-nothing()}, \text{!op2()} \rangle$$
$$\chi_2 = \langle \text{!op1()}, \text{!op1()}, \text{!do-nothing()}, \text{!op2()}, \text{!op2()} \rangle$$
$$\chi_3 = \langle \text{!op1()}, \text{!op1()}, \text{!op1()}, \text{!do-nothing()}, \text{!op2()}, \text{!op2()}, \text{!op2()} \rangle$$
$$\ldots$$

La première solution pour résoudre ce problème est de contraindre la description des méthodes pour qu'elles ne puissent être appelées récursivement qu'un nombre

limité de fois. Cette restriction est une contrainte classique posée par les planificateurs hiérarchiques. La seconde solution consiste à effectuer une expansion de l'arbre de raffinements en largeur d'abord. Cette seconde solution est plus coûteuse en temps de calcul.

En gardant cette problématique en tête, rappelons que l'objectif de la planification sous hypothèses est de produire des conjectures « raisonnables » et « robustes ». Par raisonnable nous entendons des conjectures qui formulent le moins possible d'hypothèses. Plus une conjecture contient d'hypothèses, plus il sera difficile de les raffiner pour un obtenir un plan-solution. La notion de robustesse est à associer à la notion de réfutation. En effet, plus une conjecture contient de liens causaux, plus elle peut être réfutée.

À partir de ces considérations, nous proposons de guider l'expansion de l'arbre de raffinements par une recherche de type *« greedy search »*. À chaque récursion, le nœud choisi est le nœud formulant le moins d'hypothèses. En cas d'égalité, les nœuds sont départagés en fonction du nombre de liens causaux contenus dans le raffinement qu'ils représentent. Nous couplons à cette recherche une borne d'exploration b qui définit le nombre maximum d'hypothèses que peut formuler un raffinement. Ceci limite l'expansion de l'arbre aux seuls raffinements dont le nombre d'hypothèses est inférieur à b. Cette possibilité peut être utile pour adapter l'algorithme aux ressources du système et restreindre les raffinements produits par les agents. D'autre part, si l'objectif est de trouver la ou les raffinements optimaux en termes d'hypothèses, la procédure de recherche peut être effectuée de manière itérative par incrément successif de b. Cette technique prend en compte le problème évoqué par l'exemple 6.7 et garantit la complétude de la procédure de raffinements. Notons que, dans le cas où $b = 0$, les raffinements produits par la procédure ne contiennent aucune hypothèse et représente, par conséquent, des plans au sens « classique » de la planification.

Conclusion

Dans ce chapitre nous avons détaillé le dernier mécanisme de raffinement sur lequel est fondé notre modèle dialectique de synthèse de plans : la planification sous hypothèses[3]. Étant donné une séquence initiale d'actions, l'algorithme cherche le raffinement formulant le moins d'hypothèses en décomposant récursivement cette séquence en actions primitives qui peuvent être exécutées par un agent. À chaque étape, si aucune instanciation complète d'un opérateur ou d'une méthode n'est trouvée, les contraintes de leur déclenchement sont relaxées et les hypothèses nécessaires à leur exécution sont calculées. La construction de l'arbre de raffinements est guidée par une fonction d'évaluation qui guide la production des conjectures les plus raisonnables (*i.e.,* formulant le moins d'hypothèses) et les plus robustes (*i.e.,* contenant le moins de liens causaux).

Le choix d'une technique de planification hiérarchique présente de nombreux avantages dans le cadre de notre approche. Tout d'abord, comme l'a démontré (Erol

[3]Une version du planificateur est disponible à l'adresse http://core.imag.fr.

et al., 1996), les représentations utilisées en planification hiérarchique sont plus expressives que celles utilisées par les modèles classiques. Il est notamment possible d'introduire des mécanismes d'inférence au sein du processus de planification, en conservant les propriétés de complétude et de correction. Ce point particulier paraît intéressant si l'on désire, par la suite, étendre les mécanismes de réfutations de notre approche.

D'une manière générale, la description des méthodes engendre un surcoût lors de la spécification des agents. Toutefois, ces méthodes qui peuvent être vues comme des heuristiques guidant la production des conjectures permettent l'élaboration d'hypothèses « réalistes ». Nos tentatives d'adaptation des algorithmes fondés sur les graphes (Blum and Furst, 1997) se sont avérées infructueuses et ceci pour deux raisons. L'espace des hypothèses pouvant être produites est moins contraint de part l'absence dans la description du domaine des méthodes qui régulent la production des hypothèses. La seconde difficulté rencontrée réside dans le fait que l'introduction d'une hypothèse à un niveau propositionnel du graphe impose le calcul des relations d'exclusion entre les propositions. Ce calcul est très coûteux puisqu'il nécessite le test de la cohérence de l'hypothèse introduite.

Finalement, dans un contexte multi-agent, l'algorithme est adapté au interactions entre agents. Lorsqu'une conjecture est réfutée et qu'aucune réparation ne peut être construite, la conjecture doit être abandonnée. Dans ce cas, le coût d'élaboration d'une autre conjecture est faible car les agents peuvent réutiliser l'arbre de raffinements précédemment construit et poursuivre son exploration.

Chapitre 7

Étude de cas

Introduction

Ce dernier chapitre a pour but d'illustrer, au travers d'une étude de cas[1], les mécanismes de la synthèse dialectique de plans. Dans un premier temps, nous introduirons l'architecture des agents en tissant le lien entre le modèle introduit dans les chapitres précédents et ses différents composants. Puis, nous présenterons deux exemples soulignant les caractéristiques de notre approche.

7.1 Une architecture pour des agents planifiants

L'architecture des agents, représentée à la figure 7.1, est composée de deux principaux composants : le *module de raisonnement* qui spécifie les mécanismes pour produire les opérateurs dialectiques et le *gestionnaire de dialogue* qui spécifie les mécanismes permettant à l'agent d'interagir. Ces deux composants s'articulent autour du *tableau de démonstration*.

Le tableau de démonstration. Comme nous l'avons déjà souligné, le tableau de démonstration représente à la fois l'espace de conjectures co-construit par les agents ainsi qu'un enregistrement du dialogue. Il se définit comme un graphe orienté dont les nœuds représentent des conjectures et les arêtes des opérateurs de dialectiques. Chaque arête sortant d'un nœud χ est un opérateur dialectique qui transforme une conjecture χ en une conjecture successeur χ'. La mise à jour du tableau de démonstration est effectuée par le gestionnaire de dialogue soit sur réception d'une proposition émanant d'un autre agent soit sur proposition directe du module de raisonnement de l'agent lui-même. Deux sous-modules sont en charge de garantir l'intégrité des conjectures stockées dans le tableau de démonstration :

[1] Les cas sont extraits du jeu « The Settlers » qui présente l'intérêt d'être un benchmark connu de la communauté en planification (cf. http://planning.cis.strath.ac.uk/competition/domains.html).

1. *Le gestionnaire des contraintes d'ordre* qui vérifie que les modifications apportées aux conjectures n'introduisent pas de cycle de dépendance entre les actions (cf. figure 4.3).
2. *Le gestionnaire des contraintes d'instanciation* qui teste que chaque conjecture du tableau de démonstration possède un ensemble de contraintes d'instanciation cohérent.

Le module de raisonnement. Ce module définit le comportement de l'agent en dirigeant ses interactions. Il s'appuie sur le tableau de démonstration pour sélectionner la conjecture la plus « *prometteuse* » (cf. § 5.2.2) à raffiner, réparer ou encore réfuter et sur le *gestionnaire de dialogue* pour soumettre les propositions aux autres agents. Le module de raisonnement est constitué de la boucle de raisonnement par conjecture - réfutation (cf. algorithme 5.1) ainsi que de trois sous modules. Chaque sous-module est responsable de la production d'un type d'opérateur dialectique particulier (cf. § 5.3).

Le gestionnaire de dialogue. Le rôle du gestionnaire de dialogue est de gérer la contextualisation du dialogue, *i.e.,* l'état du dialogue en fonction des propositions reçues par les autres agents, mais également celles reçues par le module de raisonnement. Les états du dialogue sont représentés par un automate à états finis (cf. § 5.3). Chaque état représente une phase du dialogue, tandis que les arêtes représentent les actes de dialogue émis ou reçus par le gestionnaire. Le processus d'envoi et de réception des messages est assuré par une couche de communication CoRe-DMS (cf. annexe B) qui garantit l'ordre causal (Lynch, 1996).

7.2 Cas 1 : « La production d'outils »

Nous proposons dans ce paragraphe un exemple dans lequel le processus dialectique de synthèse de plans échoue. Cet exemple nous permettra de souligner plus particulièrement les mécanismes d'évaluation mis en oeuvre pour détecter l'absence de plan-solution.

7.2.1 Présentation

Considérons le domaine composé de trois agents : un fabriquant d'outils tool-maker, un bûcheron lumberjack et un scieur sawmill-worker. Le but qui leur est assigné est le suivant :

$$g \quad = \quad \{\text{has-tool}(\text{tool-maker}, \text{scythe})\}$$

Avant de présenter plus en détails les différents opérateurs dialectiques proposés par les agents, illustrons le processus de co-construction du plan-solution au travers du dialogue informel suivant :

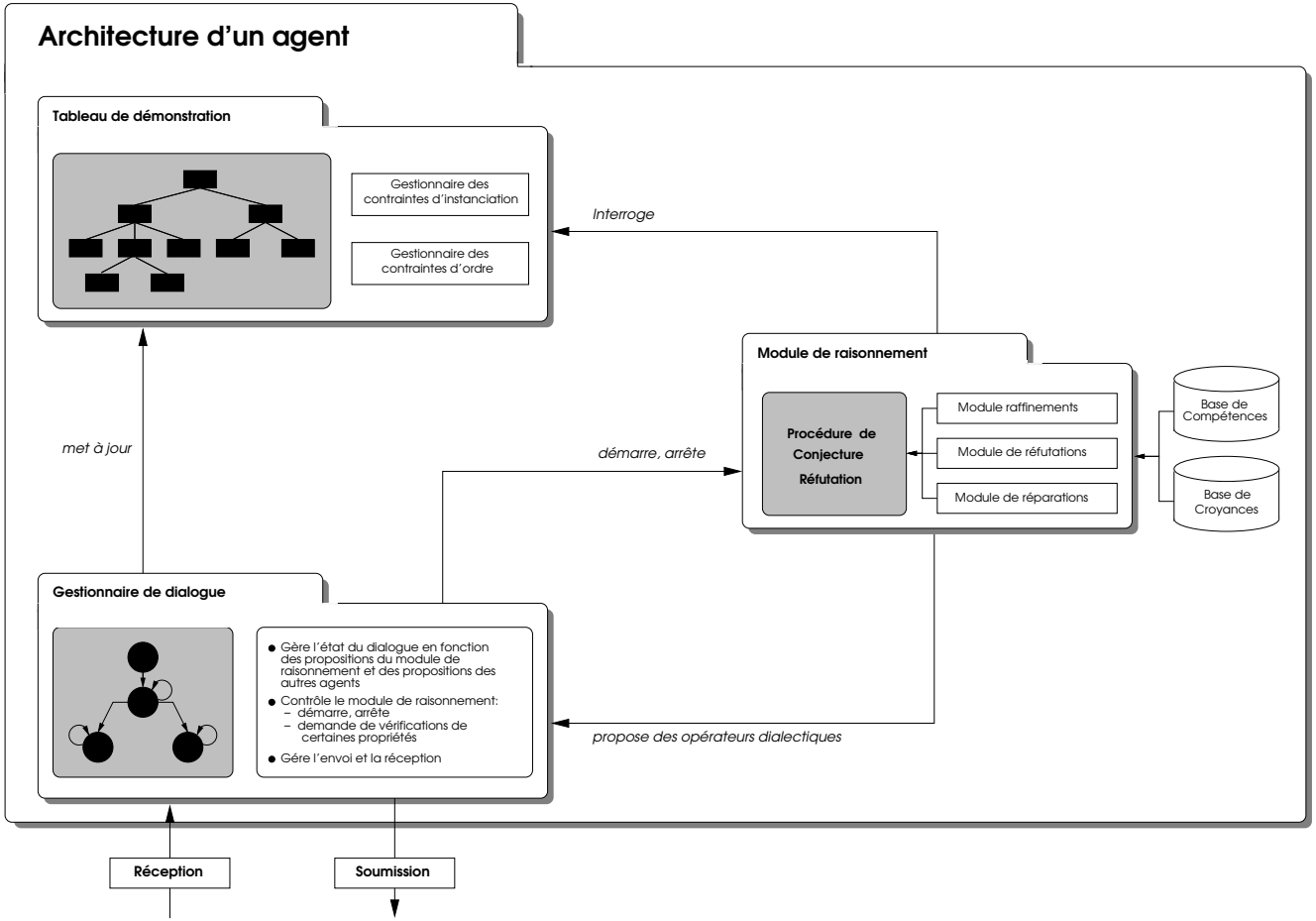


FIG. 7.1: Schéma de l'architecture interne des agents. Les flèches traduisent les dépendances entre les différents modules de l'architecture.

tool-maker$_1$:	« Je propose de fabriquer la faux. Mais, il me manque du bois. »
sawmill-worker$_1$:	« Je peux te le vendre si tu veux. Mais, il faut que tu me paies la somme de 1 euros. Pour cela, je vais scier un tronc d'arbre. J'espère que quelqu'un sera capable de me le fournir. »
tool-maker$_2$:	« D'accord, je te paie donc 1 euros pour le bois. »
lumberjack$_1$:	« D'après ce que je comprends, il te faut un tronc d'arbre. Je me charge d'en couper un dans la forêt. Mais, il faut que tu paies 2 euros pour mes services et que je me procure une hache. »
sawmill-worker$_2$:	« C'est d'accord pour moi, je te paie 2 euros. »
tool-maker$_3$:	« Je ne suis pas à un outil près. Je te fabriquai donc ta hache. Tu me paies 3 euros et je me débrouillerai pour obtenir le bois et le fer qu'il me manque. »
lumberjack$_2$:	« Je suis désolé mais je n'ai pas cette somme. »
tool-maker$_4$:	« Puisque personne ne semble capable de me payer, je crois que cela ne sert à rien de continuer à réfléchir. Nous n'arriverons pas à confectionner la faux. »
lumberjack$_3$:	« Je suis d'accord avec toi. »
sawmill-worker$_3$:	« Moi aussi. »

Les états initiaux des croyances des agents sont définis par les ensembles suivants :

$$\mathit{beliefs}(\mathsf{tool\text{-}maker}) = \left\{ \begin{array}{l} \mathsf{has\text{-}goods}(\mathsf{tool\text{-}maker}, \mathsf{wood}, 0), \\ \mathsf{has\text{-}goods}(\mathsf{tool\text{-}maker}, \mathsf{iron}, 2), \\ \mathsf{has\text{-}cash}(\mathsf{tool\text{-}maker}, 1), \\ \mathsf{price}(\mathsf{tool\text{-}maker}, \mathsf{axe}, 3) \end{array} \right\}$$

$$\mathit{beliefs}(\mathsf{lumberjack}) = \left\{ \begin{array}{l} \mathsf{has\text{-}goods}(\mathsf{lumberjack}, \mathsf{tree\text{-}trunk}, 0), \\ \mathsf{has\text{-}cash}(\mathsf{lumberjack}, 0), \\ \mathsf{price}(\mathsf{lumberjack}, \mathsf{tree\text{-}trunk}, 2) \end{array} \right\}$$

$$\mathit{beliefs}(\mathsf{sawmill\text{-}worker}) = \left\{ \begin{array}{l} \mathsf{has\text{-}goods}(\mathsf{sawmill\text{-}worker}, \mathsf{tree\text{-}trunk}, 0), \\ \mathsf{has\text{-}cash}(\mathsf{sawmill\text{-}worker}, 3), \\ \mathsf{price}(\mathsf{sawmill\text{-}worker}, \mathsf{wood}, 1) \end{array} \right\}$$

7.2.2 Déroulement du dialogue

Regardons maintenant comment se déroule la co-construction du plan par les agents. La synthèse dialectique du plan débute après l'envoi de l'acte $\mathit{prop.solve}(\mathcal{P})$ à l'ensemble des agents du domaine. Tous les agents se trouvent alors dans l'état **CoRe** de l'automate de contextualisation (cf. figure 5.3). À la réception de cet acte, le gestionnaire de dialogue de chaque agent initialise le tableau de démonstration avec

une action a_∞ comportant la précondition has-tool(tool-maker,scythe) et une action a_0 dont les effets représentent l'état initial des croyances de l'agent. Finalement, le gestionnaire démarre le module de raisonnement qui dirige les propositions de raffinements, de réfutations et de réparations (cf. algorithme 5.1).

tool-maker$_1$. L'agent tool-maker débute le dialogue en proposant de raffiner[2] la propriété has-tool(tool-maker,scythe). Les tableaux de démonstration des différents agents sont alors mis à jour[3] et contiennent dorénavant le raffinement suivant :

```
;; Le fabriquant d'outils tool-maker confectionne l'outil scythe.
!make(tool-maker,scythe)
    precond:  has-goods(tool-maker,iron,2),
              has-goods(tool-maker,wood,1)*
    add:      has-goods(tool-maker,iron,1),
              has-goods(tool-maker,wood,0),
              has-tool(tool-maker,scythe)
    del:      has-goods(tool-maker,iron,1),
              has-goods(tool-maker,wood,1)
```

Pour l'instant, rien ne garantit que l'agent tool-maker possède la quantité de bois nécessaire à la fabrication de la faux. Le raffinement formule donc l'hypothèse has-goods(tool-maker,wood,1)[4].

sawmill-worker$_1$. Le scieur de bois propose alors de raffiner l'hypothèse has-goods(tool-maker,wood,1) en énonçant le raffinement suivant :

```
;; Le scieur sawmill-worker scie une quantité 1 tronc d'arbre.
!saw(sawmill-worker,tree-trunk,1)
    precond:  has-goods(sawmill-worker,wood,0),
              has-goods(sawmill-worker,tree-trunk,1)*
    add:      has-goods(sawmill-worker,wood,1),
              has-goods(sawmill-worker,tree-trunk,0)
    del:      has-goods(sawmill-worker,wood,0),
              has-goods(sawmill-worker,tree-trunk,1)
```

[2]Tous les raffinements considérés dans cet exemple sont des raffinements par ajout de sous-conjecture (cf. § 5.3.1).

[3]Nous supposons que chaque acte entraîne la mise à jour des tableaux de démonstration des autres agents en respectant un ordre causal.

[4]Les hypothèses sont marquées par un astérisque.

;; Le scieur sawmill-worker vend au fabriquant tool-maker la quantité 1 de bois.
!sell(sawmill-worker,tool-maker,wood,1)
precond: has-cash(sawmill-worker,2),
has-goods(sawmill-worker,wood,1),
receive-cash(sawmill-worker,tool-maker,wood,1)*
add: has-cash(sawmill-worker,3),
has-goods(sawmill-worker,wood,0)
del: has-cash(sawmill,2),
has-goods(sawmill-worker,wood,1),
receive-cash(sawmill-worker,tool-maker,wood,1)

Dorénavant, la conjecture formule deux hypothèses. En effet, le scieur suppose qu'un autre agent sera capable de lui fournir un tronc d'arbre, has-goods(sawmill-worker,tree-trunk,1) et qu'il recevra un euro de la part du fabriquant d'outils, receive-cash(sawmill-worker,tool-maker,wood,1).

tool-maker$_2$. Le fabriquant d'outils décide de payer le scieur en énonçant le raffinement suivant :

;; Le fabriquant d'outils tool-maker paie 1 euros au scieur sawmill-worker.
!pay(tool-maker,sawmill-worker,wood,1)
precond: has-cash(tool-maker,1)
add: has-cash(tool-maker,0)
del: has-cash(tool-maker,1)

Il ne reste alors qu'une seule hypothèse à raffiner, has-goods(sawmill-worker,tree-trunk,1).

lumberjack$_1$. Le bûcheron propose ses services pour raffiner la dernière hypothèse de la conjecture en coupant un arbre et en le vendant au scieur de bois :

;; Le bûcheron lumberjack coupe 1 arbre.
!cut(lumberjack,tree-trunk,1)
precond: has-tool(lumberjack,axe)*,
has-good(lumberjack,tree-trunk,0)
add: has-good(lumberjack,tree-trunk,1)
del: has-good(lumberjack,tree-trunk,0)

;; Le bûcheron lumberjack vend au scieur sawmill-worker un 1 tronc d'arbre.
!sell(lumberjack,sawmill-worker,tree-trunk,1)
precond: has-cash(lumberjack,0),
has-goods(lumberjack,tree-trunk,1),
receive-cash(lumberjack,sawmill-worker,tree-trunk,1)*
add: has-cash(lumberjack,2),
has-goods(lumberjack,tree-trunk,0)
del: has-cash(lumberjack,0),
has-goods(lumberjack,tree-trunk,1),
receive-cash(lumberjack,sawmill-worker,tree-trunk,1)

Ce raffinement ajoute deux hypothèses à la conjecture : le bûcheron suppose qu'un agent sera capable de lui fournir une hache, has-tool(lumberjack,axe) et que le scieur de bois lui paiera la coupe de l'arbre, receive-cash(lumberjack,sawmill-worker,tree-trunk,1).

sawmill-worker$_2$. Le scieur décide de régler ses dettes en raffinant l'hypothèse receive-cash(lumberjack,sawmill-worker,tree-trunk,1). Le raffinement proposé est constitué d'une seule action :

;; Le scieur de bois sawmill-worker paie 2 euros au bûcheron lumberjack pour le tronc d'arbre.
!pay(sawmill-worker,lumberjack,tree-trunk,2)
precond: has-cash(sawmill-worker,3)
add: has-cash(sawmill-worker,3)
del: has-cash(sawmill-worker,1)

À nouveau, la conjecture ne contient plus qu'une hypothèse. Il ne reste alors plus qu'à fournir au bûcheron une hache pour couper l'arbre nécessaire au scieur de bois.

tool-maker$_3$. Le fabriquant d'outils propose naturellement son aide pour fabriquer la hache et la vendre au bûcheron :

;; Le fabriquant d'outils tool-maker confectionne l'outil axe.
!make(tool-maker,axe)
precond: has-goods(tool-maker,iron,1),
has-goods(tool-maker,wood,1)*
add: has-goods(tool-maker,iron,0),
has-goods(tool-maker,wood,0),
has-tool(tool-maker,axe)
del: has-goods(tool-maker,iron,1),
has-goods(tool-maker,wood,1),

;; Le fabriquant d'outils tool-maker vend au bûcheron lumberjack l'outil axe.
!sell(tool-maker,lumberjack,axe,3)
precond: has-cash(tool-maker,0),
has-tool(tool-maker,axe),
receive-cash(lumberjack,tool-maker,axe,3)*
add: has-cash(tool-maker,2)
del: has-cash(tool-maker,0),
has-tool(tool-maker,axe),
receive-cash(lumberjack,tool-maker,axe,3)

Le fabriquant d'outils suppose que le bûcheron lui paiera la somme de trois euros, receive-cash(lumberjack,tool-maker,axe,3), et qu'il lui sera possible de se procurer le bois nécessaire à la fabrication de la hache, has-goods(tool-maker,wood,1).

lumberjack$_2$. Le bûcheron ne peut malheureusement pas payer car il ne possède pas cette somme d'argent. Il propage son échec de raffinement (*failure()*) concernant l'hypothèse receive-cash(lumberjack,tool-maker,axe,1).

tool-maker$_2$. Le fabriquant d'outils propose aux autres agents de mettre fin à la synthèse de plans (*prop.failure()*). Le fabriquant passe alors dans l'état **Failure** (cf. figure 5.3). En effet, il ne peut plus énoncer de raffinement pour l'une des hypothèses de la conjecture. À la réception de cette proposition, le bûcheron et le scieur passent dans l'état **Instance Failure**. Ils évaluent alors la proposition de sortie du fabriquant.

lumberjack$_3$. En réponse, le bûcheron acquitte la proposition de sortie. Comme le fabriquant d'outils, il n'est plus en mesure de faire progresser la synthèse dialectique de plans en proposant le raffinement d'une hypothèse de la conjecture. L'envoi de l'acquittement (*ack.failure()*) fait passer l'automate de contextualisation du bûcheron dans l'état **Failure**.

sawmiller-worker$_3$. Le scieur de bois acquitte également la proposition de sortie(*ack.failure()*) et passe dans l'état **Failure**. Après son acquittement, tous les agents sont dans l'état **Failure** traduisant ainsi l'absence de plan-solution.

Détaillons le mécanisme d'évaluation qui a conduit le scieur à acquitter la proposition de sortie. Contrairement aux autres agents, le scieur peut encore énoncer un raffinement pour l'une des hypothèses de la conjecture (*i.e.,* has-goods(tool-maker,wood,1)). Toutefois, étant donné qu'il n'existe aucun autre raffinement possible pour vérifier receive-cash(lumberjack,tool-maker,axe,3), le but global ne peut être atteint quel que soit le raffinement qu'il proposera. Ceci provient du fait que les hypothèses représentent des branches ET dans le tableau de démonstration (cf. figure 5.4). Autrement dit, toutes les hypothèses d'une conjecture doivent être vérifiées pour qu'une conjecture soit considérée comme un plan-solution. L'absence de raffinement pour l'une des hypothèses (dans notre exemple receive-cash(lumberjack,tool-

maker,axe,3)) permet ainsi d'éliminer au plus tôt les conjectures qui ne peuvent être des plans solutions.

La conjecture construite par les agents est représentée à la figure 7.2. Seule les actions ainsi que les contraintes d'ordre sont indiquées. Les hypothèses non raffinées sont mentionnées au-dessus des actions.

7.2.3 Discussion

La gestion des boucles de raffinements. Un problème importante lié à notre approche réside dans la détection des boucles de raffinements, *i.e.,* un agent formule une hypothèse qui est raffinée par un autre agent qui énonce à son tour l'hypothèse démontrée par le premier raffinement. Dans l'exemple présenté, le cas ne se produit pas. En effet, le bûcherons n'a plus assez d'argent pour payer la hache nécessaire à la coupe d'un arbre. Toutefois, il est possible d'imager aisément un exemple dans lequel le processus de synthèse de plans conduit aux raffinements récursifs des mêmes hypothèses.

Exemple 7.1 Supposons que nous ayons deux agents (cf. figure 7.3). Le premier ag1 produit un raffinement pour démontrer la propriété porte-ouverte. Le raffinement proposé contient une action !ouvrir et formule une hypothèse porte-fermee. Le second agent ag2 propose alors un raffinement pour démontrer l'hypothèse porte-fermee. Son raffinement est constitué d'une action !fermer qui formule l'hypothèse porte-ouverte. L'agent ag1 peut à nouveau énoncer le raffinement !ouvrir pour démontrer porte-ouverte, *etc.*

Une boucle de raffinements traduit le fait qu'aucun des agents impliqués dans la synthèse du plan ne connaît la valeur de vérité d'une propriété du monde. Par exemple, si un agent fait l'hypothèse porte-ouverte et qu'un autre agent sait comment la vérifier, la récursion s'arrête. Une façon de gérer ce problème consiste à interdire aux agents de formuler une hypothèse qu'ils ont précédemment énoncée, tant qu'ils ne sont pas certains qu'elle pourra être démontrée, *i.e.,* qu'il existe au sein de la conjecture un sous plan-solution permettant de la vérifier. En effet, ce plan-solution n'introduisant pas de nouvelle hypothèse, la récursion sur porte-ouverte ne peut plus se produire.

Pour mettre en œuvre ce mécanisme, il faut gérer pour chaque conjecture du tableau de démonstration la liste des hypothèses qui ne sont pas autorisées. À chaque nouveau raffinement ou réparation, les hypothèses pour lesquelles il existe un plan-solution sont supprimées de la liste, tandis que les hypothèses pour lesquelles il n'existe pas encore de plan-solution y sont maintenues. La prise en compte de ces hypothèses particulières dans le mécanisme de raffinements (cf. chapitre 6) consiste simplement à vérifier que les hypothèses engendrées lors de l'application d'un opérateur ou d'une méthode sont autorisées (cf. algorithme 6.2).

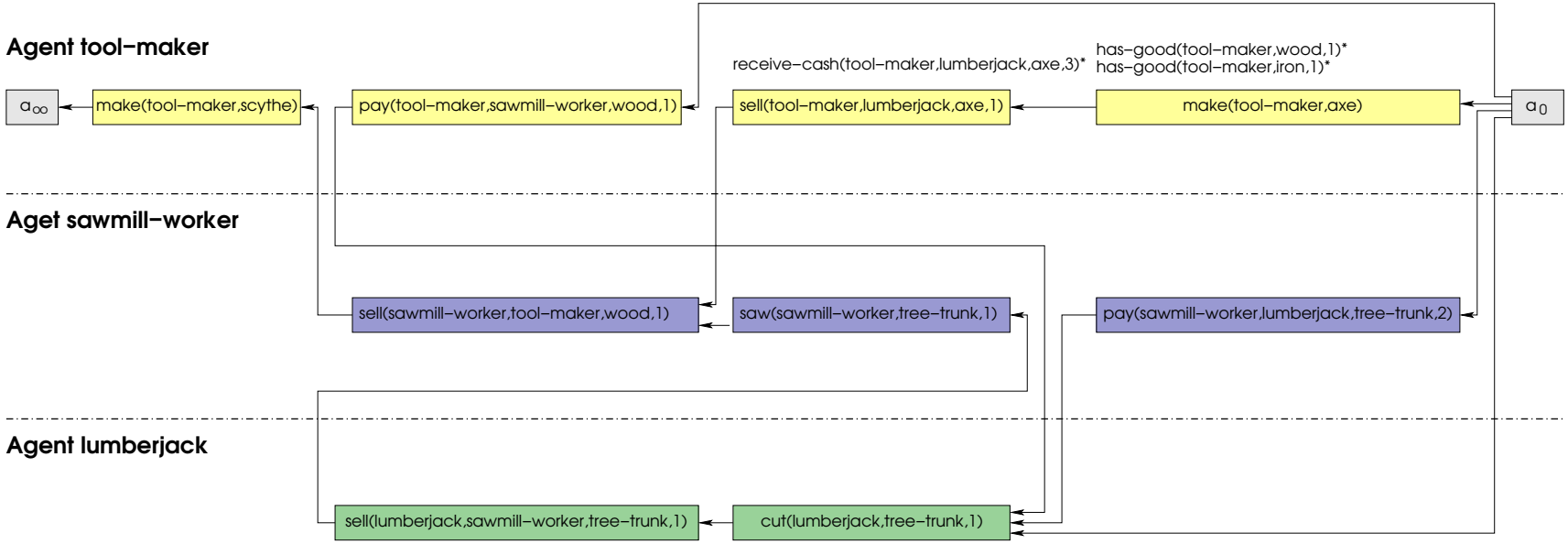


FIG. 7.2: Échec du processus de synthèse de plan.

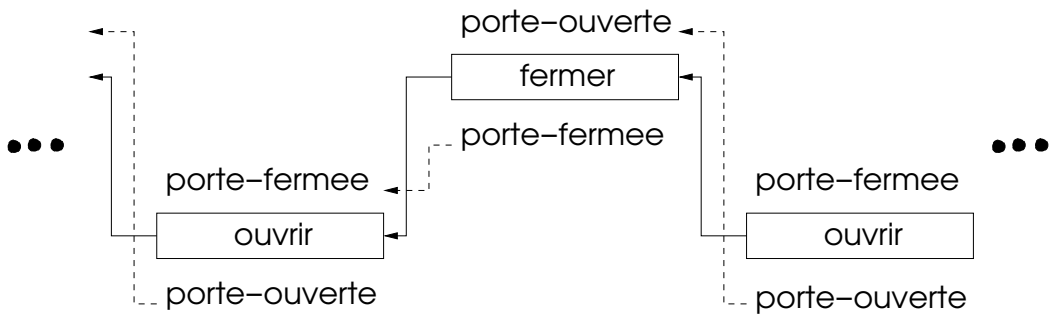


FIG. 7.3: Exemple de raffinements récursifs des hypothèses porte-ouverte et porte-fermee.

Extension à des systèmes ouverts. Il est important de souligner que dans le cas d'un échec, le processus de synthèse de plans permet d'identifier la ou les propriétés du monde qui en sont à l'origine (*e.g.,* receive-cash(lumberjack,tool-maker,axe,3)). Cette détection peut être utile si l'on désire étendre notre modèle. En effet, ces propriétés définissent les compétences manquantes pour mener à bien la synthèse d'un plan. Il est alors possible de mettre en place des mécanismes permettant de recruter des agents possédant ces compétences.

7.3 Cas 2 : « Le transport de marchandises »

Nous allons nous intéresser dans ce paragraphe au problème du transport de marchandises. Ce problème nous permettra de mettre en avant les différents types de réparations pouvant être réalisés ainsi que les mécanismes de partage de croyances.

7.3.1 Présentation

Nous supposons dans cet exemple que le domaine est composé de deux agents transporteurs dont le but est de déplacer du blé et de la farine dans un village donné. Le but est décrit ci-dessous :

$$g \quad = \quad \{\text{is-available}(\text{flour}, \text{village1}), \text{is-available}(\text{wheat}, \text{village1})\}$$

Le premier agent transporteur conveyor1 sait qu'une charrette cart1 se trouve au village village3 et que la farine est disponible au village village2. Il ne possède aucune information concernant la topologie des routes entre les villages, si ce n'est que le village1 est connecté au village3. L'état initial (cf. figure 7.4) de ses croyances est défini par l'ensemble de prédicats suivant :

$$\mathit{beliefs}(\text{conveyor1}) = \left\{ \begin{array}{l} \text{at}(\text{village3}, \text{cart1}), \\ \text{is-available}(\text{flour}, \text{village2}), \\ \text{connected}(\text{village1}, \text{village3}) \end{array} \right\}$$

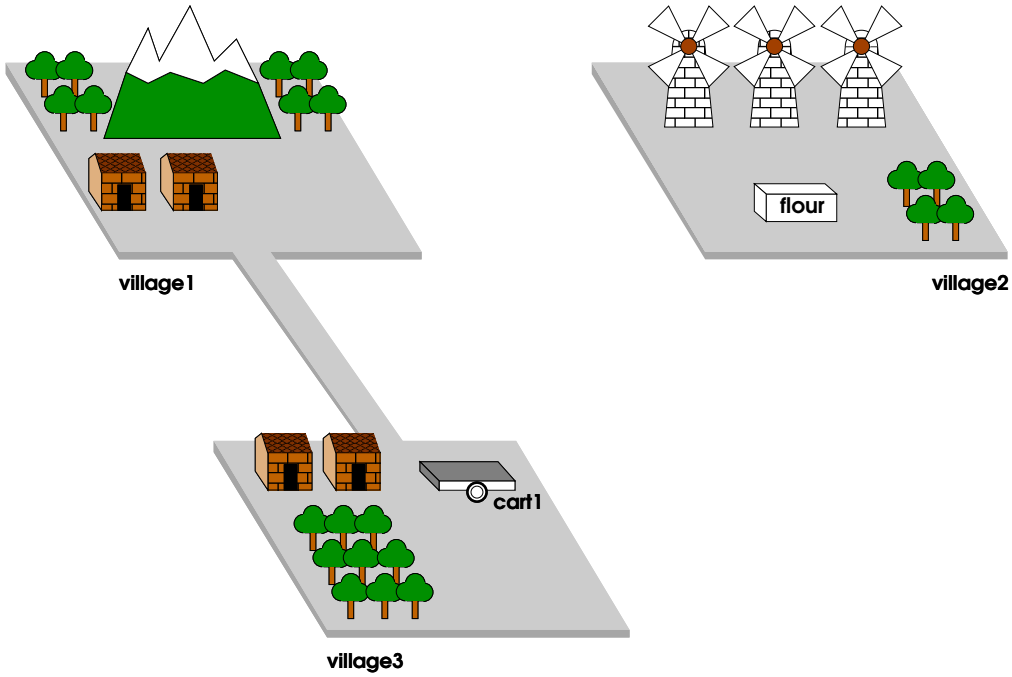


FIG. 7.4: État de croyance de l'agent conveyor1

De son côté, le second transporteur conveyor2 possède la carte complète des routes entre les différents villages. Il sait également que le blé est disponible dans le village village3 et que la charrette cart1 s'y trouve (cf. figure 7.5).

$$beliefs(\mathsf{conveyor1}) = \left\{ \begin{array}{l} \mathsf{connected}(\mathsf{village1}, \mathsf{village2}), \\ \mathsf{connected}(\mathsf{village1}, \mathsf{village3}), \\ \mathsf{connected}(\mathsf{village2}, \mathsf{village1}), \\ \mathsf{connected}(\mathsf{village2}, \mathsf{village3}), \\ \mathsf{connected}(\mathsf{village3}, \mathsf{village1}), \\ \mathsf{connected}(\mathsf{village3}, \mathsf{village2}), \\ \mathsf{is\text{-}available}(\mathsf{wheat}, \mathsf{village3}), \\ \mathsf{at}(\mathsf{village3}, \mathsf{cart1}) \end{array} \right\}$$

Illustrons le processus de co-construction du plan par le dialogue informel suivant :

conveyor1$_1$: « Je m'occupe de la farine. Je déplace la charrette cart1 du village3 au village2, je charge la farine dans la charrette, puis je déplace une nouvelle fois la charrette du village2 au village1 et finalement je décharge la farine. Tout ceci reste conditionné au fait qu'il existe une route entre les différents villages. »

conveyor2$_1$: « En ce qui me concerne, je charge le blé au village3 dans la charrette cart1, je la déplace au village1 et je décharge la marchandise. »

conveyor2_2 : « De plus, je t'informe qu'il existe une route entre les villages 3 et 2 et entre les villages 2 et 1. »

conveyor1_2 : « Merci pour les informations. Toutefois, je ne pense pas que nous ayons trouvé une solution à notre problème. En effet, je ne peux pas déplacer la charrette au village2 pendant que tu charges la farine. »

conveyor2_3 : « Oui effectivement, tu as raison. Je voudrais bien utiliser une autre charrette mais je n'en connais aucune autre. »

conveyor1_3 : « Je suis dans le même cas que toi. »

conveyor2_4 : « Je propose d'effectuer le chargement du blé dans la charrette cart1 et après tu pourras la déplacer. Qu'en penses tu ? »

conveyor1_4 : « Je ne suis toujours pas convaincu par ta proposition. Il n'est pas possible que je déplace la charrette cart1 au village3 pendant que toi tu la déplaces au village1. »

conveyor2_5 : « Décidément, tu as vraiment l'esprit de contradiction.»

conveyor1_5 : « J'ai peut être la solution. Je vais réaliser mon plan initial et je te rapporterai la charrette dans le village3. »

conveyor2_6 : « Je crois que nous avons enfin la solution. Est-ce que tu vois un autre problème qui pourrait empêcher la réalisation de notre plan ? »

conveyor1_6 : « Non, je ne crois pas. »

7.3.2 Déroulement du dialogue

De manière similaire à notre premier cas d'étude, le tableau de démonstration de chaque agent est initialisé à partir de leurs croyances respectives et du but à atteindre. Les deux transporteurs se trouvent dans l'état **CoRe** de l'automate de contextualisation (cf. figure 5.3). Intéressons nous maintenant aux raffinements proposés :

conveyor1_1. Le transporteur conveyor1 énonce un raffinement pour atteindre le but is-available(flour,village1). Ce raffinement formule l'hypothèse que les villages 3 et 2 ainsi que 2 et 1 sont connectés.

;; Le transporteur conveyor1 déplace la charrette cart1 du village3 au village2.
!move(conveyor1,cart1,village3,village2)
precond: connected(village3,village2)*, at(cart1,village3)
add: at(cart1,village2)
del: at(cart1,village3)

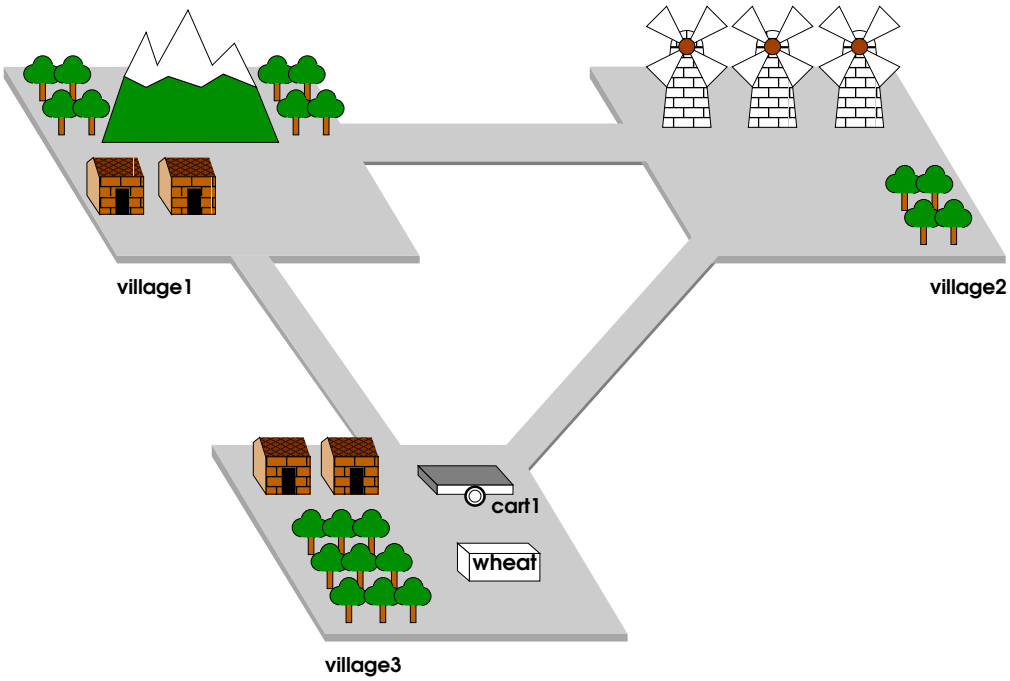


FIG. 7.5: État de croyance de l'agent conveyor2

;; Le transporteur conveyor1 charge la marchandise flour dans la charrette cart1 au village2.
!load(conveyor1,cart1,village2,flour)
precond: is-available(flour,village2), at(cart1,village2)
add: in(cart1,flour), at(cart1,village2)
del: is-available(flour,village2)

;; Le transporteur conveyor1 déplace la charrette cart1 du village2 au village1.
!move(conveyor1,cart1,village2,village1)
precond: connected(village2,village1)*, at(cart1,village2)
add: at(cart1,village1)
del: at(cart1,village2)

;; Le transporteur conveyor1 décharge la marchandise flour de la charrette cart1 au village1.
!unload(conveyor1,cart1,village1,flour)
precond: in(cart1,flour), at(cart1,village1)
add: is-available(flour,village1), at(cart1,village1)
del: in(cart1,flour)

conveyor2$_1$. Le transporteur conveyor2 propose de raffiner le second but, is-available(wheat,village1), sans formuler d'hypothèse.

;; Le transporteur conveyor2 charge la marchandise wheat dans la charrette cart1 au village3.
!load(conveyor2,cart1,village3,wheat)
precond: is-available(wheat,village3), at(cart1,village3)
add: in(cart1,wheat), at(cart1,village3)
del: is-available(wheat,village3)

;; Le transporteur conveyor2 déplace la charrette cart1 du village3 au village1.
!move(conveyor2,cart1,village3,village1)
precond: connected(village3,village1), at(cart1,village3)
add: at(cart1,village1)
del: at(cart1,village3)

;; Le transporteur conveyor2 décharge la marchandise wheat de la charrette cart1 au village1.
!unload(conveyor2,cart1,village1,wheat)
precond: in(cart1,wheat), at(cart1,village1)
add: is-available(wheat,village1), at(cart1,village1)
del: in(cart1,wheat)

conveyor2$_2$. Le raffinement proposé par conveyor2 est un raffinement par ajout de lien causal (cf. § 5.3.1). Ce type de raffinement permet (lorsque cela est possible) d'utiliser les effets d'une action déjà présente dans la conjecture pour démontrer la validité d'une hypothèse. Dans l'exemple, le tableau de démonstration du conveyor2 contient l'action a_0 dont les effets ont été initialisés à partir de l'état initial de ses croyances. Par conséquent, a_0 a pour effets connected(village3,village2) et connected(village2,village1). Après raffinement, les deux hypothèses formulées par le conveyor1 sont supportées par un lien causal provenant de l'action a_0 du conveyor2. Le conveyor1 sait dorénavant que les propriétés connected(village3,village2) et connected(village2,village1) sont vérifiées et qu'il peut les ajouter à sa base de croyances.

conveyor1$_2$. Le transporteur conveyor1 énonce une réfutation à l'encontre de la conjecture. En effet, l'action !move(conveyor1,cart1,village3,village2) réfute le lien causal entre les actions a_0 et !load(conveyor2,cart1,village3,wheat) (cf. figure 7.6).

conveyor2$_3$. Le conveyor2 cherche alors une réparation pour prendre en compte la réfutation. Il essaie, dans un premier temps, de modifier les contraintes d'instanciation portant sur la propriété at(cart1,village3) (cf. § 5.3.3). Malheureusement, la réparation échoue car il ne connaît aucune autre charrette disponible.

conveyor1$_3$. Le conveyor1 cherche également à réparer la réfutation qu'il a formulée à l'encontre de la conjecture en appliquant le même mécanisme. La réparation échoue à nouveau pour les mêmes raisons.

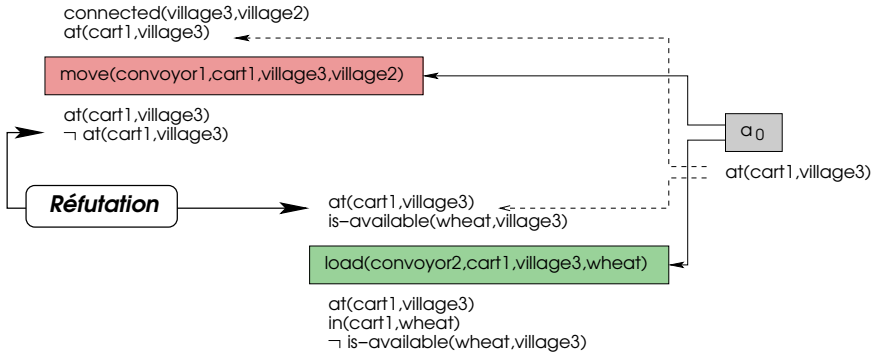


FIG. 7.6: Réfutation énoncée par $\mathsf{conveyor1}_2$.

$\mathsf{conveyor2}_4$. Le conveyor2 tente alors d'appliquer une réparation par ajout de contraintes d'ordre (cf. § 5.3.3). Il propose d'effectuer l'action !load(conveyor2, cart1,village3,wheat) avant l'action !move(conveyor1,cart1,village3,village2) garantissant ainsi que la charrette cart1 ne sera pas déplacée au cours du chargement (cf. figure 7.7).

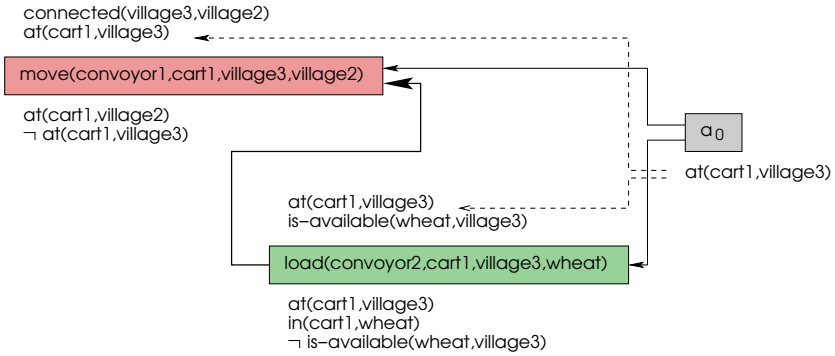


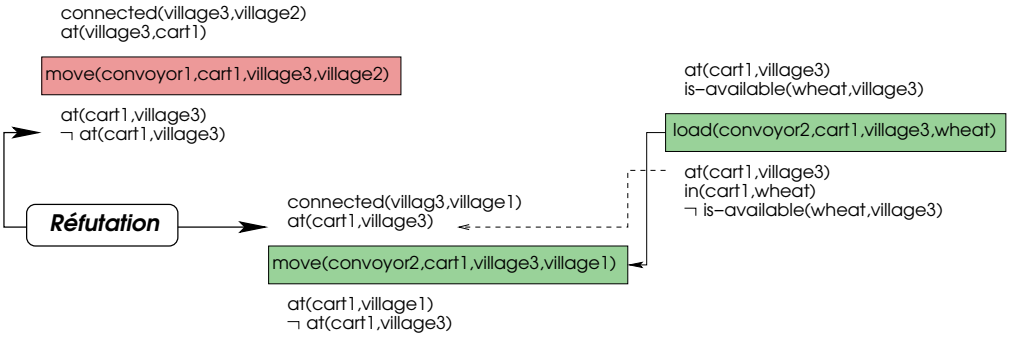
FIG. 7.7: Réparation énoncée par $\mathsf{conveyor2}_4$.

$\mathsf{conveyor1}_4$. Le convoyor1 accepte la réparation. Toutefois, il réfute une nouvelle fois la conjecture en soulignant que l'action !move(conveyor1,cart1,village3,village2) invalide le lien causal at(cart1,village3) entre les actions !load(conveyor2, cart1,village3,wheat) et !move(conveyor2,cart1,village3,village1) (cf. figure 7.3.2).

$\mathsf{conveyor2}_5$. Le conveyor2 accepte la réfutation et cherche un moyen de réparer la conjecture.

$\mathsf{conveyor1}_5$. Le conveyor1 propose une réparation par ajout d'une conjecture (cf. § 5.3.3). La conjecture résultat peut s'exprimer comme la séquence d'actions suivante :

1. !move(conveyor1,cart1,village3,village2)



2. !load(conveyor1,cart1,village2,flour)
3. !move(conveyor1,cart1,village2,village1)
4. !unload(conveyor1,cart1,village1,flour)
5. !move(conveyor1,cart1,village1,village3) (action ajoutée)
6. !load(conveyor2,cart1,village3,wheat)
7. !move(conveyor2,cart1,village3,village1)
8. !unload(conveyor2,cart1,village1,wheat)

Le mécanisme de réparation par ajout d'actions consiste à produire un conjecture capable de démontrer que la propriété du monde conflictuelle pourra être à nouveau vérifiée. Dans l'exemple, la réfutation porte sur la propriété at(cart1,village3). L'ajout de l'action !move(conveyor1,cart1,village1,village3) après l'exécution du plan du conveyor1 produit la propriété at(cart1,village3) nécessaire à la réalisation de celui du conveyor2.

conveyor2$_6$. conveyor2 n'a pas de réfutation à formuler à l'encontre de la réparation. De plus, elle ne formule plus aucune hypothèse. Par conséquent, conveyor2 propose de mettre fin au raisonnement sur succès (*prop.success()*). Son automate de contextualisation passe dans l'état **Success**.

conveyor1$_6$. À la réception de la proposition de sortie sur succès (état **Instance Success** de l'automate de contextualisation), conveyor1 vérifie qu'il ne peut plus réfuter la conjecture et que la conjecture ne formule plus aucune hypothèse (cf. proposition 4.1). Il acquitte alors la proposition de sortie et passe dans l'état **Success** exprimant qu'un plan-solution a été trouvé.

7.3.3 Discussion

Cette étude de cas nous a permis de mettre en évidence un certain nombre de points particuliers de notre modèle. Tout d'abord, nous avons montré comment les raffinements par ajout de liens causaux servaient de support au partage des croyances entre les agents. Puis, dans un second temps, au travers des différentes réfutations, nous avons illustré les trois types de réparations dont dispose notre modèle :

1. Les réparations par modifications de contraintes d'instanciation, qui définissent les mécanismes autorisant les agents à choisir une autre ressource lorsque celle-ci est conflictuelle ;
2. Les réparations par ajout de liens causaux, qui permettent d'ordonner temporellement l'accès à une ressource ;
3. Les réparations par ajout d'une conjecture, qui formalisent la coopération entre les agents pour produire à nouveau une ressource lors celle-ci a été consommée.

Conclusion

Nous avons présenté dans ce chapitre une étude de cas en s'appuyant sur le jeu « The Settlers »[5]. Ces cas ont mis en relief les conjectures et les réfutations produites par les agents ainsi que l'entrelacement entre les actes du niveau informationnel (*i.e.,* *refine()*, *refute()*, *repaire()*, *failure()*) et les actes du niveau de contextualisation nécessaires aux agents pour débuter ou clôturer le dialogue.

Pour plus de détails sur l'implémentation et le codage des agents, nous invitons le lecteur à se reporter aux annexes A et B.

[5] L'ensemble des agents présentés dans ce chapitre sont disponibles à l'adresse http://core.imag.fr.

Conclusion générale

Dans ce manuscrit, nous avons étudié la problématique de la synthèse de plans dans un contexte multi-agent en proposant de rapprocher le domaine de la planification et de l'argumentation. Dans cette perspective, nous avons examiné, tout d'abord, les mécanismes de coordination ainsi que les modèles de coopération par états mentaux. Puis, nous nous sommes plus particulièrement intéressés aux modèles dialectiques de l'argumentation en essayant de tisser un lien avec le domaine de la planification. Finalement, cette étude nous a conduits naturellement à proposer un modèle dialectique pour la synthèse de plans.

Contrairement aux approches dans lesquelles la planification n'est qu'un outil parmi d'autres pour définir les activités des agents et la coordination une « surcouche » nécessaire à leur synchronisation, la contribution de nos travaux repose sur la conception d'un modèle de planification entièrement distribué dans lequel les agents raisonnent de manière hypothétique sur leurs activités, en intégrant leurs compétences hétérogènes et leurs croyances partielles sur le monde. La synthèse de plans est considérée comme un raisonnement collectif et révisable fondé sur l'échange entre les agents de conjectures, *i.e.,* des plans qui peuvent être exécutés si certaines conditions sont vérifiées et de réfutations, *i.e.,* des objections quant à leurs réalisations. De manière similaire à un dialogue argumentatif, les agents proposent leurs compétences pour démontrer la validité de certaines hypothèses, réfuter les conjectures non réalisables, réparer lorsque cela est possible les conjectures précédemment réfutées et ainsi élaborer pas à pas un plan-solution.

Cependant, il convient également de souligner les limites de ce travail :

- ▷ L'approche proposée s'adresse à une classe de problèmes pour lesquels le nombre d'agents est restreint. Ceci est dû au nombre important d'échanges nécessaires à l'élaboration d'un plan. Toutefois, il est possible d'envisager une généralisation de l'approche dans laquelle l'entité minimale n'est plus un agent mais un groupe d'agents capable de produire des conjectures, des réfutations et des réparations à un niveau donné. Chaque groupe peut alors être vu comme un pôle de compétences. De manière plus générale, nous ne traitons pas dans ce manuscrit les aspects organisationnels associés à la synthèse de plans dans un contexte multi-agent. Cependant, ces aspects nous semblent importants dans la mise en œuvre de systèmes autonomes aptes à raisonner conjointement à la réalisation d'un objectif commun. En effet, accomplir une tâche complexe,

c'est aussi s'organiser dynamiquement en fonction des compétences disponibles, recruter éventuellement celles manquantes, prendre en compte les capacités limitées des agents afin de résoudre au mieux le but assigné au système.

- ▷ Les agents doivent partager un langage commun décrivant les propriétés du monde sur lesquelles ils peuvent raisonner de manière hypothétique. Dans l'optique de l'interconnexion et de la conception d'agents à plus grande échelle, *e.g.,* par plusieurs concepteurs, il apparaît important de s'appuyer sur des langages standards du domaine de la représentation de connaissances. Nous souhaitons au travers de cette remarque souligner le lien possible entre les résultats obtenus depuis quelques années concernant la sémantique des représentations de connaissances (reposant sur des formalismes tels que la logique terminologique ou les graphes conceptuels) et les représentations classiquement utilisées en planification. En effet, ces travaux peuvent constituer une voie de recherche prometteuse pour la conception de systèmes planifiants ouverts.

L'état d'avancement de nos travaux permet d'entrevoir un certain nombre de perspectives :

- ▷ Si, au cours de notre étude, nous nous sommes focalisés sur les mécanismes permettant à un groupe d'agents de produire un plan-solution, nous n'avons que très peu considéré la notion de robustesse des plans produits. Par robustesse, nous entendons la capacité d'un plan à tolérer plus ou moins des aléas d'exécution ou des croyances erronées sur l'état du monde. Supposons que nous soyons capables de construire un faisceau de plans solutions, *i.e.,* un plan contenant différentes alternatives. Les questions qui se posent alors sont les suivantes : comment identifier leurs points faibles pouvant conduire le cas échéant à les rendre non réalisables ? Comment choisir de manière rationnelle le plan le plus robuste minimisant ainsi le risque d'échec à l'exécution ? Par exemple, dans le cadre de la planification d'un voyage, cela revient à envisager plusieurs méthodes pour se rendre à l'aéroport ainsi que plusieurs vols possibles. Ces considérations peuvent s'étendre également aux ressources et aux compétences des agents. En effet, le fait de posséder deux voitures ou que plusieurs agents soient capables de les conduire participe à la robustesse du plan proposé. La notion de robustesse peut également passer par la mutualisation des ressources. En rationalisant leur consommation, il peut alors être possible de dégager des économies qui sont susceptibles de fournir une plus grande marge de manœuvre au moment de l'exécution. Une étude approfondie de ces mécanismes permettrait, selon nous, de produire des plans plus réalistes prenant en compte au plus tôt, *i.e.,* au moment de leur construction, les aspects liés à leur échec éventuel.

- ▷ L'absence de prise en compte de ces échecs est un reproche souvent formulé à l'encontre de la planification. Par exemple, lors de l'exécution d'un plan par un robot mobile, les propriétés du monde supposées vérifiées lors de la conception du plan peuvent s'avérer erronées : les portes qui auraient dues être ouvertes sur le chemin du robot sont finalement fermées, les objets devant

être déplacés ne sont pas à l'endroit où ils devraient être. Par conséquent, il est indispensable de contrôler l'exécution du plan afin de le mettre à jour lorsqu'un échec se produit au cours de son exécution. Dans la mesure où la modélisation de l'environnement est nécessairement incomplète et incertaine, de tels aléas doivent être tenus pour inévitables. Ceci nous amène à nous interroger sur les modifications à apporter à notre approche dans la perspective d'introduire une phase d'exécution. Si l'on se réfère aux notions d'hypothèse et de réfutation présentées dans ce manuscrit, un échec peut être vu comme une réfutation non plus déduite par un agent mais provenant de l'environnement lui-même[6]. Cette réfutation ne peut être remise en cause puisqu'elle est issue d'une perception de l'environnement. Dans ce cas, il n'est pas possible de la résoudre par des mécanismes de négociation. Si l'exécution du plan nécessite que la porte soit ouverte et qu'un agent constate que la propriété n'est pas satisfaite, les agents doivent alors la considérer comme une hypothèse implicitement formulée qui n'a pas été encore raffinée.

- ▷ Finalement, nous souhaitons aborder une perspective à plus long terme. Les mécanismes dialectiques par conjecture - réfutation, adaptés dans le cadre de ce manuscrit à la synthèse de plans, possèdent selon nous une portée plus générique. En effet, il serait intéressant d'examiner comment ces mécanismes peuvent être mis en œuvre afin de permettre à des agents de raisonner sur des propriétés du monde et construire ainsi collaborativement des croyances communes.

Nous espérons avoir montré, au travers de ce travail et des perspectives que nous venons d'évoquer, l'intérêt d'une approche dialectique de la synthèse de plans et plus généralement du mécanisme de conjecture - réfutation.

[6] Nous retrouvons ici le sens premier donné par (Lakatos, 1976) à la notion de réfutation : *« une réfutation est une expérimentation ou un test qui conduit à invalider la preuve. »*

Annexe A

Tutoriel de CoRe Assumption-Based Planner

Introduction

L'objectif de cette annexe est de décrire les mécanismes du planificateur CoRe Assumption-Based Planner (Pellier and Fiorino, 2004) et de fournir les connaissances nécessaires à son utilisation. CoRe-ABP est utilisé au sein du projet CoRe[1] (Pellier and Fiorino, 2005a; Pellier and Fiorino, 2005b) dont le but est la conception d'un planificateur multi-agent basé sur le partage des compétences et des croyances.

Dans un premier temps, nous présenterons succinctement les caractéristiques du planificateur CoRe-ABP puis détaillerons le codage de deux domaines. Finalement, nous introduirons formellement la syntaxe utilisée.

A.1 Présentation générale

CoRe-ABP aborde la problématique de la conception d'agents intelligents capables de produire des plans à partir de croyances incomplètes. Supposons qu'une porte soit fermée, et qu'un agent ne soit pas capable de l'ouvrir mais en revanche qu'il soit capable de réaliser 100% de ses objectifs derrière la porte, la planification classique échoue. Imaginons maintenant que l'agent fasse l'hypothèse que la porte est ouverte et qu'un autre agent sera capable de lui ouvrir, si l'hypothèse se vérifie alors le plan pourra être réalisé. C'est dans cette optique que CoRe-ABP a été conçu. Contrairement aux planificateurs classiques qui ne parviennent pas à produire un plan lorsque l'ensemble des propriétés nécessaires à sa réalisation ne sont pas connues, CoRe-ABP propose d'identifier ces propriétés et de produire malgré tout un plan. Le plan produit peut alors être vu comme *un plan sous hypothèses*, *i.e.,* un plan qui peut s'exécuter si certaines conditions sont vérifiées. Bien évidemment, le but ne sera considéré comme atteint que lorsque toutes les hypothèses formulées par

[1] CoRe est l'acronyme de Conjecture Refutation.

le plan seront vérifiées. C'est alors le rôle des autres agents de vérifier les hypothèses précédemment énoncées. En ce sens, les hypothèses sont vues comme des sous-buts à atteindre pour les autres agents.

A.1.1 Principe

Les mécanismes de CoRe-ABP repose sur une technique de planification *domaine indépendante* appelée HTN, *« Hierarchical Transition Network »*. Le but est atteint par décomposition récursive d'actions complexes en actions primitives qui peuvent être réalisées par un agent. Les actions complexes sont décrites par des méthodes tandis que les actions primitives par des opérateurs. L'ensemble des méthodes et des opérateurs que peut exécuter un agent constitue ses compétences.

Les planificateurs classiques ne déclenchent une actions à partir d'un état du monde que si l'ensemble des préconditions de l'action y sont vérifiées. Dans le but de produire des plans sous hypothèses, nous relaxons cette contrainte. Nous considérons qu'une action est toujours applicable quel que soit l'état dans lequel l'agent souhaite la déclencher. Les préconditions de l'action qui ne sont pas vérifiées constituent alors les hypothèses nécessaires à son exécution.

A.1.2 Utilisation

CoRe-ABP[2] est implémenté en Java^{TM}1.5 et nécessite une machine virtuelle java[3]. Le code source ainsi que les fichiers binaires sont disponibles à l'adresse http://core.imag.fr/downloads/CoRe-Planner-1.0.tar.gz. CoRe-ABP prend en paramètres deux fichiers de description. Le premier contient les opérateurs et les méthodes que peut réaliser un agent, appelé le *domaine* de compétence de l'agent. Le second décrit l'état initial des croyances de l'agent ainsi que le but qu'il doit atteindre, *i.e.,* le *problème* soumis à l'agent.

Le lancement du planificateur est réalisé par la ligne de commande suivante :

```
java -jar planner.jar -d domain -p problem
```

Les options disponibles sont les suivantes :

```
-d  the path of the file containing the domain description.
-p  the path of the file containing the problems description.
-s  the name of the problem to solve. If the name is not specified all
     problems are solved.
-n  use this option to express the number of plans to find.
    This number must be greater than 1. Use -n all to find all plans.
-f  use this option if problems use negative facts.
-l  the log level for debugging. The log level parameter is included in
```

[2]Pour plus de détails sur l'implémentation, le javadoc est disponible en ligne à l'adresse http://core.imag.fr/downloads/CoRe-Planner-1.0-doc/index.html.

[3]http://java.sun.com.

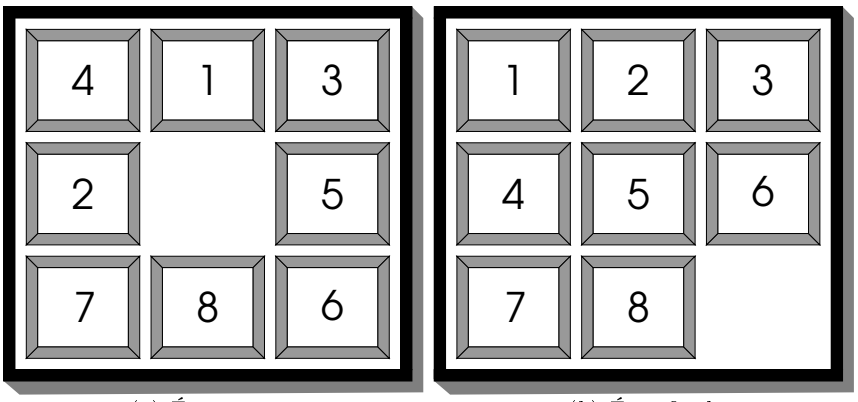
```
    range [0;10]. 0 means no output. The default value is 1.
-w  the size of the assumption window. The window parameter cannot be
    negative. The default value is 0 (i.e., no assumption must be done).
-h  Display this help message.
```

A.2 Exemples

Afin d'illustrer les mécanismes mis en œuvre par CoRe-ABP, nous proposons dans ce paragraphe de considérer deux domaines[4]. Le premier domaine que nous traitons est celui du jeu « le taquin ». Ce jeux, souvent utilisé comme exemple pour la complexité de sa résolution, nous permettra d'introduire les concepts de base nécessaires à l'écriture d'un domaine. Le second domaine traité est extrait du jeu « The Settlers ». Ce domaine nous permettra de présenter les concepts liés à l'écriture de domaine prenant en compte les croyances partielles d'un agent.

A.2.1 Exemple 1 : « le taquin »

Un taquin est un tableau rectangulaire rempli, à l'exception d'une case vide, de pièces marquées de signes (lettres, chiffres, *etc.*). On suppose ici que ces signes sont distincts deux à deux. Le jeu consiste à faire passer ce tableau d'une disposition initiale à une disposition finale par une succession de déplacements horizontaux ou verticaux de pièces adjacentes vers la case vide. Par exemple, la figure A.1 montre un taquin 3×3.



(a) État initial. (b) État final.

FIG. A.1: Exemple d'un état initial et de l'état final solution.

Il n'est pas toujours possible de passer d'une disposition quelconque à une autre. En pratique, on pose un problème de taquin en déplaçant des pièces à partir de la position de référence et en proposant à un autre joueur de reconstituer la position but. On simulera donc le jeu réel en utilisant comme départ une position obtenue

[4]L'ensemble des domaines utilisées dans ce tutoriel sont disponibles à l'adresse http://core.imag.fr.

par permutations aléatoires en N coups à partir de la position but : les mouvements autorisés étant réversibles, ceci garantit l'existence d'une solution en N coups au plus.

Description du domaine

Nous commençons par décrire le langage nécessaire à la description du jeu. Ce langage est composé uniquement de trois prédicats :

1. size(s) qui spécifie la taille du tableau.
2. on(x,y,*tile*) qui indique que la pièce *tile* est située à la position (x,y) sur le tableau de jeu. Dans la suite de ce tutoriel, nous supposons que les pièces sont identifiées par un numéro. La case vide du tableau a pour valeur 0.
3. goal(on(x,y,*tile*)) qui exprime que la position solution de la pièce *tile* a pour coordonnées (x,y).

À partir de ce langage, nous pouvons maintenant définir les quatre opérateurs qui permettent de bouger les pièces du taquin : les opérateurs !up et !down qui déplacent une pièce de haut en bas et les opérateurs !left et !right qui la déplacent de gauche et droite. Considérons à titre d'exemple l'opérateur !up qui prend en paramètres les coordonnées (x,y) de la pièce *tile* et qui ne possède aucune précondition :

```
(:operator (!up ?x ?y ?tile)
    ()
    ((on ?x ?y ?tile) (on ?x (call - ?y 1) 0))
    ((on ?x ?y 0) (on ?x (call - ?y 1) ?tile))
)
```

Le déplacement d'une pièce *tile* vers le haut nécessite, tout d'abord, de supprimer de l'état du monde les anciennes positions occupées par *tile* et la case vide. C'est ce que traduisent les deux prédicats on(x,y,*tile*) et on(x,(y-1),0). Nous appelons cet ensemble de prédicats les *effets négatifs* de l'opérateur. Puis, il est nécessaire d'ajouter à l'état du monde les nouvelles positions de la case vide et de *tile*, exprimées par les prédicats on(x,y,*0*) et on(x,(y-1),tile). Ces prédicats sont appelés les *effets positifs* de l'opérateur.

Les autres opérateurs (*i.e.,* !down, !left, !right) peuvent être décrits de manière similaire :

```
;; Déplace la pièce ?tile initialement en (?x,?y) vers le bas.
(:operator (!down ?x ?y ?tile)
    ()
    ((on ?x ?y ?tile) (on ?x (call + ?y 1) 0))
    ((on ?x ?y 0) (on ?x (call + ?y 1) ?tile))
)

;; Déplace la pièce ?tile initialement en (?x,?y) vers la gauche.
(:operator (!left ?x ?y ?tile)
```

```
    ()
    ((on ?x ?y ?tile) (on (call - ?x 1) ?y 0))
    ((on ?x ?y 0) (on (call - ?x 1) ?y ?tile))
)

;; Déplace la pièce ?tile initialement en (?x,?y) vers la droite.
(:operator (!right ?x ?y ?tile)
    ()
    ((on ?x ?y ?tile) (on (call + ?x 1) ?y 0))
    ((on ?x ?y 0) (on (call + ?x 1) ?y ?tile))
)
```

Remarque. A.1 Le mot clé call permet de faire appel aux fonctions arithmétiques ou booléennes prédéfinies dans le planificateur. L'ensemble des fonctions disponibles est donnée dans le § A.3.

Nous enrichissons notre domaine en ajoutant une méthode move-tile. Pour chaque direction de déplacement et pour chaque pièce du taquin, move-tile teste si le déplacement est possible avant de l'effectuer. Par exemple, pour bouger une pièce vers le haut, initialement positionner on(x,y,*title*), move-tile vérifie tout d'abord si la pièce est mal placée, misplaced(x,y,*tile*) et si elle peut être déplacée, moveable(x,y,*tile*,up). Si toutes ces préconditions sont satisfaites dans l'état courant alors move-tile tente de réaliser la séquence d'actions !up(x,y,*tile*) et do-plan(0,nil,*nomove*,nil). Cette séquence de deux actions est appelée la réduction de la méthode.

```
(:method (move-tile ?nomove)

    ;; Déplace une pièce vers le haut
    ((on ?x ?y ?tile) (misplaced ?x ?y ?tile) (moveable ?x ?y ?tile up))
    ((!up ?x ?y ?tile) (do-plan 0 nil ?nomove nil))

    ;; Déplace une pièce vers le bas
    ((on ?x ?y ?tile) (misplaced ?x ?y ?tile) (moveable ?x ?y ?tile down))
    ((!down ?x ?y ?tile) (do-plan 0 nil ?nomove nil))

    ;; Déplace une pièce vers la gauche
    ((on ?x ?y ?tile) (misplaced ?x ?y ?tile) (moveable ?x ?y ?tile left))
    ((!left ?x ?y ?tile) (do-plan 0 nil ?nomove nil))

    ;; Déplace une pièce vers la droite
    ((on ?x ?y ?tile) (misplaced ?x ?y ?tile) (moveable ?x ?y ?tile right))
    ((!right ?x ?y ?tile) (do-plan 0 nil ?nomove nil))

)
```

Intéressons nous maintenant plus particulièrement aux préconditions misplaced(x,y,*tile*) et moveable(x,y,*tile*,up). Ces deux préconditions définissent ce que nous nommons des axiomes.

Le sens à donner à misplaced(x,y,*tile*) est le suivant : si une pièce *tile* en position (x,y) n'est pas dans sa position solution alors misplaced(x,y,*tile*) sera évalué à faux.

```
(:- (misplaced ?x ?y ?tile)
    ((not (goal (on ?x ?y ?tile))))
)
```

L'axiome moveable(x,y,*tile*,d) vérifie qu'une pièce *tile* est dans une position (x,y) et peut être déplacée dans une direction donnée d. Le test s'effectue en vérifiant que la position de destination de la pièce est la case vide.

```
(:- (moveable ?x ?y ?tile ?d)
    ;; Test pour déplacer une pièce vers le haut
    ((same ?d up) (different ?tile 0) (on ?x (call - ?y 1) 0))
    ;; Test pour déplacer une pièce vers le bas
    ((same ?d down) (different ?tile 0) (on ?x (call + ?y 1) 0))
    ;; Test pour déplacer une pièce vers la gauche
    ((same ?d left) (different ?tile 0) (on (call - ?x 1) ?y 0))
    ;; Test pour déplacer une pièce vers la droite
    ((same ?d right) (different ?tile 0) (on (call + ?x 1) ?y 0))
)
```

Nous introduisons également deux axiomes très utiles qui permettent de comparer deux valeurs :

```
(:- (different ?x ?y) ((not (same ?x ?y))))
(:- (same ?x ?x) ())
```

Il nous reste maintenant à définir la méthode principale qui permet de résoudre le jeu du taquin, *i.e.,* do-plan. Le rôle de cette méthode est double. D'une part, elle construit l'état courant du taquin, *i.e.,* la liste des positions des différentes pièces et teste si l'état courant est l'état solution et, d'autre part, déplace les pièces mal positionnées grâce à la méthode move-tile.

La méthode do-plan possède quatre paramètres :
- *i* qui identifie le numero de la pièce courante ;
- *state* qui spécifie la liste des positions courantes des pièces ;
- *nomove* qui contient la liste des états du taquin déjà explorés ;
- *goal* qui définit la liste des positions solutions des pièces.

```
(:method (do-plan ?i ?state ?nomove ?goals)

    ;; Construit l'état initial et le but à atteindre
    (:first (on ?x ?y ?i) (size ?s) (call <= ?i ?s) (goal (on ?z ?t ?i)))
    ((do-plan (call + ?i 1) ;;the recursive call with i+1
        (list (on ?x ?y ?i) . ?state)
        ?nomove
        (list (on ?z ?t ?i) . ?goals)))
```

```
    ;; Le but est atteint
    (:first (call member ?state (list ?goals)))
        ()

    ;; Déplace une pièce
    (:first (not (call member ?state ?nomove)))
        ((move-tile (list ?state . ?nomove)))
)
```

Remarque. A.2 Le mot clé list permet de définir des listes de prédicats ou plus généralement de termes. L'opérateur « . » représente l'opérateur de concaténation et le symbole nil spécifie une liste vide.

Finalement, nous introduisons un ensemble d'opérateurs et de méthodes permettant d'exprimer le but comme une méthode particulière qui prend en paramètre la liste des d'états du monde solutions.

```
(:method (achieve-goals ?goals)
    ()
    ((assert-goals ?goals nil) (do-plan 0 nil nil nil))
)

(:method (assert-goals (list ?goal . ?goals) ?out)
    ()
    ((assert-goals ?goals (list (goal ?goal) . ?out)))
)

(:method (assert-goals nil ?out)
    ()
    ((!assert ?out))
)

(:operator (!assert ?g)
    ()
    ()
    ?g
)
```

Description du problème

Considérons le problème définit à la figure A.1. La première partie du problème décrit l'état initial, *i.e.,* les positions respectives des pièces du taquin, tandis que la seconde décrit la liste des positions solutions. Ce problème peut s'écrire de la manière suivante :

```
(defproblem tq-pb-03-01 taquin
     (
         (size 9)
```

```
          (on 1 1 4) (on 1 2 1) (on 1 3 3)
          (on 2 1 0) (on 2 2 2) (on 2 3 5)
          (on 3 1 7) (on 3 2 8) (on 3 3 6)
     )

     ((achieve-goals (list
          (on 1 1 1) (on 1 2 2) (on 1 3 3)
          (on 2 1 4) (on 2 2 5) (on 2 3 6)
          (on 3 1 7) (on 3 2 8) (on 3 3 0))
     ))
)
```

Le plan trouvé par CoRe-ABP est le suivant (cf. figure A.2) :

```
1.  (!down 1 2 1)
2.  (!right 1 1 4)
3.  (!up 2 1 2)
4.  (!left 2 2 1)
5.  (!down 1 2 4)
6.  (!right 1 1 2)
7.  (!up 2 1 1)
8.  (!left 2 2 4)
9.  (!left 2 3 5)
10. (!up 3 3 6)
```

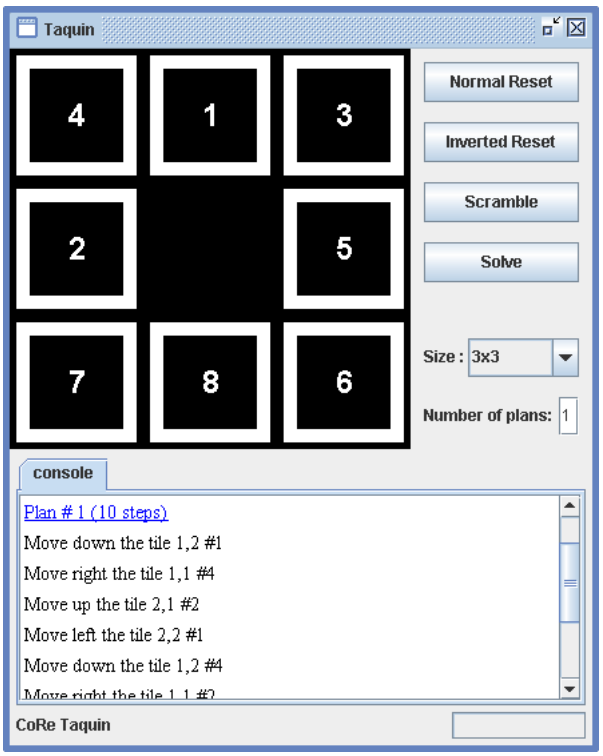


FIG. A.2: *Applet* du taquin.

Remarque. A.3 L'*applet* du jeu est accessible à l'adresse http://core.imag.fr/demo/taquin.jnlp. Elle peut être lancée depuis un terminal par la commande `javaws http ://core.imag.fr/demo/taquin.jnlp`.

A.2.2 Exemple 2 : « The Settlers »

« The Settlers » est un jeu vidéo édité par Byte Software en 1993. L'objectif de ce jeu est de développer une cité médiévale peuplée de colons. Chaque colon joue un rôle prédéfini au sein de la cité. Par exemple, les pêcheurs, les boulangers et les bouchers produisent la nourriture nécessaire aux autres colons ; les mineurs extraient les blocs de pierre, le minerai de fer ou encore le charbon indispensables à la construction des bâtiments et à la fabrication des outils.

Comme l'illustre la figure A.3, il existe des dépendances fortes entre les agents. Ceci constitue un cadre favorable pour expliciter les mécanismes de CoRe-ABP. Dans la suite de ce tutoriel, nous nous intéresserons à la description de l'agent boulanger.

Définition du domaine de compétence du boulanger

Le domaine de compétence du boulanger peut se définir de manière abstraite en s'appuyant sur les prédicats suivants :

- has-bread(b,q) : le boulanger b possède une quantité q de pain ;
- has-flour(b,q) : le boulanger b possède une quantité q de farine ;
- has-cash(b,p) : le boulanger b possède une somme p d'argent ;
- bread-price(b,p) : le prix du pain chez le boulanger b est p ;
- agent-name(b) : le nom du boulanger est b ;
- agent-village(b,v) : le village où habite le boulanger b est v ;
- is-available(g,v) : la marchandise g est disponible dans le village v ;
- receive-cash(b,a,bread,p) : le boulanger b reçoit de l'agent a la somme p pour le pain vendu ;

Nous pouvons maintenant décrire les actions primitives que peut réaliser un boulanger :

1. !make-bread(q) : le boulanger fabrique une quantité q de pain ;

```
(:operator (!make-bread ?q)
    ((has-flour ?b ?ifq) (has-bread ?b ?ibq))
    ((has-flour ?b ?ifq) (has-bread ?b ?ibq))
    ((has-flour ?b (call + ?ifq ?q)) (has-bread ?b (call + ?ibq ?q)))
)
```

2. !sell-bread(q) : le boulanger vend une quantité q de pain ;

```
(:operator (!make-bread ?q)
    ((has-flour ?b ?ifq) (has-bread ?b ?ibq))
    ((has-flour ?b ?ifq) (has-bread ?b ?ibq))
    ((has-flour ?b (call + ?ifq ?q)) (has-bread ?b (call + ?ibq ?q)))
)
```

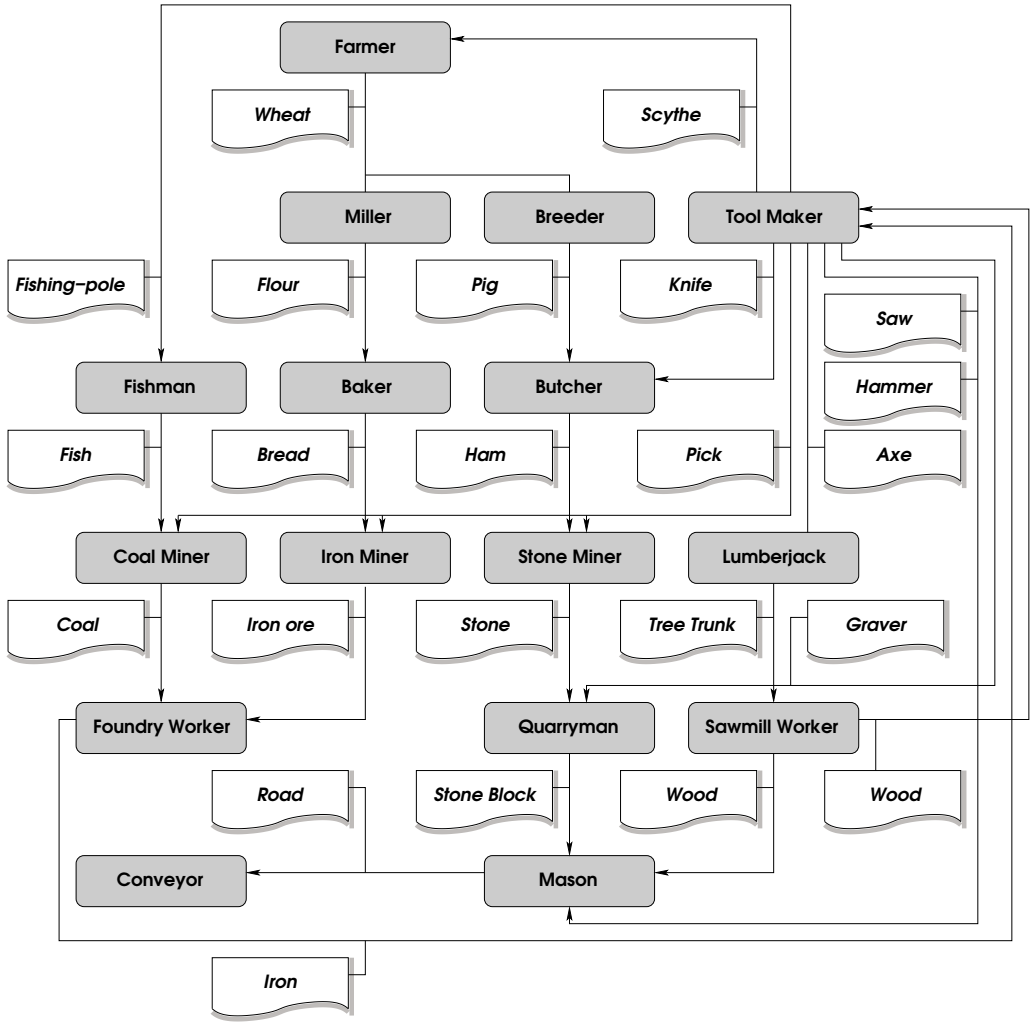


FIG. A.3: Graphe de dépendances entre les agents.

3. !pay(a,p) : le boulanger paie la somme p à l'agent a.

```
(:operator (!pay ?a ?p)
    ((has-cash ?b ?c))
    ((has-cash ?b ?c))
    ((has-cash ?b (call - ?c ?p)))
)
```

Nous introduisons, en suite, deux méthodes qui permettent d'exprimer les propriétés du monde sur lesquelles le boulanger peut raisonner de manière hypothétique (précédées par le symbole ∼) :

1. Le boulanger peut supposer que la farine dont il a besoin pour produire le pain est disponible dans son village en quantité suffisante.

```
(:method (make-bread ?q)
    (:first (has-bread ?b ?ibq)
        (agent-village ?v)
        (~is-available flour ?b ?v)
        (~has-flour ?b ?ifq)
        (~call >= ?ifq ?q))
    ((!make-bread ?q))
)
```

2. Lorsque le boulanger désire vendre du pain, il doit différencier deux cas. Soit il possède assez de pain pour vendre directement à l'agent demandeur et il suppose qu'il sera payé. Soit il ne possède pas la quantité de pain nécessaire pour réaliser la vente et il doit fabriquer le pain manquant en supposant qu'il pourra être payé.

```
(:method (sell-bread ?q ?a)

    ;; Le boulanger possède une quantité suffisante de pain
    (:first (has-bread ?b ?ibq) (call >= ?ibq ?q)
        (bread-price ?b ?p)
        (~receive-cash ?b ?a bread (call * ?q ?p)))
    ((!sell-bread ?q))

    ;; Le boulanger ne possède pas assez de pain et doit en fabriquer
    (:first (has-bread ?b ?ibq) (call < ?ibq ?q)
        (bread-price ?b ?p)
        (~receive-cash ?b ?a bread (call * ?q ?p)))
    ((make-bread (call - ?q ?ibq)) (!sell-bread ?q))
)
```

Finalement, nous définissons la méthode principale qui guide la production des plans du boulanger. Deux cas sont également à envisager. Soit le but à atteindre est de produire une certaine quantité de pain. Soit l'agent doit payer la farine qu'il a supposé posséder.

```
(:method (do-plan)
```

```
    ;; another agent a wants q breads
    (:first (goal (has-bread ?a ?q)))
    ((sell-bread ?b ?q ?a))

    ;; the baker b must pay some agent a for flour
    (:first (agent-name ?b)
        (goal (receive-cash ?a ?b flour ?p))
        (has-cash ?b ?c) (call >= ?c ?p))
    ((!pay ?a ?p))

)
```

Description du problème

En guise d'illustration, nous donnons un exemple de problème pouvant être soumis au boulanger.

```
(defproblem baker-problem baker
    (
        ;; static properties
        (agent-name baker)
        (agent-village village1)
        (bread-price baker 1)

        ;; dynamic properties
        (has-flour baker 2)
        (has-bread baker 1)
        (has-cash baker 2)

    )
    ((achieve-goals (list
        (has-bread miner 4)
    ))
)
```

Le plan trouvé pour atteindre le but has-bread(*miner*,4) formule deux hypothèses :

```
1. [ (has-flour baker 3)] (!make-bread 3)
2. [ (receive-cash baker miner 4 bread)] , (!sell-bread 4)
```

L'action !make-bread(3) suppose que le boulanger possède 3 sacs de farine alors qu'il n'en possède que 2 et l'action !sell-bread(4) suppose que le boulanger recevra la somme de 4 euros de la part du meunier.

A.3 Spécification de la syntaxe (BNF)

Ce paragraphe donne la syntaxe (Backus Naur Form) complète de CoRe-ABP. L'analyse syntaxique a été réalisée en s'appuyant sur l'outil Javacc[5].

$< letter >$ ::= ["**a**"-"**z**","**A**"-"**Z**","**_**","**-**","**#**"] ;

$< digit >$ ::= ["**0**"-"**9**"] ;

$< number >$ ::= $< digit > +$;

$< char >$ ::= $< letter > | < digit >$;

$< symbol >$ ::= $< letter > (< char > *)$;

$< alias >$ ::= $< symbol >$;

$< problem\_name >$ ::= $< symbol >$;

$< domain\_name >$ ::= $< symbol >$;

$< problem >$ ::= "(**defproblem** "
$< problem\_name >< domain\_name >$
$< initial\_state > [< task\_list >]$ ")" ;

$< initial\_state >$ ::= "("$< literal > *$ ")" ;

$< domain\_rule >$ ::= $< axiom > | < operator > | < method >$;

$< domain >$ ::= "(**defdomain** " $< domain\_name >$ "("
$< domain\_rule > *$ "))" ;

$< operator >$ ::= "(**:operator** " $< operator\_head >$
$< preconditions >< del >< add >$
$[< number >]$ ")" ;

$< preconditions >$ ::= $< conjunction >$;

$< del\_list >$ ::= $< conjunction >$;

$< add\_list >$ ::= $< conjunction >$;

$< method >$ ::= "(**:method** " $< method\_head >$
$(< decomposition > *)$ ")" ;

$< method\_head >$ ::= "(" $< symbol >(< term > *)$ ")" ;

$< operator\_head >$ ::= "(**!**" $< symbol >(< term > *)$ ")" ;

$< decomposition >$ ::= "(" $[< alias >] < preconditions >$
$< task\_list >$ ")" ;

$< task\_list >$ ::= "**nil**" | "(" $< task\_atom > *$ ")" ;

$< task\_atom >$ ::= $< operator\_head > | < method\_head >$;

[5] https://javacc.dev.java.net

$< axiom >$ ::= "(**:-**" $< literal >< conjunction\_list >$ ")" ;

$< conjunction\_list >$ ::= [$< alias >$] $< conjunction >$;

$< conjunction >$ ::= "**nil**"
| $< term >$
| "(" ["**:first**"] $< literal > *$ ")" ;

$< literal >$ ::= "(**not**" $< logical\_atom >$ ")"
| $< logical\_atom >$;

$< logical\_atom >$ ::= $< term >$
| "(" $< symbol >$ ($< term > *$) ")"
| "(~" $< term > *$ ")" ;

$< term >$ ::= $< variable >$
| $< constant >$
| $< list\_term >$
| $< call\_term >$
| $< function\_term >$;

$< variable >$ ::= " **?**"$< symbol >$;

$< constant >$ ::= $< symbol >$ | $< number >$;

$< list\_term >$ ::= "(**list**)" | "(**list**" $< term >< sublist >$ ")" ;

$< sublist >$ ::= "**.**" $< term >$ | $< term >< sublist >$;

$< call\_term >$ ::= "(**call** " $< fctn >< term >< term >$ ")"
| "(~**call** " $< bool\_fctn >< term >< term >$ ")" ;

$< function\_term >$ ::= "(" $< symbol >$ ($< term > *$) ")"
| "(~"$< term > *$ ")" ;

$< fctn >$ ::= $< bool\_fctn >$ | "*****" | "/" | "+" | "**-**" | "**cat**" ;

$< bool\_fctn >$ ::= ">"
| "<"
| ">="
| "<="
| "="
| "! ="
| "**min**"
| "**max**"
| "**member**" ;

Remarque. A.4 La syntaxe, pour commenter une ligne ou un bloc de lignes, respecte celle de C/C++/Java :

- commentaire d'une ligne ';;' or '//' ;
- commentaire d'un bloc '/*' et '*/'.

Annexe B

Tutoriel de CoRe Distributed Message Service

Introduction

L'objectif de cette annexe est de décrire les mécanismes de l'intergiciel pour agents CoRe Distributed Message Service et de fournir les connaissances nécessaires à son utilisation. CoRe-DMS est utilisé au sein du projet CoRe[1] (Pellier and Fiorino, 2005a; Pellier and Fiorino, 2005b) dont le but est la conception d'un planificateur multi-agent basé sur le partage des compétences et des croyances.

Dans un premier temps, nous donnerons la procédure d'installation de CoRe-DMS puis nous décrirons l'architecture sous-jacente. Dans un second temps, nous illustrerons son fonctionnement au travers de la programmation de deux agents. Finalement, nous aborderons les mécanismes de déploiement et d'administration associés à CoRe-DMS.

B.1 Installation de CoRe-DMS

CoRe-DMS[2] est une librairie implémentée en Java^{TM}1.5 qui nécessite une machine virtuelle java[3]. Les sources et le fichier binaire de CoRe-DMS sont disponibles à l'adresse http://core.imag.fr/downloads/CoRe-DMS-beta1.tar.gz. Le binaire est contenu dans le répertoire *lib* et porte le nom de *dms.jar*. Pour utiliser CoRe-DMS, il faut ajouter le fichier binaire *dms.jar* au répertoire de classes.

[1] CoRe est l'acronyme de Conjecture Refutation.

[2] Pour le détails de l'implémentation, le javadoc est disponible à l'adresse http://core.imag.fr/downloads/CoRe-DMS-beta1.0-doc/index.html.

[3] http://java.sun.com.

La compilation de CoRe-DMS s'effectue par l'intermédiaire de l'utilitaire `ant`[4]. Pour recompiler entièrement CoRe-DMS, il est nécessaire d'entrer la commande `ant rebuild` et `ant jar` pour régénérer le binaire *dms.jar*.

B.2 Architecture

CoRe-DMS repose sur les concepts d'*AgentServers* et d'*Agents*. Les *AgentServers* définissent le contexte d'exécution des *Agents*, *i.e.*, une machine virtuelle. Un *AgentServer* est une application Java™1.5 qui s'exécute sur une machine *physique* en tâche de fond et peut être contrôlée par une interface d'administration (cf. § B.5). Les *AgentServers*, une fois connectés entre eux, masquent complètement la distribution physique des agents en leur fournissant un ensemble de services de communication communs. La figure B.1 montre un exemple de topologie qui peut être construite grâce à CoRe-DMS.

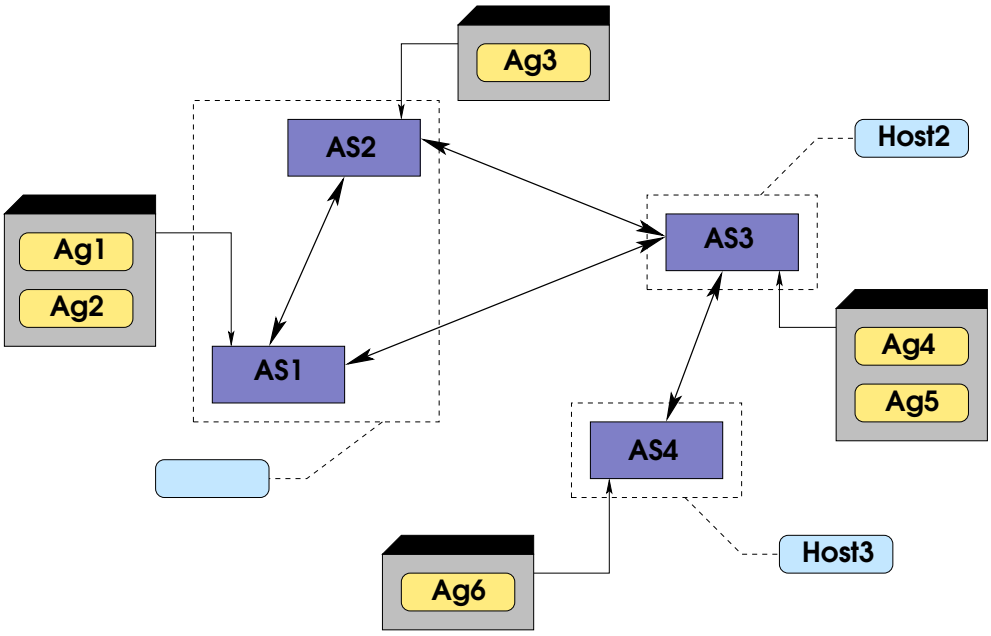


FIG. B.1: Exemple d'une topologie d'*AgentServers*.

L'architecture de CoRe-DMS est structurée en couches (cf. figure B.2). Chaque couche joue un rôle particulier dans la gestion des services rendus aux agents :

La couche Java RMI[5] représente la couche de communication de base sur laquelle est construite les *AgentServers*.

La couche *Agentservers* implémente les services de communication entre agents synchrones et asynchrones, *e.g.*, **send()**, **receive()** *etc.*

La couche *AbstractAgents* définit une abstraction de la notion d'agent et les services communs accessibles sur tous les *Agentservers*.

[4] http://ant.apache.org.
[5] « *Remote Method Invocation* »

La couche *Agents* représente la couche application, *i.e.*, le niveau d'exécution des agents sur CoRe-DMS.

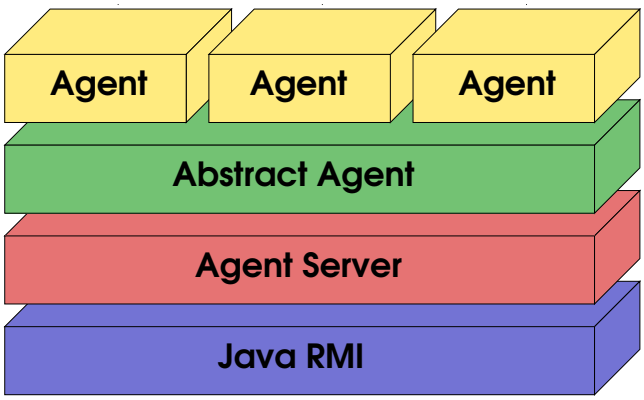


FIG. B.2: Architecture de CoRe-DMS.

La ligne de commande pour lancer un *AgentServer* est la suivante :

```
Usage AgentServer -n <name> -p <port> -c <config> -a (<agent>:<class>)*
 -n  the name of the agent server (default agent-server).
 -p  the port of the agent server (default 1250).
 -c  the configuration file. If config file is specified -n and -p are
     ignored.
 -a  (<agent>:<class>)* the agents list to launch at the agent server
     start (default: no agent is launched).
```

La configuration des *AgentServers* peut être réalisée grâce à un fichier de configuration au format XML. Ce fichier permet notamment de préciser la liste des autres *AgentServers* auquel un *AgentServer* doit se connecter lors de son initialisation, ainsi que le niveau de trace pour le débuggage des agents.

Exemple B.1 (Fichier de configuration)

```
<properties>
    <entry key="name">magma1</entry>
    <entry key="port">1251</entry>
    <entry key="entrance-servers">magma2 magma3</entry>
    <entry key="magma2">rmi://sun-magma2.imag.fr:1252/magma2</entry>
    <entry key="magma3">rmi://sun-magma3.imag.fr:1253/magma3</entry>
    <entry key="connection-mode">passing</entry>
    <entry key="connection-attempts">2</entry>
    <entry key="connection-timeout">20</entry>
    <entry key="log-file">magma1.log</entry>
    <entry key="log-level">all</entry>
</properties>
```

B.3 Un agent minimal

Intéressons nous maintenant au développement d'un agent minimal. Le code de cet agent est donné ci-dessous :

```
1  import core.dms.Agent;
2
3  public final class MinimalAgent extends Agent {
4
5      public MinimalAgent() {
6          super();
7      }
8
9      public void execute(String[] args){
10     }
11 }
```

Tout agent qui désire pouvoir être exécuté sur un *AgentServer* doit étendre la classe *Agent* et définir un constructeur (le constructeur se limite à l'appel du constructeur père) et une méthode **execute** (line 9). Cette méthode constitue la méthode principale d'un agent appelée lorsque l'agent est lancé sur l'*AgentServer*. La méthode **execute** est une méthode abstraite héritée de la classe *Agent*. Dans le cas de l'agent minimal, la methode **execute** est vide. Après sa création, l'agent s'arrêtera immédiatement.

Compilation. Pour compiler l'agent minimal développés avec CoRe-DMS, il est nécessaire d'entrer la ligne de commande suivante :

```
javac -classpath lib/dms.jar MinimalAgent.java
```

Une fois compilé, l'agent peut s'exécuter sur un *AgentServer*.

Exécution. Pour exécuter l'agent *MinimalAgent*, il est nécessaire de lancer également l'*AgentServer* qui lui sert de contexte d'exécution. Ceci est réalisé par la commande suivante :

```
java -classpath lib/dms.jar:. core.dms.AgentServer -a John:MinimalAgent
```

Cette commande lance un nouvel *AgentServer* sur lequel un *MinimalAgent* appelé John sera exécuté.

Remarque. B.1 L'option `-a` accepte des agents qui ont été préalablement compilés, *i.e.,* des classes. Par conséquent, les agents mentionnés doivent être accessibles depuis le répertoire de classes spécifié par la ligne de commande. Dans notre exemple, nous supposons que *MinimalAgent* est présent dans le répertoire courant.

B.4 Exemple : ChatAgent

Le paquetage *dms.jar* est composé d'un sous-paquetage appelé **core.samples.dms** qui contient un certain nombre d'exemples d'utilisation de CoRe-DMS dont une application de Chat. Cette application peut être lancée par la commande :

```
java -classpath lib/dms.jar core.samples.dms.chat.ChatServer
```

La classe *ChatServer* est un *AgentServer* qui execute deux *ChatAgents* appelés John et Bob. Le code de *ChatServer* est équivalent à la ligne de commande suivante :

```
java -classpath lib/dms.jar core.dms.AgentServer \
    -n ChatServer \
    -a John:core.samples.dms.ChatAgent Bob:core.samples.dms.ChatAgent
```

L'option `-n` spécifie le nom de l'*AgentServer* et l'option `-a` permet de préciser le nom ainsi que la classe des agents à exécuter sur l'*AgentServer*.

```
48 public final class ChatServer {
49
50     /**
51      * The port of the <tt>ChatServer</tt>.
52      */
53     private static final int PORT = 1223;
54
55     /**
56      * Create a new <tt>ChatServer</tt>.
57      */
58     private ChatServer() {
59
60         // Create the AgentServerProperties
61         final AgentServerProperties properties =
62                         new AgentServerProperties();
63
64         // Set the name of the AgentServer
65         properties.setName("ChatServer");
66
67         // Set the port of the AgentServer
68         properties.setPort(PORT);
69
70         // Create the AgentServer with specified properties
71         AgentServer as = null;
72         try {
73             as = new AgentServer(properties);
74         } catch (Exception e) {
75             System.out.println("Error creating AgentServer");
76             e.printStackTrace();
77             System.exit(1);
78         }
79
80         // Bind the AgentServer with its listener
81         as.addAgentServerListener(new AgentServerEventListener());
82
```

```
83          // Start the AgentServer
84          try {
85              as.start();
86          } catch (Exception e) {
87              System.out.println("Error starting AgentServer");
88              e.printStackTrace();
89              System.exit(1);
90          }
91
92          // Launch Agent on the AgentServer
93          try {
94              as.launchAgent("John", new ChatAgent());
95              as.launchAgent("Bob", new ChatAgent());
96          } catch (Exception e) {
97              System.out.println("Error lauching agent");
98              e.printStackTrace();
99              System.exit(1);
100         }
101     }
```

Regardons maintenant la méthode **execute** des agents *ChatAgent* :

```
384     public void execute(final String[] args) {
401         broadcast("join", new Object[0]);
402         while (!shouldStop()) {
403             final Message m = receiveMessageBlocking();
404             handleMessage(m);
405         } // end while
407     }
```

Cet exemple présente une partie de l'API fournie par CoRe-DMS[6] :

broadcast permet de diffuser un message sous la forme d'une chaîne de caractères à laquelle il est possible d'attacher un tableau d'objets.

shouldStop teste si l'agent a demandé à mettre fin à son exécution avec la méthode **askToStop**.

receiveMessageBlocking lit le message suivant dans la file d'entrée des messages de l'agent. Si la file est vide, l'agent est bloqué tant qu'un message ne lui est pas parvenu.

handleMessage effectue le traitement des messages.

Remarque. B.2 L'API de CoRe-DMS fournit également les primitives de communication asynchrones. Il est important de noter que la réception des messages est effectuée dans l'ordre causal de leur émission. Cette propriété est implémentée au sein des *AgentServers* en s'appuyant sur des matrices de temps logiques.

[6] La description complète de l'API de CoRe-DMS est disponible à l'adresse http://core.imag.fr/downloads/CoRe-DMS-beta1.0-doc/index.html

B.5 Administration des *AgentServers*

CoRe-DMS fournit une interface d'administration. Celle-ci permet de déployer, de lancer et de stopper l'exécution des agents à distance sur les différents *AgentServers*. Supposons qu'un *AgentServer* s'exécute sur une machine :

```
java -classpath lib/dms.jar core.dms.AgentServer -n server
```

À cet instant aucun agent ne s'exécute sur l'*AgentServer*. Nous désirons maintenant lancer un *Agent* John sur l'*AgentServer*. Pour effectuer cette opération, il est nécessaire de lancer la console d'administration contenue dans le paquetage **core.dms.console**.

La console d'administration (cf. figure B.3) permet de se connecter aux différents *AgentServers* afin de contrôler l'exécution des agents. La console est lancée avec la ligne de commande :

```
java -classpath /lib/dms.jar:/lib/scl.jar core.dms.console.Console
```

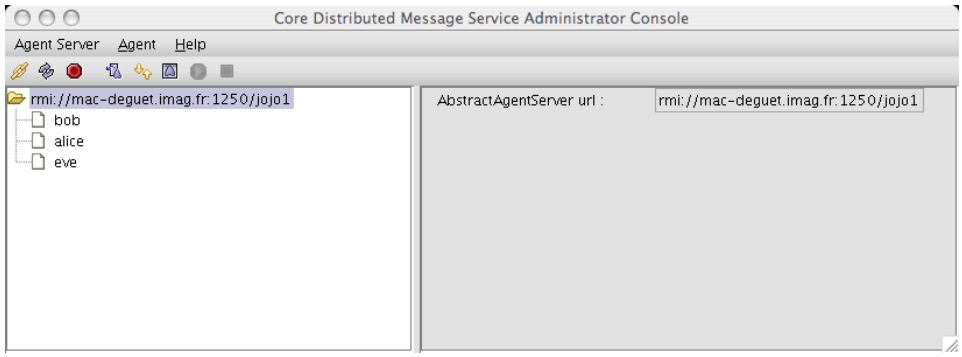
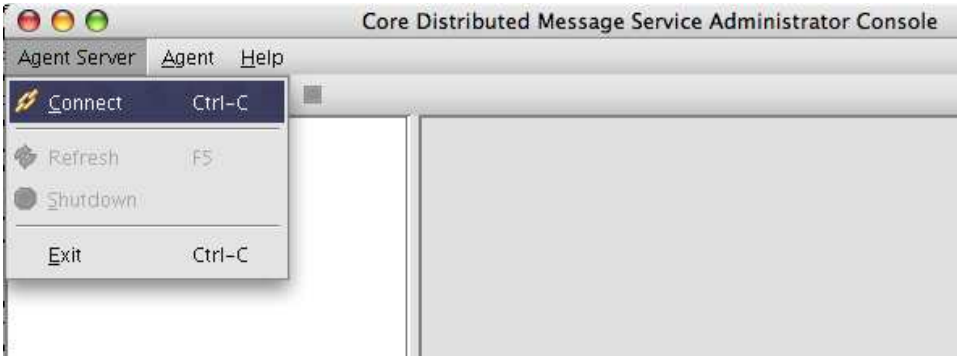


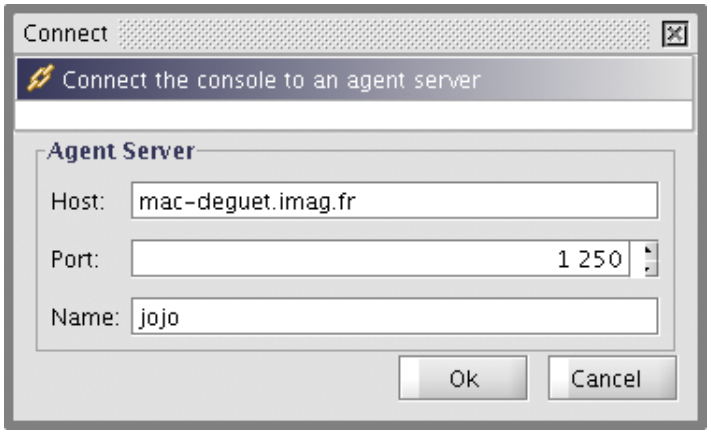
FIG. B.3: Console d'administration de CoRe-DMS.

B.5.1 Connexion

La première opération à réaliser est de connecter la console à un *AgentServer*. Cette tâche est accessible via le menu Agent Server→Connect (cf. figure B.4(a)). Une fenêtre de dialogue (cf. figure B.4(b)) s'ouvre alors et permet de préciser l'adresse de l'*AgentServer* auxquels la console d'administration doit se connecter, *i.e.,* le nom de *AgentServer*, l'adresse de machine sur lequel il s'exécute ainsi que le port utilisé (le port par défaut est 1250). Lorsque la connexion est établie, la liste des *AgentServers* présents sur le réseau ainsi que l'ensemble des *Agents* actuellement lancés sont affichés dans la fenêtre principale de la console.



(a) Menu de connexion.



(b) Dialogue de connexion.

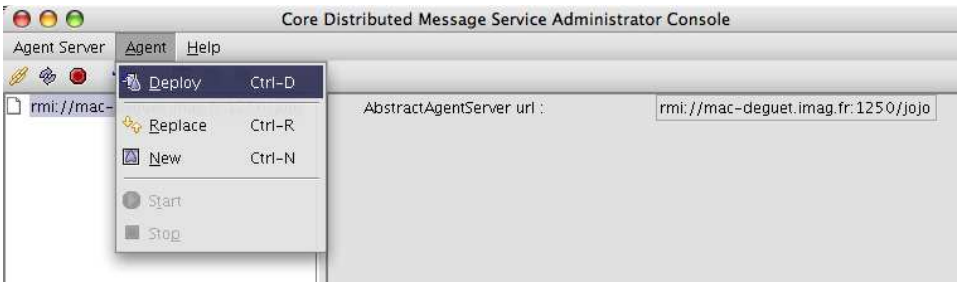
B.5.2 Déploiement

Le déploiement des agents sur les différents *AgentServers* s'effectue en utilisant le menu Agent→Deploy (cf. figure B.4(c)). Le dialogue (cf. figure B.4(d)) s'ouvre et il est alors nécessaire de préciser les classes ou le fichier jar contenant les fichiers binaires de l'agent à déployer.

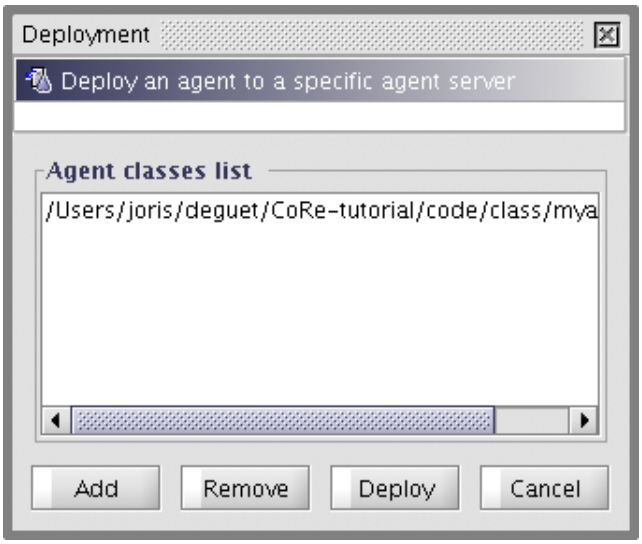
Remarque. B.3 Lorsqu'un *Agent* est en cours d'exécution sur un *AgentServer*, l'opération de déploiement remplace dynamiquement le contexte d'exécution en utilisant les nouvelles classes déployées de l'agent. Ceci est une fonctionnalité intéressante qui permet de mettre à jour les agents sans être, à aucun moment, obligé d'arrêter le système.

B.5.3 Création

Après avoir déployé le code des agents sur un *AgentServer*, il est possible de créer des agents en utilisant ce code. La création d'un agent est accessible par le



(c) Menu de déploiement.



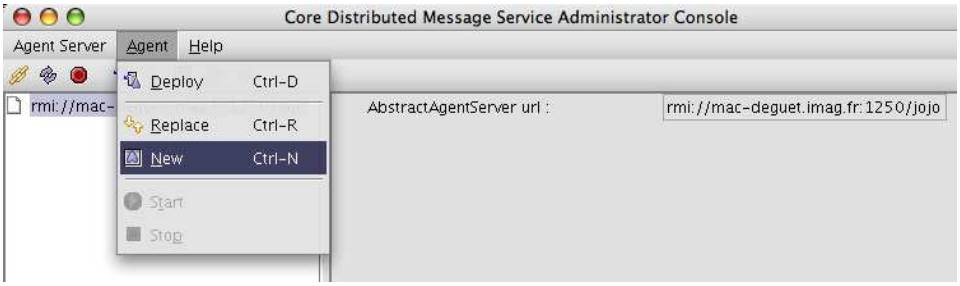
(d) Dialogue de déploiement.

Agent→Create (cf. figure B.4(e)). Le dialogue (cf. figure B.4(e)) s'affiche et il faut alors renseigner le nom de l'agent à créer ainsi que le nom de sa classe principale.

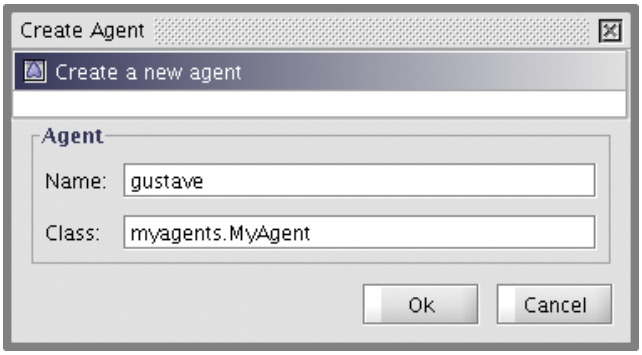
Remarque. B.4 Il est possible de créer autant d'instances d'agents souhaitées à partir d'une même classe.

B.5.4 Exécution

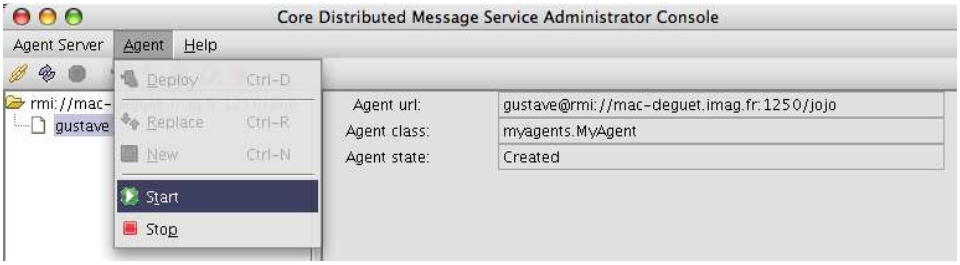
La dernière étape consiste à débuter l'exécution d'un agent sur un *AgentServer*. L'opération est disponible dans le menu Agent→Start (cf. figure B.4(g)). Une fenêtre de dialogue (cf. figure B.4(h)) demande alors de préciser les éventuels paramètres nécessaires à l'exécution de l'agent.



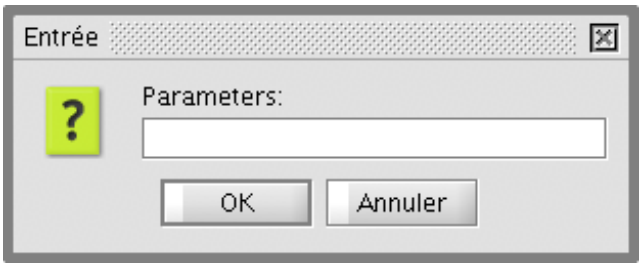
(e) Menu de création.



(f) Dialogue de création.



(g) Menu de lancement.



(h) Dialogue de lancement.

Bibliographie

Adler, M., Davis, A., Weihmayer, R., and Worres, R. (1989). Conflict-resolution strategies for nonhierarchical distributed agents. In Gasser, L. and Huhns, M., editors, *Distributed Artificial Intelligence*, volume 2, pages 139–162. Morgan Kaufmann Publishers, San Mateo, CA, USA.

Alami, R., Fleury, S., Herrb, M., Ingrand, F., and Robert, F. (1998a). Multi robot cooperation in the martha project. *IEEE Robotics and Automation Magazine*, 5(1) :36–47.

Alami, R., Ingrand, F., and Qutub, S. (1998b). A scheme for coordinating multi-robots planning activities and plans execution. In Prade, H., editor, *Proceedings of European Conference on Artificial Intelligence*, pages 617–621, Brighton, England. John Wiley & Sons, Chichester.

Alami, R., Ingrand, F., and Suzuki, S. (1994). A paradigm for plan-merging and its use for multi-robot cooperation. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, pages 612–617, San Antonio, USA.

Alami, R., Ingrand, F., and Suzuki, S. (1995). Multi-robot cooperation through incremental plan-merging. In *Proceedings of International Conference on Robotics and Automation*, pages 2573–2579, Nagoya, Japan.

Allen, J. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23(2) :123–154.

Amgoud, L., Belabbes, S., and Prade, H. (2005). Towards a formal framework for the search of a consensus between autonomous agents. In *Proceedings of the International Joint Conference on Autonous Agents and MultiAgent Systems*, pages 537–543, Utrecht, Netherland. ACM Press.

Amgoud, L. and Cayrol, C. (2002). A reasoning model based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence*, 34(1-3) :34–197.

Amgoud, L., Maudet, N., and Parson, S. (2000). Modelling dialogues using argumentation. In Durfee, E., editor, *Proceedings of the International Conference on Multiagent Systems*, pages 31–38, Boston, USA. IEEE Press.

Amgoud, L., Maudet, N., and Parson, S. (2002). An argumentation-based semantics for agent communication languages. In *Proceedings of the European Conference on Artificial Intelligence*, pages 38–42, Lyon, France. IOS Press.

Austin, J. (1962). *How to do things with words*. Harvard University Press, Cambridge, Massachusetts, USA, 2nd edition.

Barth, E. and Krabbe, E. (1982). *From Axiom to Dialogue : A Philosophical Study of Logics and Argumentation.* Walter de Gruyter, Berlin, Germany.

Bench-Capon, T. (1998). Specification and implementation of toulmin dialogue game. In *Proceedings of the International Conference on Legal Knowledge-Based Systems*, pages 5–20.

Blum, A. and Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2) :281–300.

Bondarenko, A., Dung, P., Kowalski, R., and Toni, F. (1997). An abstract argmentation-theory approach to default reasoning. *Artificial Intelligence*, 93(1-2) :63–101.

Boussetta, S., El-Fallah-Seghrouchni, A., Haddad, S., Moriatis, P., and Taghelit, M. (1998). Coordination d'agents rationnels par planification distribuée. *Revue d'Intelligence Artificielle*, 12(1).

Bratman, M. (1992). Shared cooperative activity. *The Philosophical Review*, 101(2) :327–341.

Brewka, G. (1994). A logical reconstruction of rescher's theory of formal disputation based on default logic. In Cohn, A., editor, *Proceedings of the European Conference on Artificial Intelligence*, pages 366–370, Amsterdam, The Netherlands. John Wiley and Sons.

Brewka, G. (2001). Dynamic argument systems : a formal model of argumentation processes based on situation calculus. *Journal of Logic and Computation*, 11(2) :257–282.

Briggs, W. and Cook, D. (1995). Flexible social laws. In Mellish, C., editor, *Proceedings of the fourteenth International Joint Conference on Artificial Intelligence*, pages 688–693, Montréal, Québec, Canada.

Cammarata, S., McArthur, D., and Steeb, R. (1983). Strategies of cooperation in distributed problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 767–770, Karlsruhe, Germany.

Castelfranchi, C. (1998). Modelling social action for AI agents. *Artificial Intelligence*, 103(1-2) :157–182.

Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, 32(3) :333–377.

Chevaleyre, Y., Endriss, U., Lang, J., and Maudet, N. (2005). Negotiating over small bundles of resources. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 296–302. ACM Press.

Clement, B. and Barrett, A. (2003). Continual coordination through shared activities. In *Proceedings of the International Conference on Autonomous Agent and Muti-Agent Systems*, pages 57–67.

Cohen, P. and Levesque, H. (1990). Intention is choice with commitment. *Artificial Intelligence*, 42(3) :213–261.

Cohen, P. and Levesque, H. (1991). Confirmations and joint action. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 951–959, Sydney, Australia. Morgan Kaufmann Publishers.

Corkill, D. (1979). Hierarchical planning in a distributed environment. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 168–175, Tokyo, Japan. Morgan Kaufmann Publishers.

Corkill, D. and Lesser, V. (1983). Use of meta-level control for coordination in a distributed problem solving network. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 748–756, Karlsruhe, Germany. Morgan Kaufmann Publishers.

Corkill, D., Lesser, V., and Hudlicka, E. (1982). Unifying data-directed and goal-directed control : An example and experiments. In *Proceedings of National Conference on Artificial Intelligence*, pages 143–147, Pittsburgh, Pennsylvania, USA.

Davis, R. and Smith, R. (1983). Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1) :63–109.

de Weerdt, M. (2003). *Plan Merging in Multi-Agent Systems.* PhD thesis, Delft University of Technology, Delft, The Netherlands.

Decker, K. (1996). Tæms : A framework for environment centred analysis and design of coordination mechanisms. In O'Hare, G. and Jennings, N., editors, *Foundations of Distributed Artificial Intelligence*, chapter 16, pages 429–448. Wiley Inter-Sciences.

Decker, K. and Lesser, V. (1992). Generalizing the partial global planning algorithm. *International Journal of Intelligence and Cooperative Information Systems*, 1(2) :319–346.

DesJardins, M. and Wolverton, M. (1999). Coordinating a distributed planning system. *Artificial Intelligence Magazine*, 20(1) :45–53.

D'Inverno, M., Luck, M., Georgeff, M., Kinny, D., and Wooldridge, M. (2004). The dmars architecture : A specification of the distributed multi-agent reasoning system. *Autonomous Agents and Multi-Agent Systems*, 9(1-2) :5–53.

Dung, P. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2) :321–358.

Dung, P., Kowalski, R., and Toni, F. (2003). Argumentation-theoretic proof procedures for default reasoning. Technical report, Imperial College, 180, Queen's Gate, London, England.

Durfee, E. (2001). Distributed problem solving and planning. In *Lecture Notes in Computer Science*, pages 118–149. Springer Verlag Publishers.

Durfee, E. and Lesser, V. (1987). Using partial global plans to coordinate distributed problem solvers. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 875–883, Milan, Italy.

Durfee, E. and Lesser, V. (1989). *Negotiating task decomposition and allocation using partial global planning*, volume 2, chapter 3, pages 229–244. Morgan Kaufmann Publishers.

Durfee, E. and Lesser, V. (1991). Partial global planning : A coordination framework for distributed hypothesis formation. In *IEEE Transactions on Systems, Man, and Cybernetics*, volume 21, pages 1167–1183.

El-Fallah-Seghrouchni, A. and Haddad, S. (1996a). A coordination algorithm for multi-agent planning. In *Lecture Notes in Computer Science*, volume 1038, pages 86–99. Springer Verlag Publisher.

El-Fallah-Seghrouchni, A. and Haddad, S. (1996b). A recursive model for distributed planning. In *Proceedings of the International Conference on Multi-Agent Systems*, Kyoto, Japan.

El-Fallah-Seghrouchni, A., Moraitis, P., and Tsoukiàs, A. (2000). An aggregation-disaggregation approach for automated negotiation in multi-agent systems. In *Proceedings of the Conference on Multi-Agents and Mobile Agents in Virtual Organisation and E-Commerce*, Wollongong, Australie.

Ephrati, E. and Rosenschein, J. (1991). The clarke tax as a consensus mechanism among automated agents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 173–178, Anaheim, California, USA.

Erman, L., Hayes-Roth, F., Lesser, V., and Reddy, D. (1980). The hearsay-II speech-understanding system : Integrating knowledge to resolve uncertainty. *Computing Survey*, 12(2) :213–253.

Erol, K., Hendler, J., and Nau, D. (1994). HTN planning : Complexity and expressivity. In *Proceedings of the National Conference on Artificial Intelligence*, volume 2, pages 1123–1128, Seattle, WA, USA.

Erol, K., Hendler, J., and Nau, D. (1996). Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1) :69–93.

Ferber, J. (1995). *Les systemes multi-agents, vers une intelligence collective.* Paris InterEditions.

Ferguson, G. and Allen, J. (1994). Arguing about plans : Plan representation and reasoning for mixed-initiative planning. In Hammond, K., editor, *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, pages 43–48.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML as an agent communication language. In Adam, N., Bhargava, B., and Yesha, Y., editors, *Proceedings of the International Conference on Information and Knowledge Management*, pages 456–463, Gaithersburg, MD, USA. ACM Press.

Finke, R. and Nilsson, N. (1971). STRIPS : A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 3-4(2) :189–208.

Fiorino, H. (1999). État de l'art sur la coopération multi-robot. Technical Report 99237, Laboratoire d'Analyse et des Architectures des Systèmes, Toulouse, France.

Firby, J. (1987). An investigation into reactive planning in complex domains. In *Proceedings of the National Conference on Artificial Intelligence*, pages 202–206, Seattle, Washington, USA.

Garcia, F. (1993). *Révision des croyances et révision du raisonnement pour la planification.* PhD thesis, École Nationale Supérieur de l'Aéronotique et de l'Espace.

Gasser, L., Braganza, C., and Herman, N. (1987). MACE : A flexible testbed for distributed ai research. In Huhns, M., editor, *Distributed Artificial Intelligence*, pages 119–152, San Mateo, CA, USA. Morgan Kaufmann Publishers.

Gasser, L., Braganza, C., and Herman, N. (1988). Implementing distributed AI systems using MACE. In Bond, A. and Gasser, L., editors, *Readings in Distributed Artificial Intelligence*, pages 445–450. Morgan Kaufmann Publishers, San Mateo, CA, USA.

Geffner, H. (1992). *Default reasoning : causal and conditional theorie*. MIT Press, Cambridge, MA, USA.

Georgeff, M. (1983). Communication and interaction in multi-agent planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 125–129.

Gerevini, A. and Schubert, L. (1996). Accelerating partial-order planners : Some techniques for effective search control and pruning. *Journal of Artificial Intelligence Research*, 5(1) :95–137.

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning Theory and Practice*. Morgan Kaufmann Publishers.

Gilkinson, H., Paulson, S., and Sikkink, D. (1954). Effects of order and authority in an argumentative speech. *Quarterly Journal of Speech*, 40(1) :183–192.

Gordon, T. (1994). The pleadings game : an exercise in computational dialectics. *Artificial Intelligence and Law*, 2(4) :239–292.

Gordon, T. (1995). *The Pleadings Game. An Artificial Intelligence Model of Procedure Justice*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Gregoire, E. (1999). Gestion efficace de l'inconsistance dans les bases de connaissances stratifiées. In *Actes des Journée Nationale sur la Résolution Pratique des Problèmes NP-complets*, pages 121–127.

Grosz, B., Hunsberger, L., and Kraus, S. (1999). Planning and acting together. *The AI Magazine*, 20(4) :23–34.

Grosz, B. and Kraus, S. (1993). Collaborative plans for group activities. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 367–375, Chambéry, France.

Grosz, B. and Kraus, S. (1996). Collaborative plans for complex group action. *Artificial Intelligence*, 86(2) :269–357.

Grosz, B. and Kraus, S. (1999). The evolution of sharedplans. In Rao, A. and Wooldridge, M., editors, *Foundations and Theories of Rational Agency*, pages 227–262. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Grosz, B. and Sidner, C. (1990). Plans for discourse. In P. Cohen, J. M. and Pollack, M., editors, *Intentions in Communication*. MIT Press, Cambridge, MA, USA.

Hage, J. (1996). A theory of legal reasoning and logic to match. *Artificial Intelligence and Law*, 4(3-4) :199–273.

Hamblin, C. (1971). Mathematical models of dialogue. *Theoria*, 37(1) :130–155.

Holzmann, G. (1991). *Design and Validation of Computer Protocols*. Prentice Hall, New Jersey.

Horty, J. (2000). Argument construction and reinstatement in logics for defeasible reasoning. *Artificial Intelligence and Law*, 9(1) :1–28.

Huhns, M. and Bridgeland, D. (1991). Multiagent truth maintenance. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6) :1437–1445.

Jennings, N. (1993). Commitments and conventions : the foundation of coordination in multi-agent systems. *The knowledge Engineering Review*, 8(3) :223–250.

Jennings, N. (1995). Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2) :195–240.

Jennings, N. (2001). Automated negotiation : prospects, methods and challenges. *International Journal of Group Decision and Negociation*, 10(2) :199–225.

Jennings, N. and Wittig, T. (1992). ARCHON : Theory and practice. In Avouris, N. and Gasser, L., editors, *Distributed Artificial Intelligence : Theory and Praxis*, pages 179–195. Kluwer Academic Publishers.

Kaelbling, L., Littman, M., and Cassandra, A. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2) :99–134.

Karunatillake, N., Jennings, N., Rahwan, I., and Norman, T. (2005). Arguing and negotiating in the presence of social influences. In *Proceedings of the International Central and Eastern European Conference on Multi-Agent Systems*, pages 223–235. Springer Verlag Publishers.

Ketchpel, S. (1994). Forming coalitions in the face of uncertain rewards. In *Proceedings of the National Conference on Artificial Intelligence*, volume 1, pages 414–419, Menlo Park, CA. AAAI Press.

Kraus, S., Sycara, K., and Evenchick, A. (1998). Reaching agreements through argumentation : a logical model and implementation. *Artificial Intelligence*, 104(1-2) :1–69.

Laborie, P. (1995). *IxTet : Une approche intégrée pour la gestion de ressources et la synthèse de plans*. PhD thesis, École Nationale Supérieur des Télécommunications.

Labrou, Y. and Finin, T. (1997). Semantics and conversations for an agent communication language. In Pollack, M., editor, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 584–591, Nagoya, Japan. Morgan Kaufmann Publishers.

Lakatos, I. (1976). *Proofs and Refutations : The Logic of Mathematical Discovery*. Cambridge University Press, Cambridge, England.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communication of the ACM*, 21(7) :558–565.

Lesser, V. (1998). Reflections on the nature of multi-agent coordination and its implications for an agent architecture. *Autonomous Agents and Multi-Agent Systems*, 1(1) :89–111.

Lesser, V. and Corkill, D. (1981). Functionally accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1) :81–96.

Lesser, V. and Corkill, D. (1983). The distributed vehicle monitoring testbed : A tool for investigating distributed problem solving network. *Artificial Intelligence Magazine*, 42(3) :15–33.

Lesser, V., Corkill, D., Hernandez, J., and Whitehaire, R. (1989). Focus of control through goal relationships. In *Proceedings of International Joint Conference on Artificial Intelligence*, volume 1, pages 497–503.

Levesque, H., Cohen, P., and Nunes, J. (1990). On acting together. In *Proceedings of the American Association for Artificial Intelligence*, pages 94–99, Boston, MA, USA. MIT Press.

Lewis, D. (1969). *Convention : A Philosophical Study.* Harvard University Press, Cambridge, Massachusetts, USA.

Lochbaum, K. (1995). The use of knowledge preconditions in language processing. In Mellish, C., editor, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1260–1266, San Francisco, CA, USA. Morgan Kaufmann Publishers.

Lochbaum, K. (1998). A collaborative planning model of intentional structure. *Computational Linguistics*, 24(2) :525–572.

Lodder, A. (1999). Dialaw on legal justification and dialogical models of argumentation. In *Law and Philosophy Library.* Kluwer Academic Publishers, Dordrecht, The Netherland.

Lorenzen, P. (1987). Dialogical foundations of logical calculus. In *Constructive Philosophy*, chapter 6, pages 78–101. University of Massachussets Press, Amherst.

Loui, R. (1987). Defeat among arguments : A system of defeasible inference. *Computational Intelligence*, 3(1) :100–107.

Lynch, N. (1996). *Distributed Algorithms.* Morgan Kaufmann Publishers, San Francisco, CA, USA.

MacKenzie, J. (1979). Question-begging in non-cumulative systems. *Journal of Philosophical Logic*, 8(1) :117–133.

Mandiau, R. (1993). *Contribution à la modélisation des univers multi-agents : génération d'un plan partagé.* PhD thesis, Université de Valenciennes et du Hainaut Cambresis.

Marc, F., El-Fallah-Seghrouchni, A., and Degirmenciyan-Cartault, I. (2003). Distributed coordination based on temporal planning for tactical aircraft simulation. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1060–1061, Melbourne, Victoria, Australia. ACM.

Martial, F. V. (1992). *Coordinating plans for autonomous agents.* Springer Verlag Publishers.

Maudet, N. (2001). *Modéliser les conventions des interactions langagières : la contribution des jeux de dialogue.* PhD thesis, Université Paul Sabatier, Toulouse, France.

McCarthy, J. (1977). Epistemological problems of artificial intelligence. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1038–1044, Cambridge, MA.

McKenzie, J. (1990). Four dialogue systems. *Studia Logica*, 4(1) :567–583.

Moraitis, P. and Tsoukias, A. (1996). A multicriteria approach for distributed planning and conflict resolution for multiagent systems. In *Proceedings of the International Conference on Multiagent Systems*, pages 212–219, Kyoto, Japan.

Moraitis, P. and Tsoukias, A. (1999). Dynamic planning model for agent's preferences satisfaction. In *Proceedings of the Conference of Intelligent Agent Technology*, Hong Kong.

Morge, M. (2005). *Système dialectique multi-agents pour l'aide à la concertation*. PhD thesis, École Supérieure des Mines de Saint Étiennes.

Nash, J. (1950). The bargaining problem. *Econometria*, 18(1) :155–162.

Nau, D., Au, T., Ilghami, O., Kuter, U., Murdock, W., Wu, D., and Yaman, Y. (2003). SHOP2 : An HTN planning system. *Journal of Artificial Intelligence Research*, 20(1) :379–404.

Newell, A. (1992). Unified theories of cognition and the role of soar. In Michon, J. and Akyrek, A., editors, *SOAR : A Cognitive Architecture in Perspective - A Tribute to Allen Newell*, volume 10, pages 25–79. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Nilsson, N. (1980). *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers, San Francisco, CA, USA.

Nute, D. (1997). *Defeasible deontic logique*, volume 263 of *Synthese Library*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Ortiz, C. and Grosz, B. (2002). Interpreting information requests in context : A collaborative web interface for distant learning. *Autonomous Agents and Multi-Agent Systems*, 5(4) :429–465.

Parson, S., Sierra, C., and Jennings, N. (1998). Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3) :261–292.

Paterson, M. and Wegman, M. (1976). Linear unification. In *Proceedings of the annual ACM symposium on Theory of computing*, pages 181–186, New York, NY, USA. ACM Press.

Pearl, J. (1984). *Heuristics : intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Pellier, D. and Fiorino, H. (2004). Assumption-based planning. In *Proceedings of the International Conference on Advances in Intelligence Systems Theory and Applications*, pages 367–376, Luxembourg-Kirchberg, Luxembourg.

Pellier, D. and Fiorino, H. (2005a). Dialectical theory for multi-agent assumption-based planning. In *Proceedings of the International Central and Eastern European Conference on Multi-Agent Systems*, pages 367–376, Budapest, Hungary. Springer Verlag Publishers.

Pellier, D. and Fiorino, H. (2005b). Multi-agent assumption-based planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1717–1718, Edinburgh, Scotland.

Penberthy, J. and Weld, D. (1992). UCPO : A sound, complete, partial order planner for ADL. In B. Nebel, C. R. and Swartout, W., editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–114. Morgan Kaufmann Publishers.

Peot, M. and Smith, D. (1992). Conditional nonlinear planning. In Hendler, J., editor, *Proceedings of the International Conference on AI Planning Systems*, pages 189–197, College Park, Maryland. Morgan Kaufmann Publishers.

Perriolat, F., Skarek, P., Varga, L., and Jennings, N. (1996). Using ARCHON, part 3 : Particle accelerator control. *IEEE Expert : Intelligent Systems and Their Applications*, 11(6) :80–86.

Petrick, R. and Bacchus, F. (2004). Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 2–11.

Pitt, J., Kamara, L., Sergot, M., and Artikis, A. (2005). Formalization of a voting protocol for virtual organizations. In *Proceedings of the Internation Joint Conference on Autonous Agents and MultiAgent Systems*, pages 372–380. ACM Press.

Pitt, J. and Mamdanie, A. (1999). A protocol based semantics for an agent communication language. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 486–491. Morgan Kaufmann Publishers.

Piwek, P. (2000). Imperatives, commitment and action : Towards a constraint-based model. *Computational Linguistics and Language Technology*, 1-2(17) :140–165.

Plantin, C. (1990). *Essais sur l'argumentation : Introduction linguistique à l'étude de la parole argumentative.* Kim, Paris, France, kimé edition.

Pollack, M. (1990). Plans as complex mental attitudes. In Cohen, P., Morgan, J., and Pollack, M., editors, *Intentions in Communication*, pages 77–103. MIT Press, Cambridge, MA, USA.

Pollock, J. (1987). Defeasible reasoning. *Cognitive Science*, 11(4) :481–518.

Pollock, J. (1992). How to reason defeasibly. *Artificial Intelligence*, 57(1) :1–42.

Pollock, J. (1995). *Cognitive Carpenty. A blueprint for how to build a Person.* MIT Press, Cambridge, MA, USA.

Prakken, H. (1995). From logic to dialectics in legal argument. In *Proceedings of the International Conference on Artificial Intelligence and Law*, pages 165–174, College Park, Maryland, USA. ACM Press.

Prakken, H. (1997). *Logical Tools for Modelling Legal Argument : A Study of Defeasible Reasoning in Law.* Kluwer Academic Publishers.

Prakken, H. (2000). On dialogue systems with speech acts, arguments, and conterarguments. In *Lecture Notes in Computer Science*, pages 224–238. Springer Verlag Publishers.

Prakken, H. (2001a). Modelling reasoning about evidence in legal procedure. In *Proceedings of the International Conference on Artificial Intelligence and Law*, pages 119–128, St. Louis, Missouri, USA. ACM Press.

Prakken, H. (2001b). Relating protocols for dynamic dispute with logics for defeasible argumentation. *Synthese*, 127(1-2) :187–219.

Prakken, H. and Santor, G. (1996). A dialectical model of assessing conflicting arguments in legal reasoning. *Artificial Intelligence and Law*, 4(3-4) :331–368.

Prakken, H. and Vreeswijk, G. (2002). Logics of defeasible argumentation. In Gabbay, D. and Guenthner, F., editors, *Handbook of Philosophical Logic*, volume 4, chapter 3. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Pynadath, D., Tambe, M., Chauvat, N., and Cavedon, L. (1999). Toward team-oriented programming. In *Proceedings of the International Workshop on Intelligent Agents, Agent Theories, Architectures, and Languages*, pages 233–247. Springer-Verlag Publishers.

Qutub, S. (1997). How to solve deadlock situations within the plan-merging paradigm for multi-robot cooperation. In *Prodeedings of the International Conference on Intelligent Robots and Systems*, pages 1610–1615, Grenoble, France.

Rao, A. S., Lucas, A., Morley, D., Selvestrel, M., and Murray, G. (1993). Agent oriented architecture for air-combat simulation. Technical report, The Australian Artificiel Intelligemce Institute. Note 42.

Rescher, N. (1977). *Dialectics : a controversy-oriented approach to the theory.* State University of New-York Press, Alabany, New-York, USA.

Robert, F. (1996). *Coopération multi-robots par insertion incrémentale de plans.* PhD thesis, Intstitut National Polytechnique de Toulouse, France.

Robinson, J. (1971). Computational logic : the unification computation. *Machine Intelligence*, 6(1) :63–72.

Rosenschein, J. and Zlotkin, G. (1994). *Rules of Encounter : Designing Conventions for Automated Negotiation among Computers.* MIT Press.

Routen, T. and Bench-Capon, T. (1991). Hierarchical formalisations. *International Journal of Man-Machine Studies*, 35(1) :69–93.

Sacerdoti, E. (1977). *A structure for plans and behavior.* Elsevier Science Publishers.

Sandholm, T. (1999). Distributed rational decision making. In Weiss, G., editor, *Multiagent Systems : A Modern Approach to Distributed Artificial Intelligence.* MIT Press.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohme, F. (1999). Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1-2) :209–238.

Searle, J. (1969). *Speech act : An Essay in the Philosophy of Language.* Cambridge Unversity Press.

Searle, J. (1977). Austin on locutionary and illocutionary acts. *The Philosophical Review*, 1(1) :405–424.

Searle, J. (1979). *Expression and Meaning.* Cambridge Unversity Press.

Shehory, O. and Kraus, S. (1995). Task allocation via coalition formation among autonomous agents. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 655–661, Montréal, Québec, Canada.

Shoham, Y. and Tennenholtz, M. (1995). On social laws for artificial agents societies : off-line design. *Artificial Intelligence*, 73(1–2) :231–252.

Sichman, J., Conte, R., and Demazeau, Y. (1993). A social reasoning mechanism based on dependence network. In *Proceedings of the Italian Workshop on Distributed Artificial Intelligence*, Rome, Italie.

Singh, M. (1998). Agent communication languages : Rethinking the principles. *IEEE Computer*, 31(12) :40–47.

Singh, M. (1999). An ontology for commitments in multi-agent systems : toward a unification of normative concepts. *Artificial Intelligence and Law*, 7(1-2) :97–113.

Singh, M. (2000). A social semantics for agent communication language. In Dignum, F. and Geaves, M., editors, *Issue in Agent Communication*, pages 31–45. Springer Verlag Publishers.

Smith, I. and Cohen, P. (1996). Toward a semantics for an agent communications language based on speech acts. In Shrobe, H. and Senator, T., editors, *Proceedings of the National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, Vol. 2*, pages 24–31, Menlo Park, California, USA. AAAI Press.

Smith, R. and Davis, R. (1981). Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man and Cybernetics*, 11(1) :61–69.

Stanojevic, M., Vranes, S., and Velasevic, D. (1994). Using truth maintenance systems : A tutorial. *IEEE Expert : Intelligent Systems and Their Applications*, 9(6) :46–56.

Sycara, K. (1989a). Argumentation : Planning other agents' plans. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 517–523, Detroit, MI, USA.

Sycara, K. (1989b). Multi-agent compromise via negotiation. In *Distributed Artificial Intelligence*, volume 2, pages 119–138. Morgan Kaufmann Publishers.

Sycara, K. (1990). Persuasive argumentation in negotiation. *Theory and Decision*, 28(3) :203–242.

Takahashi, K., Nishibe, Y., Morihara, I., and Hattori, F. (1996). Collecting shop and service information with software agents. In *Proceedings of the International Conference on the Practical Application of Intelligent Agents and Multi-Agents Technology*, volume 11, pages 489–499. Taylor.

Tambe, M. (1996). Teamwork in real-world, dynamic environments. In *Proceedings of the International Conference on Multi-Agent Systems*, Menlo Park, California, USA. AAAI Press.

Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7(1) :83–124.

Tambe, M. (1998). Implementing agent teams in dynamic multi-agent environments. *Applied Artificial Intelligence*, 12(2-3) :189–210.

Tambe, M. and Zhang, W. (1998). Towards flexible teamwork in persistent teams. In *Proceedings of the International Conference on Multiagent Systems*.

Tang, Y. and Parson, S. (2005). Argumentation-based dialogues for deliberation. In *Proceedings of the International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 552–559.

Tate, A., Dalton, J., and Levine, J. (1999). Multi-perspective planning - using domain constraints to support the coordinated development of plans,. Technical Report AFRL-IF-RS-TR-1999-60, University of Edinburgh.

Tavares-Da-Silva, J.-L. (2003). *Programmation orientée multi-Agents : coordination dans les systèmes multi-agents voyelles.* PhD thesis, Université Joseph Fourier.

Tessier, C., Chaudron, L., and Müller, H., editors (2000). *Conflicting Agents : Conflict Management in Multi-Agent Systems.* Kluwer Academic Publishers.

Tonino, H., Bos, A., de Weerdt, M., and Witteveen, C. (2002). Plan coordination by revision in collective agent-based systems. *Artificial Intelligence*, 142(2) :121–145.

Toulmin, S. (1958). *The uses of arguments.* Cambridge University Press.

Toulmin, S., Rieke, R., and Janik, A. (1979). *Introduction to reasoning.* Macmillan Publisher Inc., New-York, USA.

Traum, D. (2004). Issues in multi-party dialogues. In Dignum, F., editor, *Advances in Agent Communication*, pages 201–211. Springer Verlag Publishers.

van der Krogt, R., de Weerdt, M., and Witteveen, C. (2003). A resource based framework for planning and replanning. In *Proceedings of the International Conference on Intelligent Agent Technology.*

Vauvert, G. and El-Fallah-Seghrouchni, A. (2000). Coalition formation for egoistic autonomous and weak rational agents. In *Proceedings of Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce*, Wollongong, Australie.

Vernant, D. (1997). *Du discours à l'études pragmatiques.* Press Universitaire de France, Paris.

Walton, D. and Krabbe, E. (1995). *Commitment in dialogue. Basic Concepts of Interpersonal Reasoning.* State University of New-York Press, Albany, New-York.

Wilkins, D. (1988). *Practical Planning : extending the classical AI planning paradigm.* Morgan Kaufmann Publishers, San Francisco CA, USA.

Winograd, T. and Flores, F. (1987). *Understanding computers and cognition : a new foundation for design.* Addison Wesley Professional, Boston, MA, USA, paperback edition.

Wittig, T. (1992). ARCHON *: an architecture for multi-agent systems.* Ellis Horwood, London.

Wooldridge, M. and Jennings, N. (1994a). Formalizing the cooperative problem solving process. In *Proceedings of the International Workshop on Distributed Artificial Intelligence*, pages 403–417, Lake Quinhalt, WA, USA.

Wooldridge, M. and Jennings, N. (1994b). Towards a theory of cooperative problem solving. In *Proceedings of the Conference on Modelling Autonomous Agents in a Multi-Agent World*, pages 15–26, Odense, Danemark.

Wooldridge, M. and Jennings, N. (1999). The cooperative problem-solving process. *Journal of Logic and Computation*, 9(4) :563–592.

Wooldridge, M., O'Hare, G., and Elks, R. (1991). Feline – a case study in the design and implementation of a co-operating expert system. In *Proceedings of the International Conference on Expert Systems and their Applications*, Avignon, France.

Wurman, P., Wellman, M., and Walsh, W. (1998). The michigan internet auctionbot : A configurable auction server for human and software agents. In Sycara, K. and Wooldridge, M., editors, *Proceedings of the International Conference on Autonomous Agents*, pages 301–308, New York. ACM Press.

Résumé. L'objectif de cette thèse est l'étude de la synthèse de plans dans un contexte multi-agent. La problèmatique principale est de comprendre les mécanismes qui permettent à une société d'agents autonomes de construire collectivement un plan global. Contrairement aux approches classiques dans lesquelles la planification n'est qu'un outil parmi d'autres pour définir les activités des agents et la coordination une « surcouche » nécessaire à leur synchronisation, la contribution de nos travaux repose sur la conception d'un modèle de planification entièrement distribué dans lequel les agents raisonnent conjointement sur leurs activités respectives pour atteindre un but commun prédéfini en intégrant leurs compétences hétérogènes ainsi que leurs croyances partielles sur le monde. Dans cette perspective, nous proposons de considérer la synthèse de plans comme un raisonnement collectif et révisable fondé sur l'échange entre les agents de conjectures, *i.e.,* des plans qui peuvent être exécutés si certaines conditions sont vérifiées et de réfutations, *i.e.,* des objections quant à la réalisation du plan. Les intéractions entre agents peuvent se définir comme un processus dialectique d'investigation dans lequel les agents proposent leurs compétences pour démonter la validité de certaines hypothèses, réfuter les conjectures non réalisables, réparer lorsque cela est possible les conjectures précédemment réfutées et ainsi élaborer pas à pas un plan solution.

Abstract. This PhD. thesis is devoted to the study of multi-agent planning. The issues we shall be concerned with is to understand the mechanisms which allow a group of autonomous agents to build collaboratively a global shared plan. Contrary to classical approaches in which planning is only a tool among many to specify agents' activities and coordination a subsuming component needed in order to synchronized them, our contribution relies on the conception of a completely distributed planning model : agents reason together on their own activities to achieve a shared goal, taking into account their heterogeneous skills and their partial knowledge of the world. We argue that multiagent planning process can be considered as collective and defeasible reasoning where agents exchange conjectures, *i.e.,* formulate plan steps based on hypothetical states of the world and refutations, *i.e.,* objections relative to plan achievement. The interaction between agents is a joint dialectical investigation process in which agents propose their skill to prove assumptions, refute unachievable conjecture, repair when it is possible previously refuted conjectures and finally build step by step a solution plan.