N^o d'ordre :

THÈSE

présentée devant

l'Université Paul Sabatier de Toulouse III

pour obtenir le titre de

DOCTEUR DE L'UNIVERSITÉ PAUL SABATIER

Mention INFORMATIQUE

par

DAVY CAPERA

École doctorale :	Informatique et Télécommunication
Laboratoire d'acceuil :	Institut de Recherche en Informatique de Toulouse (CNRS - INP - UPS)
Équipe d'accueil :	Equipe Systèmes Multi-Agents Coopératifs

Systèmes multi-agents adaptatifs pour la résolution de problèmes

Application à la conception de mécanismes

---

soutenue le 23 juin 2005 devant la commission d'examen :

Rapporteurs :	François Charpillet
	Joël Quinqueton
Examinateurs :	Claudette Cayrol
	Marie-Pierre Gleizes
	Pierre Glize
	Alain Remouchamps

Résumé

L'objet de cette thèse est le développement d'un processus automatique de conception de systèmes mécaniques basé sur la théorie des systèmes multi-agents adaptatifs. Le processus de conception débute avec l'ajout, par le concepteur, des éléments du problème : but à atteindre, enveloppe, contraintes, composants mécaniques connus... Par la suite, ce sont les composants mécaniques eux-mêmes (préalablement agentifiés dans un environnement de simulation) qui vont s'auto-organiser afin de trouver une solution en modifiant leurs paramètres, en créant de nouveaux composants mécaniques selon les besoins ou en modifiant la topologie du système.

Le développement du logiciel (*Mechanical Synthesis Solver*) mettant en oeuvre ce principe a nécessité l'expérimentation de nombreux d'algorithmes de traitement et nous a amené nous intéresser à l'élaboration d'un cadre formel pour la théorie des systèmes multi-agents adaptatifs. Ainsi, un modèle basé sur les automates a vu le jour et permet de modéliser de nombreux aspects et caractéristiques des systèmes multi-agents en général. Nous présenterons un exemple d'utilisation de ce modèle en l'appliquant au problème de conception mécanique.

Table des matières

Résumé **i**

Introduction **1**

I Contexte et état de l'art **5**

1 Les systèmes multi-agents adaptatifs **7**

1.1 Les systèmes multi-agents 7

1.1.1 Historique 7

1.1.1.1 L'Intelligence Artificielle (IA) 8

1.1.1.2 L'intelligence Artificielle Distribuée (IAD) et les Systèmes Multi-Agents (SMA) 9

1.1.2 Principes et concepts des SMA 10

1.1.2.1 L'agent 10

1.1.2.2 Les interactions 12

1.1.2.3 Adaptation et apprentissage 12

1.1.2.4 Récursivité 14

1.1.2.5 Emergence 14

1.2 La théorie AMAS 16

1.2.1 Adéquation fonctionnelle et couplage par interactions 17

1.2.2 Auto-organisation par coopération 18

1.2.2.1 Situations non coopératives (SNC) 19

1.2.2.2 Attitude coopérative 21
1.2.3 Intérêts, contraintes et problèmes 21
1.2.3.1 Intérêts d'une "néo-computation" 22
1.2.3.2 Contraintes de développement 24
1.2.3.3 Problèmes 25

2 Conception de systèmes mécaniques 27

2.1 Type Synthesis 28
2.1.1 Méthodologie systématique 29
2.1.1.1 Correspondance entre graphes et mécanismes 29
2.1.1.2 Sélection 33
2.1.1.3 Analyse 36
2.1.2 A-Design 36
2.1.2.1 Un processus itératif 37
2.1.2.2 Sélection 37
2.1.2.3 Représentation 39
2.1.2.4 Les agents 43
2.1.2.5 Analyse 45
2.2 Optimisation 46
2.2.1 Considérations générales 46
2.2.2 Méthodes semi-analytiques 48
2.2.3 Métaheuristiques 49
2.2.3.1 Avantages, inconvénients et développements 49
2.2.3.2 Recherche tabou 51
2.2.3.3 Recuit simulé 53
2.2.3.4 Algorithmes évolutionnaires 56
2.2.3.5 Essaims particulaires 58
2.3 Discussion 58

II Mechanical Synthesis Solver : conception automatique de mécanismes **61**

3 Architecture **63**

3.1 Le système 63

3.1.1 Besoins préliminaires et finals 64

3.1.1.1 Les besoins des utilisateurs 65

3.1.1.2 Caractérisation de l'environnement 65

3.1.2 Spécification du domaine 67

3.1.3 Identification des agents 68

3.1.4 Le processus global 69

3.2 Les agents 70

3.2.1 Modules des agents coopératifs 71

3.2.2 Implémentation des agents 74

3.2.2.1 Agent *point trajectoire* 76

3.2.2.2 Agent *barre* 77

3.2.2.3 Agent *point d'attachement* 78

3.2.2.4 Agent *charnière* 79

3.3 Conclusion 80

4 Optimisation - Régulation **81**

4.1 Compétences des agents 82

4.2 Algorithmes d'ajustement 83

4.2.1 Algorithmes réactifs 83

4.2.1.1 Principe 84

4.2.1.2 Implémentation 84

4.2.1.3 Résultats et analyse 85

4.2.2 Algorithmes basés sur l'apprentissage 89

4.2.2.1 Principe 89

4.2.2.2 Implémentation 90

4.2.2.3 Résultats et analyse 90
4.2.3 Algorithme basé sur une mémoire coopérative 91
4.2.3.1 Principe 92
4.2.3.2 Implémentation 96
4.2.3.3 Résultats et analyse 96
4.3 Analyse 98

5 Type Synthesis - Évolution 101
5.1 Situations Non Coopératives 101
5.2 Résultat 104
5.3 Conclusion 107

III Modélisation des systèmes multi-agents basée sur les automates 111

6 Modèle basé sur les automates 113
6.1 Composants élémentaires 114
6.1.1 Agents et automates 114
6.1.2 Automates “types” 114
6.1.3 Configurations élémentaires 116
6.2 Système, configurations et transitions globales 117
6.2.1 Système et configurations globales 117
6.2.1.1 Transitions globales 117
6.3 Générateurs ou axiomes de la relation de transition 118
6.3.1 Règles 118
6.3.2 Axiomes ou règles “localisées” pour les transitions. 119
6.3.3 Application des règles 120
6.3.3.1 Types de conditions 120
6.3.3.2 Règles globales 120
6.3.4 Système engendré par un ensemble de règles 121
6.4 Un modèle pour les SMA 121

6.4.1 Les composants essentiels 121
6.4.2 Les interactions 122
6.4.3 Application concurrente ou distribuée des règles 123
6.4.4 Notion d'implémentation, de simulation 123
6.5 Concepts et notions exprimables dans le modèle 124

7 Exemple d'application du modèle au *MSS* 127

7.1 Représentation de l'environnement 127
7.2 Représentation des agents 127
7.2.1 Automate des agents mécaniques 127
7.2.2 Description des connaissances des agents 129
7.3 Règles de transition 129
7.3.1 $q_{Wait} \rightarrow q_{DetectNcs}$ 130
7.3.2 $q_{DetectNcs} \rightarrow q_{Wait}$ 133
7.3.3 $q_{DetectNcs} \rightarrow q_{Regulation}$ 134
7.3.4 $q_{DetectNcs} \rightarrow q_{Re-organisation}$ 135
7.3.5 $q_{Regulation} \rightarrow q_{Wait}$ 135
7.3.6 $q_{Re-organisation} \rightarrow q_{Wait}$ 137
7.4 Conclusion 138

Conclusion 141

7.5 Le *Mechanical Synthesis Solver* 141
7.6 Modèle des systèmes multi-agents 142
7.7 Sur la théorie et la méthodologie 143

Bibliographie 145

Liste des figures 151

Liste des tables 153

Introduction

AVEC l'essor des technologies de l'informatique et de l'Internet, les applications se complexifient et les systèmes multi-agents deviennent de plus en plus incontournables pour leur conception. Le domaine de recherche des systèmes multi-agents, très actif depuis les années 80, se divise en trois grandes familles :

1. la simulation de systèmes complexes, notamment les systèmes naturels et sociaux ;
2. les systèmes dans lesquels les utilisateurs humains sont représentés par des artefacts logiciels ;
3. la résolution de problèmes difficiles.

Mes travaux s'intègrent dans cette dernière catégorie qui s'intéresse à la mise en oeuvre de processus de résolution se démarquant des approches traditionnelles en informatique. Plus particulièrement, l'objet de cette thèse est l'étude et la formalisation de la théorie des systèmes multi-agents adaptatifs comme paradigme alternatif pour la mise en oeuvre d'applications dédiées à la résolution de problèmes complexes. Pour ce faire, nous avons choisi une approche tant pratique, au travers du développement d'un outil de conception automatique de mécanismes, que théorique, en s'attachant à la mise en place d'un modèle basé sur les automates pour les systèmes multi-agents en général.

Le contexte de mon étude est celui de la théorie des systèmes multi-agents adaptatifs, développée au sein de l'équipe Systèmes Multi-Agents Coopératifs[1] depuis un peu plus de dix ans. Auto-organisation coopérative et émergence sont les pierres angulaires de cette approche et permettent de concevoir des systèmes qui peuvent, en s'appuyant sur l'auto-organisation guidée par la coopération, adapter leur comportement à l'environnement et produire ainsi les fonctionnalités attendues sans que celles-ci soient codées explicitement. Notre objectif est d'utiliser ce processus pour résoudre des problèmes considérés comme complexes et ce de manière efficace.

L'élaboration de la théorie s'est, depuis ses prémisses, largement appuyée sur l'expérimentation afin de s'étayer par le biais des enseignements que l'on peut retirer du développement effectif d'applications ainsi que de la confronter à la réalité pour tester et valider la cohérence des idées qui participent à son évolution. C'est dans cette optique que le point focal de ce travail de thèse est une application très concrète développée, dans le cadre d'un projet européen, avec des partenaires tant académiques

[1] http ://www.irit.fr/SMAC

qu'industriels : la conception de mécanismes. En effet, cette activité s'avère être complexe et coûteuse car elle ne dispose, à l'heure actuelle, d'aucun outil dédié permettant d'aider les ingénieurs lors des phases préléminaires durant lesquelles ils doivent développer le "concept" d'un mécanisme pour répondre à un problème donné. Ce travail a donné lieu à l'implémentation du *Mechanical Synthesis Solver*, outil permettant la synthèse automatique de mécanismes et dont les résultats seront exposés ici.

Les enseignements retirés de cette réalisation pratique, portent essentiellement sur les difficultés que l'on peut rencontrer lors du développement de telles applications et, plus particulièrement, lors de l'élaboration des situations non coopératives et des méthodes de traitement associées qui guident le processus de ré-organisation. Ce problème de conception, bien qu'ayant été récemment l'objet d'un travail dont l'aboutissement a été la création d'une méthodologie spécifique nommée *ADELFE*, demeure le principal défi auquel la théorie est confrontée. Si cette difficulté revêt un aspect pratique en terme de génie logiciel, sa source est plus d'ordre théorique : la théorie manque cruellement d'outils formels.

Partant de ce constat, nous avons établi un modèle basé sur les automates qui permet de modéliser nombre de systèmes multi-agents ainsi que d'exprimer leurs propriétés. Le premier objectif de ce travail est de réaliser un modèle adapté à l'expression de n'importe quel système multi-agent. De fait, ce modèle permet plus particulièrement de modéliser les systèmes multi-agents adaptatifs. A titre d'exemple, l'outil de conception de mécanisme développé a été traité dans le cadre de ce modèle. Le travail effectué jusqu'à présent sur cette formalisation des systèmes multi-agents ne concerne que l'établissement d'un modèle. Néanmoins, nous avons, à plus long terme, l'ambition de pouvoir grâce à lui exprimer et prouver des propriétés intéressantes notamment sur la convergence des systèmes construits selon la théorie des *AMAS*. La deuxième perspective majeure de ce travail est le développement d'outils qui pourront être intégrés dans la méthodologie *ADELFE* afin d'aider à la conception des agents au niveau des situations non coopératives.

Ce mémoire se divise donc en trois parties :

1. La présentation du contexte et de l'état de l'art. Nous y aborderons le domaine des systèmes multi-agents au moyen d'un historique des considérations ayant présidées à l'élaboration de la théorie des *AMAS*, ainsi qu'un synthétique aperçu du domaine de la mécanique par le biais de la présentation de deux applications dédiées à la conception de mécanismes et des techniques d'optimisations classiquement utilisées pour la résolution des problèmes d'ajustement des dimensions des mécanismes générés.

2. Le développement du *Mechanical Synthesis Solver* sera l'objet de la deuxième partie. Nous présenterons les différentes étapes de la conception de cette application et les résultats que nous

avons pu obtenir lors des tests que nous avons menés. L'accent a été mis sur la partie du développement portant sur les situations non coopératives : plusieurs algorithmes basés sur des principes différents ont été développés et sont présentés avec les résultats correspondant.

3. La troisième partie présente le modèle basé sur les automates et son application pour la modélisation du *Mechanical Synthesis Solver*. Les perspectives pour l'utilisation de ce modèle concluront cette partie.

Première partie

Contexte et état de l’art

1 Les systèmes multi-agents adaptatifs

LA théorie des AMAS (Adaptive Multi-Agent Systems), qui entre dans le cadre de l'intelligence artificielle distribuée, est le thème de recherche principal de l'équipe SMAC[1] au sein de laquelle ce travail a été réalisé. Elle offre une approche alternative pour la conception de systèmes complexes voués à être immergés dans des milieux fortement dynamiques ou bien impossibles à spécifier exhaustivement. Partant du constat que les programmeurs se heurtent de plus en plus fréquemment à une trop grande complexité conduisant à des logiciels peu fiables voire impossibles, a priori, à réaliser, ce nouveau paradigme propose d'utiliser des processus élémentaires faciles à programmer (les agents) qui s'auto-organisent grâce à la coopération pour produire une fonction globale complexe émergente. Cet axe de recherche encore jeune est actuellement le sujet de plusieurs études théoriques portant plus particulièrement sur les thèmes de l'émergence et des propriétés de convergence.

Dans ce chapitre, je présente tout d'abord les systèmes multi-agents de manière générale - au travers de leur historique et des concepts qui s'y rattachent - avant de décrire dans une deuxième partie la théorie AMAS, elle-même, qui fournit un cadre pour la conception de systèmes adaptatifs utilisant l'auto-organisation par la coopération pour obtenir des systèmes fonctionnellement adéquats vis-à-vis de leur environnement.

1.1 Les systèmes multi-agents

Les systèmes multi-agents (SMA) se présentent comme l'objet d'étude incontournable de ces dernières années et, probablement, celui des années à venir. Traditionnellement rattachés au domaine de l'Intelligence Artificielle, largement inspirés des systèmes naturels, les SMAs offrent aujourd'hui une approche novatrice pour la construction de systèmes artificiels.

1.1.1 Historique

Il convient de commencer par les raisons qui ont motivé la naissance des systèmes multi-agents. Je présente donc, dans un premier temps, l'Intelligence Artificielle classique et analyse les insuffisances

[1] Systèmes Multi-Agents Coopératifs

et problèmes qui ont amené au développement de l'Intelligence Artificielle Distribuée.

1.1.1.1 L'Intelligence Artificielle (IA)

La naissance de l'IA en tant que champ de recherche est souvent associée à la conférence d'été de Dartmouth en août 1956. C'est dans la proposition pour cette conférence (McCarthy *et al.*, 1955) que l'on peut lire qu'une machine intelligente "devrait tendre à l'élaboration interne d'un modèle abstrait représentant l'environnement dans lequel elle est placée. A la présentation d'un problème, il convient d'abord d'étudier l'ensemble des solutions dans ce modèle interne de l'environnement et ensuite de tenter de les appliquer à l'extérieur "[2]. C'est cette approche, axée sur la représentation symbolique des connaissances et un raisonnement formel sur ces symboles, qui a dominé la majeure partie de la recherche pendant les trente années qui suivirent.

Les plus belles réalisations de l'IA classique se situent sans doute en robotique, domaine le plus proche du "Graal de l'être artificiel intelligent", et dans les systèmes d'aide à la décision ou au diagnostic avec la mise au point, à la fin des années 70, des systèmes experts.

Cependant, on assiste depuis une quinzaine d'années à l'apparition de nouveaux besoins - systèmes répartis sur plusieurs lieux et interconnectés via des réseaux locaux ou l'Internet, immersion des systèmes dans des milieux fortement dynamiques - qui mènent à une complexité croissante ne permettant plus, parfois, de pouvoir appréhender le système dans son ensemble voire d'être capable de le concevoir sans simplification importante de ses fonctionnalités. C'est dans ce sens que Rodney A. Brooks dans (Brooks, 1990) critique ces systèmes délibératifs en ajoutant la difficulté de maintenir des réactions en temps réel lors de brutaux changements dans l'environnement. Si Brooks souligne les faiblesses de l'IA classique en 1990, c'est qu'elles sont alors réellement apparentes et d'actualité. En effet, face à l'explosion de la complexité de l'Internet, aux progrès technologiques qui permettent la généralisation de l'utilisation de réseaux à toutes les échelles et à la demande de plus en plus forte en systèmes intelligents dans tous les domaines (industrie, jeux vidéo, bureautique...) l'IA traditionnelle se trouve être de moins en moins performante (tant au niveau des possibilités qu'au niveau de la fiabilité) et de plus en plus coûteuse (notamment en terme de conception et de maintenance) et ce malgré l'augmentation considérable de la puissance de calcul et la baisse conséquente de son coût.

C'est l'ensemble de ces critiques qui ont amené une partie de la communauté scientifique à s'intéresser aux systèmes distribués plutôt qu'aux artefacts monolithiques et parfois peu intuitifs qui ont fait les beaux jours de l'Intelligence Artificielle.

[2]The important result that would be looked for would be that the machine would tend to build up within itself an abstract model of the environment in which it is placed. If it were given a problem, it could first explore solutions within the internal abstract model of the environment and then attempt external experiments.

1.1.1.2 L'intelligence Artificielle Distribuée (IAD) et les Systèmes Multi-Agents (SMA)

Face au premier courant de pensée issu de la préoccupation d'atteindre un niveau d'intelligence semblable à l'humain, apparaît, dès le début des années 60, l'IAD. L'idée à la base de la conception du système Pandémonium (Selfridge et Neisser, 1960), précurseur du domaine, est que de multiples processus sont capables de générer un comportement cohérent en coopérant ou en entrant en compétition tout comme le font des formes de vie jugées peu évoluées mais capables d'exhiber tout de même des comportements collectifs complexes.

En fait, la première génération de systèmes distribués est caractérisée par une volonté de distribuer les connaissances (plusieurs bases de connaissances) et le traitement (plusieurs systèmes experts), même si l'on garde un contrôle général centralisé. L'archétype de ces systèmes est sans doute celui de reconnaissance et compréhension de la parole continue HEARSAY II (Erman *et al.*, 1980). Ce système est constitué de sources de connaissances asynchrones qui communiquent à l'aide d'une structure de donnée globale appelée "Tableau Noir". On peut aussi voir l'apparition d'une terminologie pour nommer ces entités qui constituent les systèmes : ce sont les notions d'*Acteurs* de Hewitt (Hewitt, 1977) ou d'*êtres* (*Beings*) de Lenat (Lenat, 1975).

Une seconde génération, caractérisée par la décentralisation du contrôle, fit son apparition au début des années 80 ainsi que la terminologie d'*Agent* marquant la naissance des systèmes multi-agents. Le modèle d'application Contract Net dans (Davis et Smith, 1983) dénote bien le souci de trouver un moyen flexible et dynamique de distribuer l'exécution d'une tâche globale divisée entre plusieurs agents totalement autonomes, en proposant un modèle de passage de contrat inspiré de l'économie de marché. On peut citer également la plate-forme générique MACE (Gasser *et al.*, 1987) qui, avec son organisation en "couches d'agents ", essaie d'apporter une réponse au souci de réutilisabilité des systèmes. A cette époque, les travaux de l'équipe SMAC dans le cadre du projet SYNERGIC (Cannizzo-Puyet *et al.*, 1991) participent au développement de cet axe de recherche.

Depuis le début des années 90, l'IAD a donné naissance à une troisième génération dont les concepts clés sont issus des systèmes existants dans la nature au travers de l'utilisation de connaissances des domaines de l'éthologie, la sociologie, la vie artificielle, etc. Les thématiques principales sont les interactions entre les agents, l'hétérogénéité dans les systèmes, l'autonomie (de contrôle, de connaissance et éventuellement de ressource), l'interopérabilité entre systèmes, l'auto-organisation des agents dans un système et l'émergence de fonctionnalités que ces capacités permettent d'observer dans certains systèmes.

La réalisation de modèles informatiques composés d'entités artificielles qui communiquent entre elles et interagissent avec l'environnement possède, à mon sens, une beauté subjective qui mérite que l'on s'y attache. De plus, les perspectives offertes par les systèmes multi-agents sont séduisantes et

excitent la curiosité car ces systèmes obéissent aux principes des phénomènes émergents et/ou chaotiques qui semblent être de première importance, ou en tout cas fortement présents, dans la nature et constituent aujourd'hui un axe de recherche majeur que l'on peut regrouper sous le terme de systèmes complexes.

1.1.2 Principes et concepts des SMA

Science intégrative plus qu'analytique jusqu'à maintenant, les systèmes multi-agents suscitent de nombreuses interrogations : quelles sont les caractéristiques d'un agent ? Quels types d'interactions ont-ils ? Quels mécanismes d'adaptation et d'apprentissage peuvent-ils utiliser ? Comment se structure un SMA ?

1.1.2.1 L'agent

L'agent étant la pierre angulaire du fonctionnement des SMA, on pourrait penser qu'il en existe une définition relativement précise et admise par tous. Dans les faits, chacun apporte sa propre définition en fonction du domaine auquel il appartient ou au travail qu'il effectue sur le moment, ce qui a amené certains chercheurs comme J. Ferber ou encore Y. Demazeau à tenter d'établir une définition minimale.

Selon (Ferber, 1995), un agent est une entité physique ou virtuelle autonome, ayant des capacités de perception et d'action relatives à l'environnement dans lequel il est immergé et dont il ne possède qu'une représentation partielle (principe de localité). On peut attribuer la structure suivante à un agent :

- Compétences : ce qu'il sait faire, son rôle au sein du système.
- Croyances : connaissances partielles et éventuellement partiales sur lui-même, sur les autres agents du système et sur son environnement.
- Accointances : attitudes sociales adoptées dans la relation avec les autres agents.
- Aptitudes : "boîte à outil" pour traiter les perceptions, actions, raisonnements, communications et apprentissage de l'agent.

Malgré ces éléments de définition, il est tout de même difficile de se faire une idée très précise sur ce qu'est effectivement un agent ou sur ce qui n'est pas un agent. Un peu de dialectique socratique...

Tout est donc agent ? Il convient de voir l'agent comme un concept, une manière d'aborder les choses. Avec un certain regard, une certaine interprétation, beaucoup de choses peuvent être nommées agent : une fourmi, un robot, une application ou, de manière moins évidente dans certaines applications, un atome, un paramètre, un mot, une boîte... tout dépend des capacités

rattachées à ces objets.

Tout processus serait donc un agent ? Tout processus n'est pas un agent : un simple processus n'ayant pour code qu'une boucle infinie vide n'a pas de croyances, pas d'accointances, pas d'interaction... Par contre, les processus sont sans aucun doute un outil qui permet de construire des agents logiciels en les dotant d'une autonomie de fonctionnement évidente.

Les objets (au sens de la programmation orientée objets) sont des agents ? Tout comme pour les processus, les objets ne sont sont pas forcément des agents mais sont des outils très utiles pour créer des agents logiciels. Attention tout de même : utiliser les objets pour concevoir des agents n'est en rien nécessaire, de même que concevoir une application en utilisant UML n'oblige en rien le développeur à coder des objets.

Un agent est donc forcément "intelligent" ? Effectivement les agents ont toujours une certaine "intelligence", mais celle-ci est élaborée et contrôlée par l'ingénieur pour que l'agent effectue des actions rationnelles. Ainsi on peut distinguer des agents sociaux qui ont la capacité à interagir avec les autres agents via des langages de communication pour satisfaire leurs buts, des agents réactifs qui maintiennent des interactions avec leur environnement et répondent aux stimuli, ainsi que des agents dits pro-actifs qui peuvent générer et atteindre des buts en n'étant pas dirigés seulement par les événements et peuvent prendre des initiatives pour satisfaire leurs buts (Wooldridge et Jennings, 1995).

En fin de compte, l'autonomie est sans doute la caractéristique clé permettant de singulariser l'agent. Un essai de classification systématique des travaux sur le sujet a été fait dans (Carabelea *et al.*, 2003) à partir de cinq points de vue : l'agent, l'environnement, l'interaction, l'organisation et l'utilisateur (Demazeau, 2001). Sur chacun de ces axes, on peut caractériser un type d'autonomie particulier :

1. Un agent équipé d'une A-autonomie adaptable peut varier d'une autonomie non consciente qui ne prend pas en compte ses motivations à une autonomie consciente (self-aware), qui raisonne sur celles-ci.

2. La E-autonomie signifie que le comportement d'un agent n'est pas contrôlé par l'environnement dans lequel il est situé, mais est influencé par les changements se produisant dans cet environnement.

3. Un agent avec une I-autonomie adaptable est un agent qui adapte son comportement au contexte courant : quelquefois il accepte l'adoption de but et quelquefois il refuse. Nous pouvons dire que l'I-autonomie d'un agent peut varier d'une autonomie autiste (il refuse tout) à une autonomie bienveillante (il accepte tout), en passant par l'étape intermédiaire de I-autonomie adaptable.

4. Un agent est O-autonome par rapport à une norme (des lois sociales, des conventions ou des structures organisationnelles) s'il peut violer cette norme.
5. L'U-autonomie d'un agent peut varier entre une totale indépendance vis-à-vis de l'utilisateur à une dépendance totale en passant par la position intermédiaire d'une U-autonomie adaptable.

1.1.2.2 Les interactions

Si l'agent est de première importance dans le fonctionnement des SMA, ce sont les mises en relation dynamique de deux ou plusieurs agents qui vont permettre au système de fonctionner et d'évoluer efficacement. En effet, les agents étant plongés ensemble dans un environnement au sein duquel ils doivent réaliser une tâche précise, ils vont devoir se partager le travail à effectuer (en fonction des compétences de chacun) et s'organiser en tenant compte des contraintes provenant des ressources plus ou moins limitées dont ils disposent (Ferber, 1995). Ferber a aussi analysé les types de situations dans lesquelles peut se trouver un groupe d'agents (et donc les types d'interaction qui s'avèrent nécessaires) en fonction de la compatibilité des buts de chaque agent avec ceux des autres, de l'abondance des ressources et de la qualité des compétences de chaque agent pour atteindre ses buts (si elles sont insuffisantes, il devra coopérer avec d'autres agents).

Il est à noter que les agents étant autonomes, leurs interactions nécessitent souvent un système de communication. Cette communication peut être indirecte (utilisation d'une structure de données commune, communication via l'environnement) ou directe (envoi de messages, diffusion totale, communication de proche en proche, etc.). Le consortium FIPA (Fundation for Intelligent Physical Agents)[3] a développé en ce sens un langage (ACL pour Agent Communication Language) pour la communication entre agents ou entre systèmes (interopérabilité) et a proposé une structure générique de message, des mots-clés pour les actes de communication (accepter une proposition, demander une information, etc.) ainsi que des librairies de termes regroupés par domaines (ontologie).

1.1.2.3 Adaptation et apprentissage

L'une des principales capacités attendue par les systèmes distribués est de pouvoir réaliser leur fonction dans des environnements fortement dynamiques ou impossibles à spécifier complètement quitte à adapter, si nécessaire, cette fonction aux besoins. C'est grâce à l'adaptation et l'apprentissage que les agents peuvent satisfaire ces demandes et être réellement évolutifs et adaptatifs face à leur environnement, améliorant ainsi le fonctionnement des SMA ainsi que leur conception et leur développement. Il faut remarquer que la plupart des recherches en IAD, pour l'apprentissage ou la

[3] `http://www.fipa.org/`

coordination des agents, se concentrent sur le développement de stratégies en différé, basées sur une fonction d'évaluation globale, qui peuvent devenir rapidement inadéquates si le modèle du monde utilisé par le concepteur n'est pas correct et parfaitement complet (ce qui est impossible en milieu dynamique ouvert où l'imprévu est la règle).

Il est possible de dégager quatre axes d'apprentissage :

1. Apprentissage centré agent : l'agent peut apprendre à améliorer la qualité de son comportement (par exemple en apprenant à reconnaître les conditions dans lesquelles il peut agir et celles où il doit s'abstenir) ou bien augmenter ses connaissances vis-à-vis des agents qu'il côtoie (notamment si leurs caractéristiques sont évolutives ou bien intéressantes pour l'agent).
2. Apprentissage centré environnement : l'agent peut obtenir des informations sur des parties de l'environnement qui lui sont inaccessibles ou des données qu'il ne peut lui-même obtenir directement.
3. Apprentissage centré interactions : les agents peuvent apprendre à communiquer ou, s'ils le peuvent déjà, apprendre et créer une nouvelle terminologie si nécessaire.
4. Apprentissage centré organisation : les agents s'auto-organisent pour gérer des ressources limitées ou pour réaliser une tâche nécessitant l'intervention de plusieurs d'entre eux.

De nombreux algorithmes existent pour effectuer des apprentissages dans ces quatre axes.

1. *Centré agent* : Maja Matarić propose dans sa thèse (Matarić, 1994) une fonction, locale à chaque agent, de renforcement hétérogène utilisant des buts multiples et deux fonctions d'estimation de progrès dans la tâche à accomplir. Sati Singh Sian [Sian, 1991] utilise dans MALE (Multi-Agent Learning Environment) une méthode basée sur la coopération comme mécanisme d'adaptation et d'apprentissage.
2. *Centré environnement* : Un feed-back de l'environnement informe le système sur la qualité de ce qu'il réalise. De nombreux algorithmes existent (classifieurs, Q-learning, rétro-propagation du gradient dans les réseaux de neurones) mais ils exigent la mise au point d'un contrôle centralisé pour générer le feed-back nécessaire à l'apprentissage (pour juger la qualité du comportement).
3. *Centré interactions* : Ces travaux peuvent être grossièrement découpés en deux axes principaux :
   - émergence d'un langage commun à une population d'agents d'une même génération (Steels, 1996) (Batali, 2002) ;
   - émergence et évolution de langages au travers de plusieurs générations d'une population (Kirby, 2002) (Kasakov et Bartlett, 2003).
4. *Centré organisation* : La plupart des méthodes d'apprentissage reprennent les algorithmes fonctionnant par renforcement. Les travaux en cours au sein de l'équipe SMAC sur la théorie des AMAS (cf. partie 1.2) portent sur ce type d'apprentissage.

1.1.2.4 Récursivité

L'approche récursive pour la résolution de problèmes ou la conception de systèmes complexes constitue un des aspects les plus fascinants des SMA (Fischer, 1999) (Demazeau, 1995). Cette méthode, qui consiste à considérer un SMA comme un agent de plus haut niveau ou à concevoir un agent comme un SMA et donc avec des agents de niveau plus faible, possède de nombreux avantages : réduction de la complexité de conception, hiérarchisation selon différents niveaux d'abstraction et donc facilité d'observation du fonctionnement.

Y. Demazeau propose, par exemple, un modèle récursif nommé *MAGMA* dans (Demazeau, 1995) pour la conception de systèmes par décomposition de tâches basé sur la définition à chaque niveau d'abstraction des quatres éléments :

1. **A**gent : déterminer si l'agent est atomique ou bien décomposable en un groupe d'agent de plus faible granularité.
2. **E**nvironnement : l'environnement d'un agent de niveau n - 1 est la représentation du monde perçu par l'agent à un niveau n.
3. **I**nteraction : seuls deux agents d'un même niveau d'abstraction et d'un même groupe peuvent interagir. Cette interaction a pour but la coopération, la coordination et la communication pour l'échange d'information.
4. **O**rganisation : le SMA, à un premier niveau, est conçu comme un seul agent, qui représente le niveau d'abstraction le plus haut. A partir du deuxième niveau, les agents sont organisés en groupes et, par décompositions successives, on peut avoir des groupes formés par des groupes d'agents. La décomposition se termine lorsqu'un groupe n'est formé que d'agents élémentaires.

1.1.2.5 Emergence

L'émergence est une caractéristique intrinsèque des plus intéressantes dans les systèmes multi-agents. Elle permet de rendre possible, à partir d'un micro-niveau constitué d'agents élémentaires en interaction, l'apparition de fonctions, structures ou comportements totalement nouveaux et souvent imprévisibles à un niveau d'abstraction plus élevé (le macro-niveau) (Georgé, 2003).

Historique. On retrouve des traces du concept d'émergence dans la Grèce antique au travers des écrits de Thalés et Anaximandre dans lesquels apparaît le postulat "le tout est plus que la somme de ses parties ". Ce n'est toutefois qu'au 19ième siècle qu'une étude plus approfondie sur la différence entre émergent et résultant sera effectuée par G. H. Lewes.

C'est vers 1920 que naît le mouvement proto-émergentiste inspiré des idées de Lewes qui, n'expliquant pas le phénomène, le compare à une boîte noire. S'ils ne peuvent en décrire le fonctionnement, ils le caractérisent pourtant en notant que le niveau d'abstraction du phénomène est supérieur à celui des éléments qui en sont à l'origine. Ce courant de pensée permet de faire la distinction entre résultant et émergent en postulant que pour ce dernier il est impossible de décrire, d'expliquer ou de prédire le processus à partir des conditions de base définies aux niveaux inférieurs.

Les premières tentatives d'élaboration de méthodes visant à démystifier l'émergence ont vu le jour récemment avec le courant du néo-émergentisme dont fait partie John H. Holland, "père " des algorithmes génétiques et auteur de "Emergence - From Order to Chaos " (Holland, 1997) qui permet de bien saisir les objectifs de ce mouvement. Il y décrit, par exemple, un nouveau formalisme qu'il nomme " Constraint Generating Procedures " (CGP) qui permet de modéliser une large gamme de problèmes, qu'ils soient émergents ou non, grâce à une généralisation des automates cellulaires. Cette approche permet d'essayer de comprendre les mécanismes internes qui œuvrent à l'apparition de ces phénomènes et met en avant l'importance de l'interaction entre les parties.

D'après Jeffrey Goldstein (Goldstein, 1999), on peut dégager quatre traits communs à tous les systèmes émergents, basés sur la Théorie de la Complexité :

- la non-linéarité : correspond aux interactions système-environnement qui sont non-linéaires par nature ;
- l'auto-organisation : correspond au comportement créatif, auto-généré et à la recherche d'adaptivité d'un système complexe de la Théorie de la Complexité ;
- l'au-delà de l'équilibre : au lieu de se focaliser sur les points d'équilibre du système, on s'intéresse au voisinage de ces points. En de tels points, l'apparition de phénomènes aléatoires explique le caractère inattendu de l'émergence ;
- les attracteurs : contrairement aux systèmes primitifs où il n'y avait qu'un seul type d'attracteur menant à l'état final, les systèmes émergents possèdent différents types : le point fixe, le cycle limite et l'attracteur étrange.

On peut aussi ajouter les caractéristiques suivantes :

- la dynamique : c'est-à-dire la capacité pour le système à se métamorphoser, à s'attribuer de nouveaux attracteurs qui ne sont pas prédéfinis ;
- l'irréductibilité : l'émergence est définie en termes *d'irréductibilité des propriétés associées à des composants dans une théorie de plus bas niveau* (Ali et Zimmer, 1997). Ceci rejoint l'idée de nouveauté et d'imprédictibilité qui caractérise les systèmes émergents.
- la nouveauté : un phénomène émergent est non prédictible, a priori, à partir de la connaissance du micro-niveau.

Réflexions et intérêts Le concept d'émergence est aujourd'hui le centre d'intérêt de nombreuses équipes de recherche dans différents domaines (Thermodynamique, Sociologie, Mathématiques du chaos, Synergétique, Neurosciences) avec pour espoir de permettre un développement nouveau dans la compréhension que nous avons du monde.

Dans le domaine de l'informatique (nous parlons alors de "computation émergente" (Forrest, 1991)), ce qui nous intéresse plus précisément est la possibilité de produire un comportement global intéressant à partir d'éléments simples (notamment en terme de conception) qui entreraient en interactions elles-mêmes relativement primaires et donc faciles à gérer (Georgé, 2004). La possibilité la plus remarquable est de pouvoir concevoir un système non pas par décomposition des tâches (ce qui n'est pas toujours possible et induit souvent une gestion lourde voire centralisée des interactions) de type descendante ("Top-Down") mais par la création d'entités primaires aux interactions locales simples qui vont s'auto-organiser et dont émergera la fonction ou le comportement global désiré (conception de type ascendante ou "Bottom-Up"). De ces considérations, nous avons tiré une définition dite "utilitaire" qui est donc axée sur ce que l'on veut faire émerger, à quelle condition l'on pourra dire qu'il y a émergence et comment l'on peut l'exploiter dans le cadre des systèmes artificiels (Capera *et al.*, 2003) (Georgé *et al.*, 2003b) :

Objet. Un système informatique a pour finalité de réaliser une fonction adéquate à ce que l'on attend du système. C'est cette fonction, pouvant évoluer au cours du temps, que nous voulons faire émerger.

Condition. Cette fonction est émergente si le codage du système ne dépend aucunement de la connaissance de cette fonction. Ce codage doit contenir des mécanismes permettant l'adaptation du système au cours de ses échanges avec l'environnement afin de tendre à tout instant vers la fonction adéquate.

Méthode. Dans la théorie des AMAS[4], pour changer la fonction il suffit de changer l'organisation des composants du système. Ces mécanismes sont spécifiés par des règles régissant l'auto-organisation entre les composants et ne dépendant pas de la connaissance de la fonction collective.

1.2 La théorie AMAS

La théorie AMAS (Adaptive Multi-Agent Systems) a été développée au sein de l'équipe SMAC de l'IRIT (Glize, 2001) et propose d'utiliser le principe de l'auto-organisation par coopération comme

[4] Adaptive Multi-Agent Systems. Cf. 1.2

processus d'adaptation des systèmes artificiels immergés dans des environnements fortement dynamiques ou dont la tâche à effectuer est difficile, voire impossible, à spécifier exhaustivement.

1.2.1 Adéquation fonctionnelle et couplage par interactions

Un SMA, contenant de nombreux agents capables d'interactions mutuelles et avec l'environnement, doit réaliser une fonction correcte, du point de vue du concepteur, dans l'environnement dans lequel il est plongé. Si cette adéquation fonctionnelle ne peut être jugée que par un observateur extérieur connaissant la fonction qui doit être réalisée, ce sont les interactions mutuelles entre le système et son environnement qui vont être le moteur de l'adaptation du système.

Ce principe, appelé couplage par interaction, induit des influences réciproques générant un processus d'ajustement mutuel : un changement dans l'environnement peut entraîner une modification interne du système, mais la réponse du système peut elle-même engendrer une nouvelle réaction de l'environnement.

Les relations entre un système et son environnement peuvent être classées en trois catégories :

1. L'activité coopérative : les activités de chacun sont complémentaires, ils sont dans une sorte d'état de symbiose.
2. L'activité antinomique : l'activité de l'un se fait au détriment de la poursuite de celle de l'autre. Ils sont en conflit.
3. L'activité indifférente : cette activité est, elle aussi, non-coopérative mais un état du monde ne gêne l'action d'aucun des deux.

Pour atteindre un état d'adéquation fonctionnelle le système doit donc n'avoir, avec l'environnement, aucune activité antinomique. Ceci définit l'axiome de base de la théorie AMAS : *Un système fonctionnellement adéquat n'a aucune activité antinomique sur son environnement*.

La question qui se pose alors est : " Quelles sont les relations entre l'ensemble des systèmes coopératifs et l'ensemble des systèmes fonctionnellement adéquats ?". Pour répondre à cette question, les travaux de l'équipe ont abouti à la preuve des deux lemmes suivants (Camps *et al.*, 1998) :

Lemme 1 : *L'ensemble des systèmes coopératifs est inclus dans l'ensemble des systèmes fonctionnellement adéquats*.

Lemme 2 : *Pour tout système fonctionnellement adéquat, il existe un système coopératif qui soit fonctionnellement adéquat dans le même environnement*.

Si nous pouvons être sûrs que pour un système ayant une fonction quelconque donnée il est possible d'utiliser un système coopératif - puisqu'il existe une surjection des systèmes fonctionnellement adéquats vers ces derniers -, le problème est de pouvoir créer ce système. La théorie propose pour

cela d'utiliser un SMA dont le milieu intérieur (les agents qui le composent) est lui-même coopératif. Ce milieu interne est dit coopératif si chacune de ses parties est en interaction coopérative avec son environnement (c'est-à-dire soit d'autres parties du SMA, donc d'autres agents ou groupes d'agents, soit des parties de l'environnement du SMA).

Ainsi, il a été montré qu'il existe une opération de surjection de l'ensemble des systèmes coopératifs vers l'ensemble des systèmes à milieu intérieur coopératif (Camps *et al.*, 1998)(Glize, 2001) :

Lemme 3 : *L'ensemble des systèmes à milieu intérieur coopératif est inclus dans l'ensemble des systèmes coopératifs*.

Lemme 4 : *Pour tout système coopératif, il existe au moins un système à milieu intérieur coopératif qui soit fonctionnellement adéquat dans le même environnement*.

Grâce aux quatre lemmes précédents, le théorème suivant est prouvé : *Pour tout système fonctionnellement adéquat dans un environnement donné, il existe un système à milieu intérieur coopératif qui réalise une fonction équivalente*.

Ainsi, il est possible de produire des systèmes fonctionnellement adéquats en concevant des systèmes à milieu intérieur coopératif ou capables d'atteindre et maintenir un milieu intérieur coopératif. Dans le cadre des systèmes multi-agents adaptatifs, cela signifie que concevoir un SMA se réduit à concevoir les agents qui le composent de manière à ce qu'ils soient capables de se maintenir dans des situations coopératives. Le théorème d'adéquation fonctionnelle nous garantissant que si tel est le cas, le système dans son ensemble aura alors une activité coopérative et effectuera donc ce que l'on attend de lui. De plus, les agents n'ont pas à connaître la fonction que doit effectivement réaliser le système dans son ensemble : les fonctions réalisées pourront donc être émergentes au sens de la condition donnée en fin de la partie 1.1.2.5.

1.2.2 Auto-organisation par coopération

Nous avons vu en 1.1.2.3 que l'apprentissage dans les SMA pouvait se focaliser sur l'agent, les interactions, l'environnement ou bien l'organisation. L'auto-organisation se situe dans le champ de l'apprentissage centré organisation. En effet, considérons un système S plongé dans un environnement E et effectuant une fonction f_s (voir figure 1.1). Cette fonction étant issue de la composition des fonctions réalisées par les parties P_i de S, modifier la manière dont ces parties sont organisées (cette organisation déterminant la composition des f_{P_i} qui génère la fonction f_s du système), modifie la fonction globale réalisée par S. Ces modifications organisationnelles peuvent correspondre à des ruptures, des créations ou des modifications d'interactions entre des parties.

Ce *principe d'auto-organisation* est un phénomène omniprésent dans la nature, et ce, de l'échelle

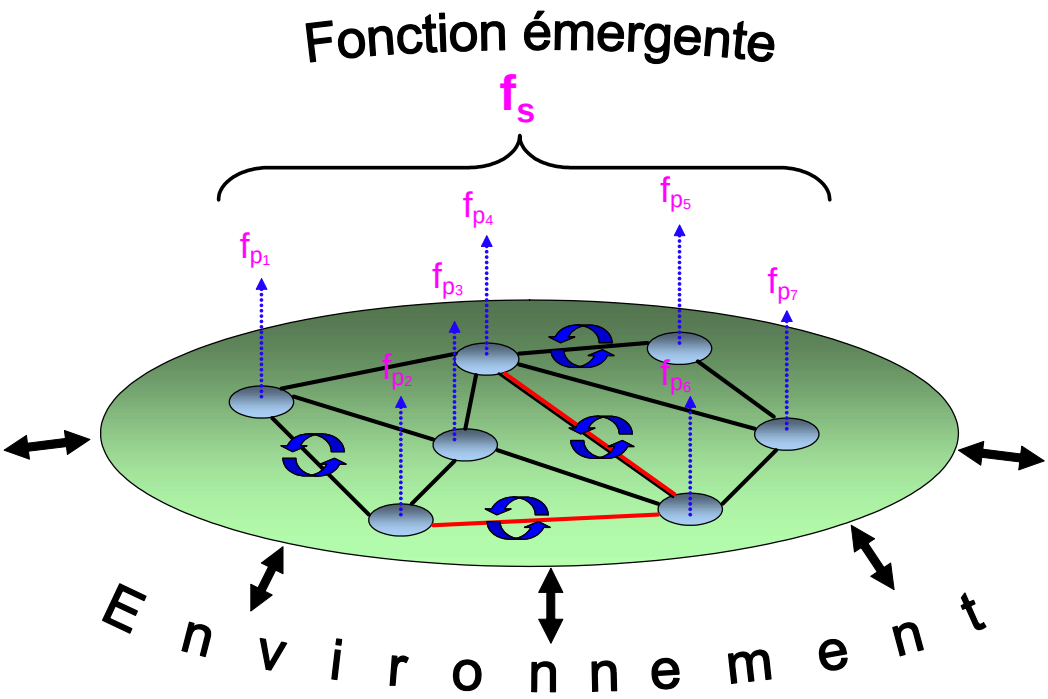

Figure 1.1 — Principe de l'auto-organisation

microscopique (cellules de convection hexagonales de l'expérience de Bénard) jusqu'aux phénomènes d'ordre social (déplacements collectifs auto-organisés d'insectes, par exemple). De manière générale, le principe d'auto-organisation permet aux systèmes naturels de produire des phénomènes complexes à partir de mécanismes très simples. Ainsi, les colonies d'insectes sociaux exhibent des comportements globaux complexes (la fabrication de nids, par exemple) et adaptatifs (lors de la récolte de nourriture, par exemple). C'est en couplant ce principe d'auto-organisation au théorème d'adéquation fonctionnelle que nous pouvons construire des systèmes multi-agents adaptatifs. Reste à préciser le mécanisme utilisé pour atteindre un état coopératif : il repose, dans la théorie AMAS, sur des agents ayant une certaine attitude sociale et sachant repérer localement les situations non coopératives ainsi que les résoudre.

1.2.2.1 Situations non coopératives (SNC)

Afin de définir ce que sont les situations non coopératives, il convient de préciser ce qu'est la coopération dans la théorie AMAS. La notion de coopération peut s'appliquer tant à un système qu'à un agent particulier, et n'a de sens que dans l'échange d'informations (en perception comme en action) entre l'entité considérée et son environnement (qu'il soit physique ou social). Dans ce cadre, une entité est dite coopérative si :

$C_{perception}$ Tout signal perçu peut être interprété sans ambiguïté.

$C_{decision}$ Toute interprétation doit produire de l'information utile au raisonnement.

C_{action} Les conclusions (actions entreprises) doivent être utiles à autrui.

Toute situation qui entraîne l'infraction à l'une de ces propositions est dite non coopérative. Ainsi, du point de vue d'un agent, sont considérées comme non coopératives les situations :

D'incompréhension : l'agent ne peut interpréter un signal qu'il perçoit ou alors ce dernier est ambigu.

D'ambiguïté : le contenu d'un signal perçu est incomplet ou l'information n'est pas assez précise et peut donc être interprétée de plusieurs manières par l'agent.

D'incompétence : l'agent ne peut répondre aux demandes d'un autre agent ou à une activité non correcte.

D'improductivité : une information reçue n'entraîne pas de conséquence logique c'est-à-dire aucun changement dans l'état interne de l'agent.

D'inutilité : les conclusions de l'agent aboutissent à une action qui n'est pas utile à autrui ou à l'environnement.

De concurrence : le résultat ou l'information déduit par l'agent l'a aussi été par un autre agent. Cet état de fait peut être profitable pour décharger des agents trop sollicités mais peut être néfaste ou inutile si trop d'agents effectuent le même travail (les agents d'un SMA sont généralement voués à être complémentaires).

De conflit : il provient généralement de l'accès multiple à des ressources communes limitées ou non partageables. Un conflit peut aussi intervenir si l'action entreprise par un agent gêne ou annule celle d'un autre.

Cette caractérisation des situations non coopératives est générique. Lors de la conception d'un système informatique basé sur la théorie AMAS, le concepteur doit déterminer de manière exhaustive, pour chaque type d'agent, l'ensemble des situations non coopératives ainsi que les actions relatives à mettre en oeuvre afin de les résoudre au mieux. Cette tâche n'est évidemment pas triviale et représente l'effort majeur de conception dans les systèmes développés selon cette théorie. Il est à noter que toutes les situations non coopératives n'apparaissent pas forcément dans tous les agents : nombre d'entre elles sont souvent évitables lors de la conception même des agents. Par exemple, les situations non coopératives d'incompréhension générées par des problèmes de langage ne peuvent apparaître lorsque l'ensemble des agents est homogène (ils partagent alors tous le même langage). De la même manière, les situations d'ambiguïté sont souvent évitées dès la conception des agents en les dotant d'une capacité exhaustive de traitement des messages.

1.2.2.2 Attitude coopérative

La coopération induit aussi pour les agents la nécessité d'une attitude coopérative. En effet, certaines situations non coopératives peuvent être évitées grâce à une attitude idoine sous-tendant les actions des agents. L'agent doit être sincère (il communique seulement les informations qui lui semblent véridiques), serviable (il cherche à atteindre l'objectif qui lui a été assigné), bien intentionné (il transmet spontanément les informations utiles à autrui) et il doit faire confiance aux autres agents (il les croit serviables, sincères et bien intentionnés). Cette attitude coopérative est traduite de deux manières majeures dans les agents.

D'une part, elle induit certains mécanismes tel que, par exemple, la relaxation restreinte de messages. La relaxation restreinte est notamment utile lors de la réorganisation des agents. Ayant une perception et une connaissance limitées de leur environnement, les agents ne connaissent parfois aucun autre agent susceptible (de par ses compétences, par exemple) de les aider pour résoudre un problème donné. Étant confiant, un agent A donné peut néanmoins envoyer un message de demande d'aide à ses accointances (mettons les agents B et C) et ces derniers relaxeront le message vers d'autres agents inconnus de l'agent A. Ce type de mécanisme permet de pallier nombre de problèmes liés à la distribution des connaissances dans les systèmes multi-agents lorsque les agents ont des accointances et des moyens de perception limités, et ce de manière efficace (Camps, 1996).

D'autre part, l'attitude coopérative permet d'"optimiser" le traitement des situations non coopératives vues précédemment. Le traitement associé à une situation non coopérative peut ne pas être unique et dépendre de la situation et des connaissances de l'agent. La coopération peut servir ici de critère de décision : l'agent mettra en oeuvre l'action la plus coopérative ou la moins non coopérative (principe du moindre préjudice (Georgé, 2004)).

1.2.3 Intérêts, contraintes et problèmes

Comme nous venons de l'exposer, la théorie des AMAS propose un nouveau paradigme pour le développement de logiciels adaptatifs : des systèmes multi-agents auto-organisateurs basés sur la coopération. Si cette approche exhibe de nombreux avantages potentiels, son application n'est pas sans contrainte au niveau de la conception et de la mise en oeuvre. De plus, la théorie des AMAS se veut une contribution au domaine des systèmes complexes et, dans une certaine mesure, des théories de l'émergence soulevant par là-même nombre de questions et butant sur de nombreux problèmes.

1.2.3.1 Intérêts d'une "néo-computation"

La théorie des AMAS se veut une réponse aux problématiques posées par la conception des nouveaux systèmes informatiques. La complexité des applications ne cessant de croître, de nombreux scientifiques et industriels s'intéressent à l'utilisation de systèmes adaptatifs qui permettrait de passer outre la conception classique par décomposition modulaire qui se révèle impuissante à garantir la stabilité et la maintenance.

IBM a proposé l'"*autonomic computing*" comme nouveau challenge scientifique en partant du constat suivant : "*Even if we could somehow come up with enough skilled people, the complexity is growing beyong human ability to manage it. As computing evolves, the overlapping connections, dependencies, and interacting applications call for administrative decision-making and responses faster than any human can deliver. Pinpointing root causes of failures becomes more difficult, while finding ways of increasing system efficiency generates problems with more variables than any human can hope to solve. Without new approaches, things will only get worse*"(Horn, 2001). L'objectif est donc de construire des systèmes capables d'ajuster leur comportement face à des circonstances variées et de préparer leurs ressources pour gérer au mieux leur charge de travail. La théorie des AMAS s'inscrit clairement comme une réponse potentielle pour ce type de problème puisque, d'une part, elle propose intrinsèquement la création de systèmes adaptatifs et, d'autre part, elle est basée sur le paradigme multi-agent qui induit un développement a priori plus simple dans le sens où la conception des agents composant le système est moins complexe (la composition des parties du système relevant a priori de la théorie par le biais du principe d'auto-organisation et non de la conception elle-même).

L'*informatique diffuse* (Drogoul et Servat, 2001) est un autre type de problématique auquel l'informatique est vouée à se heurter : "*The "UC"*[5] *era will have lots of computers sharing each of us. Some of these computers will be the hundreds we may access in the course of a few minutes of Internet browsing. Others will be imbedded in walls, chairs, clothing, light switches, cars - in everything. UC is fundamentally characterized by connections of things in the world with computation. This will take place at a many scales, including the microscopic*" (Weiser et Brown, 1996). Il est donc nécessaire de pouvoir gérer des systèmes composés intrinsèquement d'un nombre variable et probablement important d'artefacts, rendant impossible la mise en oeuvre d'un contrôle centralisé et dont la tâche globale est, a priori, impossible à spécifier. Ici encore, la théorie des AMAS est, de par sa base multi-agent, bien adaptée à la nature distribuée et ouverte de ces systèmes. De plus, elle apporte ici une réponse quant à la construction de systèmes dont la fonction n'est pas bien connue ou impossible à spécifier : le théorème d'adéquation fonctionnelle.

Au-dela d'une simple réponse à ces nouvelles difficultés, la théorie des AMAS permet aussi de

[5]Ubiquitous Computing

concevoir des "algorithmes" pour la résolution de problèmes, au sens large, plus classiques (Capera *et al.*, 2003). Son application a été fructueuse dans plusieurs domaines :

STAFF : la prévision de crues. Le but est que le système fasse une prévision de la hauteur d'eau à six heures (dans le futur). Chaque agent du système apour objectif de prévoir la variation de hauteur d'eau pour un délai unitaire de une heure et donc six agents sont nécessaires pour chaque capteur pour obtenir une prévision à six heures (ils sont nommés *agents horaires*). Les *agents horaires* sont eux-mêmes composés d'agents représentant des capteurs de hauteur d'eau (*agents capteurs*). Chaque *agent capteur* réalise, pour calculer sa prévision, une somme pondérée des valeurs fournies par tous les autres agents du système. Avant chaque calcul, les agents modifient les poids associés aux mesures des autres agents en fonction de la corrélation et de l'influence de celles-ci. Les résultats obtenus sont très bons puisqu'équivalents à ceux obtenus par des modèles mathématiques classiques prenant en compte les données topologiques, hydrologiques, etc. de la zone couverte par la prévision, ce qui demande un travail préalable important. A contrario, l'utilisation d'un système adaptatif permet au système d'être installé très rapidement et ne nécessite pas de calage préalable. De plus l'ajout d'une station, le retrait ou la mise hors service de capteurs ainsi que la prévision sur les stations en amont (ou nulle information de débit n'est disponible) sont possible de manière intrinsèque. Ce système est actuellement déployé sur le bassin de la Garonne et est en court de déploiement sur d'autres sites en France (Sontheimer et Glize, 2001).

ETTO : emploi du temps. Cette application permet la résolution des contraintes imposées par la création d'un emploi du temps dans le cadre d'une organisation de cours (enseignant et groupes d'étudiants), dans un nombre donné de salles aux caractéristiques diverses et sur une certaine période de temps (Bernon *et al.*, 2002). Chaque enseignant et chaque groupe d'étudiants sont représentés par un agent pour chaque cours (si un groupe d'étudiants doit suivre x cours, il existe x agents pour représenter ce groupe). Ces agents évoluent dans un environnement discret dont chaque emplacement correspond à une salle pour un créneau horaire donné. Les agents sont capables d'échanger des informations, de se déplacer de manière autonome et possèdent éventuellement un ensemble de contraintes (par exemple, si un enseignant n'est pas disponible sur un créneau horaire donné, tous les agents le représentant essaieront de se placer dans un emplacement de l'environnement qui ne correspond pas à ce créneau).

Résolution d'équations. Le problème consistait ici à trouver le poids des variables d'une équation à n inconnues. L'équation est donc de la forme $\sum_{i=0}^{n} a_i \cdot x_i = b$ où les a_i sont les poids des variable x_i et b la valeur de la somme. Affectant aléatoirement des valeurs initiales aux a_i, le but est d'ajuster ces valeurs pour que des couples $((a_0, a_1, ..., a_n), b)$ soient vérifiés. Pour résoudre ce problème, les poids a_i sont considérés comme des agents : ils ajustent coopérativement leurs

valeurs afin que la valeur b de la somme soit la bonne pour tout vecteur $(a_0, a_1, ..., a_n)$ proposé au système.

1.2.3.2 Contraintes de développement

Comme exposé en 1.2.2, le concepteur doit déterminer de manière exhaustive, pour chaque type d'agent, l'ensemble des situations non coopératives ainsi que les actions relatives à mettre en oeuvre afin de les résoudre au mieux. Cette approche de programmation proscriptive est néanmoins peu évidente à mettre en oeuvre, principalement à cause des biais induits par le fait qu'il est très difficile de prévoir toutes les réactions du système en fonction des choix algorithmiques (ou des modifications effectuées sur ces algorithmes) sur les actions à entreprendre pour résoudre les situations non coopératives. De plus, la question de l'exhaustivité de ces situations non coopératives est cruciale mais excessivement difficile à résoudre d'autant plus que la granularité des agents et la diversité d'informations sont importantes. Du point de vue de l'ingénierie, trois techniques permettant de résoudre ce problème se sont présentées jusqu'à présent.

Tout d'abord, comme pour tout processus empirique, l'expérience et l'intuition du concepteur se sont imposées. En effet, ne disposant pas de méthode complète ni d'une théorie parfaitement affinée (la méthode et la théorie étant les objets des recherches menées au travers des applications), on peut dire que les applications de la théorie des AMAS n'ont pu être réalisées que grâce à une grande dose d'intuition et d'expérience. Bien évidemment, si cette manière de procéder permet aux hommes de se forger une idée de plus en plus précise de ce sur quoi ils travaillent (un juste retour aux *Lumières*), l'expérimentation n'est qu'un moyen et non une fin. Jusqu'à présent, ce moyen a permis d'identifier la coopération comme invariant des applications - mettant sur pied ce que l'on nomme maintenant la théorie des AMAS, ainsi que la méthode ADELFE (Picard et Gleizes, 2004) (Picard, 2004) dédiée à cette théorie. L'identification, et la définition, de la coopération et de la non coopération ont permis de restreindre le champ de conception mais ne l'a pas encore parfaitement balisé : l'identification exhaustive des situations non coopératives repose toujours essentiellement sur le savoir-faire du concepteur. La méthode ADELFE apporte essentiellement des outils de support et un processus de développement orienté agent comprenant une aide à l'identification des situations non coopératives basée sur l'analyse des diagrammes de séquences et des protocoles d'interaction, mais sans garantir l'exhaustivité de l'ensemble des situations non coopérative trouvées. ADELFE ne propose pas non plus d'aide pour le développement des actions correctrices relatives à ces situations.

Dans la continuité de la méthodologie ADELFE, une seconde approche pourrait consister à faire évoluer les systèmes directement dans leur environnement afin de les concevoir alors qu'ils sont en cours d'exécution : c'est ce que nous appelons le *Living Design* (Georgé *et al.*, 2003a). Il s'agit de

commencer à développer un prototype en état de fonctionner (en se basant sur un modèle générique d'agent) afin de détecter les besoins et carences du système au fur et à mesure qu'il rencontre des problèmes. Ainsi, le concepteur pourrait modifier, ou préciser, la spécification d'un agent dès lors qu'un événement inattendu apparaît et qu'il peut en déduire que cela est dû à une limitation des compétences ou des aptitudes d'un agent. De la même manière, le concepteur pourrait modifier le comportement social des agents si des problèmes sont générés par les interactions entre les agents. Cette manière d'aborder la conception, qui se rapproche des développements dits itératifs, devra être supportée par des outils de *monitoring* et d'analyse (génération du diagramme de séquences correspondant à la trace de l'exécution, par exemple). Toutefois, l'état de développement d'ADELFE ne permet pas aujourd'hui d'utiliser un tel processus de conception, bien que cela figure parmi les priorités de développement de la méthode.

Enfin, il existe une solution toute théorique à ce problème de conception : concevoir les agents que l'on ne peut parfaitement spécifier comme autant de systèmes multi-agents. Comme nous l'avons vu en 1.1.2.4, l'idée est séduisante même si rarement mise en oeuvre en réalité. Cette approche permet entre autre de construire des agents coopératifs comme des systèmes multi-agents à milieu intérieur coopératif (cf. lemmes 3 et 4 dans 1.2.1). L'avantage attendu étant que les "sous-agents" - ceux composant l'agent posant problème - sont *a priori* de granularité plus faible et donc *potentiellement* plus faciles à spécifier.

1.2.3.3 Problèmes

De manière générale, les hommes construisent des théories dans le but d'expliquer, de prédire et de maîtriser des phénomènes. Les théories sont donc souvent basées sur un modèle de la réalité qui est construit sur la généralisation d'observations du réel. Stephen Hawking dans *"A Brief History of Time"* déclare *"a theory is a good theory if it satisfies two requirements : It must accurately describe a large class of observations on the basis of a model that contains only a few arbitrary elements, and it must make definite predictions about the results of future observations"*. Dans le cadre de ce que nous nommons *théorie des AMAS*, nous avons bien un large panel d'observations (tant dans la nature d'où l'on a puisé l'inspiration que dans l'expérimentation au travers des applications développées) ainsi que nombre d'idées et pistes (la définition de la coopération, l'étude de l'adéquation fonctionnelle, une définition pratique de l'émergence) mais toujours aucun modèle réellement développé. Si j'aborde ici le problème de la théorisation, c'est qu'il s'est imposé au centre des préoccupations de l'équipe SMAC d'une part, mais aussi dans mon propre travail de thèse. En fait, trois axes majeurs se dégagent des nombreuses questions qui se posent.

La première problématique concerne évidemment la dynamique des systèmes multi-agents adap-

tatifs. Comment prévoir et contrôler leur évolution (quand bien même l'on garantirait leur adéquation fonctionnelle) ? Pourquoi la coopération est-elle un critère (semble-t-il ?) valide pour diriger l'auto-organisation ? Existe-t-il d'autres critères alternatifs pour diriger l'auto-organisation ?

En second lieu, vient la question de la convergence. Peut-on établir une preuve générale quant à la convergence de ces systèmes ? Quels sont les caractéristiques de l'espace qu'ils parcourent (l'espace parcouru est relatif au critère de coopération et non à la fonction à atteindre) ? Pourquoi ces systèmes sont-ils si efficaces en ce qui concerne la vitesse (temps) et la qualité (ils semblent peu sensibles aux minima locaux) du processus d'adaptation ?

Et pour finir, l'étude du phénomène d'émergence. Pouvoir élaborer une théorie pour la conception de systèmes artificiels à fonctionnalité émergente permettrait de mieux comprendre le phénomène et, à terme, donner naissance à d'autres théories des systèmes artificiels ... ou non.

2 Conception de systèmes mécaniques

LORS de la conception d'un nouveau système mécanique (mécanismes de déploiement des volets pour les ailes d'avion, trains d'atterrissage, aubes à calage variable, tuyères ...) le concepteur doit, dans un premier temps, déterminer la meilleure architecture du mécanisme pour le problème donné. Cette étude repose généralement, et exclusivement, sur le savoir-faire de l'ingénieur et se révèle coûteuse en temps de travail, en temps de calcul, et est difficilement automatisable (espace de recherche très grand, nombre de composants du système non défini...).

Le nombre de combinaisons possibles entre les composants mécaniques élémentaires étant très grand, la résolution de cette tâche est complexe car elle entraîne le parcours d'un espace de recherche gigantesque. Cette complexité est accrue si l'on considère que :

- Le concepteur ne connaît pas forcément le nombre et le type de composants que le problème requiert ;
- Les modifications topologiques, même minimes, d'un mécanisme peuvent entraîner un grand changement pour la fonction qu'il effectue : l'espace des solutions est discontinu et il n'existe pas, a priori, d'heuristique pertinente.

De manière générale, on peut décomposer le travail de conception en quatre phases (Sen et Yang, 1998) (Pahl et Beitz, 1996) :

1. La *conception conceptuelle*[1] qui permet de déterminer le principe de la solution.
2. La *conception préliminaire* qui consiste en l'étude des solutions sélectionnées dans la première phase afin de choisir la meilleure relativement aux spécifications (Dym, 1994).
3. L'*intégration* est la phase où l'on passe du concept au produit. Durant cette phase sont déterminés les formes, matériaux et autres caractéristiques des différents composants en prenant en compte les contraintes techniques et économiques.
4. La *conception détaillée* complète la phase précédente pour achever le produit (décisions définitives sur les formes, dimensions, propriétés de surface, matériaux, etc.) et s'attache aux méthodes de production, coûts, etc.

Dans le domaine aéronautique, ce schéma de conception se traduit par deux grandes phases : la *synthèse* pour déterminer l'architecture du mécanisme et une première approximation des dimensions

[1] Traduction littérale de *Conceptual Design*

(regroupant les phases de *conception conceptuelle* et de *conception préliminaire*), et la *conception détaillée* qui consiste en l'étude du mécanisme sur un large domaine paramétrique (températures, efforts, déformations, poids, collisions, etc.) basée généralement sur des modèles en 3 dimensions et à éléments finis (regroupant les phases d'*intégration* et de *conception détaillée*).

Dans le cadre de notre étude, nous nous intéressons uniquement à la première phase de conception : le but est d'explorer l'espace de toutes les alternatives de mécanismes pour en extraire celles qui permettent de résoudre le problème spécifié (*type synthesis*) et de faire une première optimisation des dimensions. Je présente dans ce chapitre deux approches pour le traitement de cette première phase : la première est basée sur une méthodologie systématique alors que la deuxième est une approche orientée agent et évolutionnaire. Ce chapitre se termine sur les différentes méthodes que l'on peut utiliser pour effectuer l'optimisation des dimensions des mécanismes.

2.1 Type Synthesis

Cette première phase de conception est purement cinématique en ce qu'elle ne s'attache qu'à assembler des composants mécaniques (corps rigides, glissières, charnières etc.) afin d'obtenir un système effectuant une fonction objectif donnée relative au mouvement (suivre une trajectoire par exemple).

De plus, le concepteur peut vouloir satisfaire d'autres contraintes : types de composants pouvant être utilisés ou nombre maximal de composants par exemple. Cette conception préliminaire est composée de deux sous-phases :

1. Rechercher la topologie du mécanisme : c'est-à-dire déterminer le nombre de corps rigides du système et de quelle manière ils sont liés. Durant cette phase, on ne considère ni les dimensions des corps rigides ni le type des joints qui les lient. Les critères pris en compte sont d'un ordre plus "abstrait" tels que le nombre de degrés de liberté du mécanisme ou son espace dimensionnel (plan ou 3-Dimension).
2. Rechercher la meilleure alternative à partir de la topologie précédente : c'est-à-dire déterminer le type des joints.

Il n'existe pas actuellement d'outils industriels permettant d'accomplir automatiquement cette phase préliminaire de conception et donc, les ingénieurs ne peuvent se baser que sur leur expérience et leur intuition. Ils procèdent habituellement par essais-erreurs et utilisent des outils d'assistance à la conception existants (DAO, CAO, simulateurs...). Les systèmes produits à l'issue de cette analyse doivent être par la suite modifiés, voire abandonnés, lors de la phase de conception détaillée à la vue de nouveaux problèmes rencontrés, ce qui rend cette phase très coûteuse en temps. La conception de

systèmes mécaniques est donc un procédé de nature itérative dont les solutions ne sont généralement pas uniques. Récemment une approche méthodologique basée sur la génération systématique de mécanismes et leur classification pour trouver plus rapidement le meilleur mécanisme a été proposé par Lung-Wen Tsai ou encore l'utilisation de systèmes multi-agents qui a été utilisée par Campbell dans sa théorie du *A-Design*.

2.1.1 Méthodologie systématique

Cette méthode est basée sur la théorie des graphes pour représenter les types de mécanismes et sur l'analyse combinatoire pour générer les alternatives. Dans sa globalité, le processus se décompose en sept phases (Tsai, 2001, p. 3) :

1. Identifier les besoins fonctionnels (relativement aux spécifications).
2. Déterminer la nature du mouvement (mécanisme plan, sphérique ou spatial), les degrés de liberté, la complexité du mécanisme...
3. Identifier les caractéristiques structurelles relatives aux besoins fonctionnels.
4. Générer toutes les structures cinématiques qui satisfassent les caractéristiques structurelles en utilisant la théorie des graphes et l'analyse combinatoire.
5. Construire les mécanismes correspondant et évaluer qualitativement leur capacité à satisfaire les besoins fonctionnels.
6. Sélectionner le mécanisme le plus adapté et l'optimiser.
7. Passer à la phase de production.

Si cette méthodologie est somme toute classique du point de vue des étapes suivies, les phases de génération et de sélection des structures cinématiques sont intéressantes car automatisables.

2.1.1.1 Correspondance entre graphes et mécanismes

L'idée principale est de représenter les structures cinématiques par des graphes. Dans ces graphes, les noeuds représentent des joints et les arcs représentent les corps rigides[2]. Le tableau 2.1 donne la correspondance entre les éléments de la théorie des graphes et la représentation des structures cinématiques.

On notera :

F : degrés de liberté du mécanisme.

[2]notés *links* dans (Tsai, 2001). On peut remarquer que les corps rigides n'existent pas en réalité puisque tous les matériaux connus se déforment. Néanmoins pour cette étude on les considère comme strictement rigides.

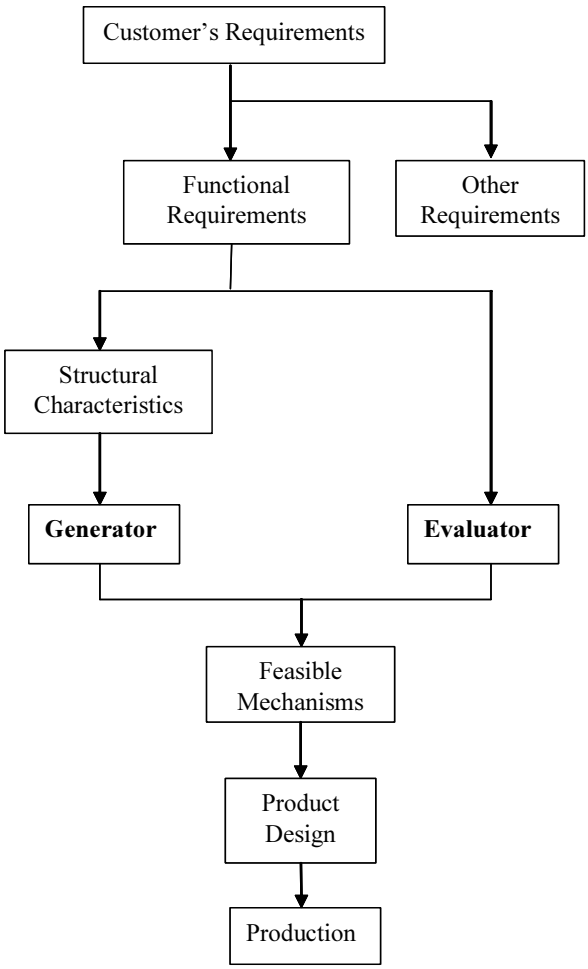

Figure 2.1 **—** Méthodologie systématique pour la conception de mécanismes

Graphe	Mécanisme	Notation
Nombre d'arcs	Nombre de corps rigides	n
Nombre de noeuds	Nombre de joints	j
Nombre d'arcs de degré i	Nombre de corps rigides de degré i	n_i
Degré du noeud i	Nombre de joints sur le corps i	d_i
Nombre de cycles indépendants	Nombre de cycles indépendants	L
Nombre total de cycles $(L+1)$	Nombre total de cycles $(L+1)$	$\widetilde{L}$
Nombre de cycles avec i noeuds	Nombre de cycles avec i joints	L_i

Tableau 2.1 **—** Correspondance entre graphes et mécanismes

f_i : degrés de liberté relatif permis par le joint i.

λ : degrés de liberté de l'espace dans lequel le mécanisme évolue.

De cette représentation, des équations portant sur les caractéristiques des mécanismes peuvent être extraites : on peut calculer le nombre de degrés de liberté, le nombre de cycles indépendants etc. Par exemple, le nombre L de cycles indépendants dans un mécanismes peut être exprimé en fonction du nombre de joints et de corps rigides présents dans la structure :

$$L = j - n + 1$$

Cette équation, dite *équation d'Euler*, peut être aussi relative au nombre total de cycles :

$$\widetilde{L} = j - n + 2$$

Ainsi, on peut générer l'atlas des chaînes cinématiques (cf. (Tsai, 2001, p. 255-260)) pour les chaînes cinématiques ayant jusqu'à trois cycles indépendants et huit corps rigides) et les classer selon différents critères permettant de guider le concepteur.

2.1.1.1.1 Degrés de liberté. L'étude des degrés de liberté d'un mécanisme est l'un des principaux centre d'intérêt lorsque l'on s'intéresse à leurs propriétés cinématiques. Le nombre de degrés de liberté correspond au nombre de paramètres indépendants nécessaires pour spécifier complètement la configuration d'un mécanisme dans l'espace (dans un espace à trois dimensions il y a six degrés de liberté : trois rotations et trois translations). On peut déduire ce nombre à partir du nombre de corps rigides, du nombre de joints et du nombre de degrés de liberté induit par chacun des joints d'un mécanisme grâce au critère de Grubler et Kutzbach :

$$F = \lambda(n - j - 1) + \sum_{i=0}^{j} f_i$$

2.1.1.1.2 Mobilité de cycle. Il se déduit du critère de Grubler et Kutzbach et de l'*équation d'Euler* :

$$\sum_{i=0}^{j} f_i = F + \lambda L$$

Ce critère est utile pour déterminer le nombre de degrés de liberté, induits par les joints, qu'il est nécessaire d'avoir pour obtenir un mécanisme ayant un nombre de degrés de liberté donné.

2.1.1.1.3 Représentation matricielle. Afin de faciliter le traitement informatique portant sur les graphes, il est possible de les représenter sous forme de matrices. Il existe de nombreuses représentations matricielles dont notamment celle sous forme de matrice d'adjacence ou celle sous forme de matrice d'incidence.

Une matrice d'adjacence A est définie de la manière suivante :

$$a_{ij} = \begin{cases} 1 & \text{si le corps rigide } i \text{ est connecté à l'arc } j \text{ par un joint} \\ 0 & \text{sinon (et notamment lorsque } i = j) \end{cases}$$

Ainsi, par définition, toutes les matrices d'adjacence sont de taille $n \times n$, symétriques et ont une diagonale nulle. Le type des joints reste non déterminé dans cette représentation, mais l'on peut associer des nombres ou des lettres aux différents types de joints pour remplacer les "1" de la matrice.

Dans une matrice d'incidence B, chaque ligne représente un corps rigide et chaque colonne un joint.

$$\begin{array}{cc} & \text{joint } j \\ B = & \begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,m} \\ b_{2,1} & b_{2,2} & \dots & b_{2,m} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n,1} & b_{n,2} & \dots & b_{n,m} \end{bmatrix} \quad \text{corps rigide } i \end{array}$$

avec

$$b_{ij} = \begin{cases} 1 & \text{si le corps rigide } i \text{ est connecté au joint } j \\ 0 & \text{sinon} \end{cases}$$

Ces matrices peuvent aussi servir à identifier mathématiquement les mécanismes structurellement isomorphes (qui partagent la même structure topologique). En effet, deux chaînes cinématiques dont les corps rigides peuvent être mis en correspondance deux-à-deux, sont en fait représentées par des matrices (notées A et B [3]) congruentes, donc si l'on a :

$$A = P^T B P$$

où P^T est la transposée de P. Théoriquement, la matrice P peut être dérivée de la matrice identité en modifiant l'ordre de ses colonnes.

2.1.1.1.4 Exemple des mécanismes plan. Pour simplifier, seuls sont considérés les mécanismes ne contenant que des joints rotoïdes (R) ou prismatiques (P). Cette classe de mécanisme est nommée *linkages*. Le nombre de degrés de liberté d'un *linkage* plan est égal au nombre de joints car les joints prismatiques et rotoïdes n'ont qu'un degré de liberté (rotation sur l'axe Z pour les rotoïdes, et translation sur le plan pour les prismatiques). De plus, l'espace plan permet seulement trois degrés de liberté ($\lambda = 3$), à savoir deux translations (en X et Y) et une rotation (sur Z), on obtient donc :

- $F = 3n - 2j - 3$ à partir du critère de Grubler et Kutzbach.

[3] Quelle que soit la représentation choisie pour A et B, i.e. que ce soient des matrices d'adjacente ou d'incidence

– $j = F + 3L$ à partir du critère de mobilité de cycle.

On peut déduire de l'équation d'Euler et du résultat précédent $n = F + 2L + 1$. Ainsi pour obtenir un degré de liberté donné, chaque ajout d'un cycle nécessite l'ajout de 2 corps rigides et de 3 joints.

Ainsi, l'ensemble des mécanismes est caractérisé par le nombre de degrés de liberté, le nombre de cycles indépendants, la liste des degrés d'arcs et les graphes contractés correspondants (Tsai, 2001, p. 261-273). A partir de cet atlas, le concepteur peut évaluer chacune des alternatives dérivées (en instanciant le type des joints) de ces chaînes cinématiques afin de déterminer celle qui correspond le mieux au problème à résoudre.

2.1.1.2 Sélection

Une méthode informatique pour sélectionner les mécanismes, à partir des atlas vus précédemment, et les évaluer d'un point de vue qualitatif a été développée par Alberto Cardona (Pucheta et Cardona, 2003). Elle se déroule suivant les trois étapes suivantes :

1. Sélection des chaînes cinématiques par isomorphisme de sous-mécanismes à l'aide du codage *Degree Code*.
2. Génération d'alternatives pré-dimensionnées de manière exacte.
3. Évaluation et optimisation à l'aide d'algorithmes génétiques.

2.1.1.2.1 Sélection des chaînes cinématiques. Le principe de cette phase est d'identifier dans l'atlas de chaînes cinématiques, celles dont des parties correspondent aux spécifications. En effet, lors des spécifications certains éléments du mécanisme (un sous-mécanisme donc) peuvent être prédéfinis (des points d'attachement, une actuation, etc.) et doivent donc faire partie du mécanisme. Toute chaîne cinématique incluant la sous-structure cinématique issue des spécifications sera donc sélectionnée.

La figure 2.2 illustre la transformation d'une chaîne cinématique (à gauche) en sa matrice d'adjacence (matrice A). A partir de la matrice A, est calculé le vecteur d des degrés de chaque corps rigide, qui correspondent à la somme de chacune des colonnes de A. Ce vecteur d est ensuite ordonné par permutation (le vecteur i représentant les indices de permutation) de ses éléments par ordre de grandeur pour obtenir le vecteur $d1$. En appliquant les mêmes permutations i à la matrice A, on obtient une nouvelle matrice A^1. Évidemment, plusieurs permutations i peuvent générer un même vecteur ordonné à partir de d (en inversant les deux premiers éléments "3", par exemple) : on note toutes les matrices d'adjacence permutées à partir de A sous la forme (A^j).

La figure 2.3 illustre le codage des matrices par association d'un *degree code*. Les matrices adjacentes étant toutes symétriques et à diagonale nulle, on peut coder de manière unique chaque matrice

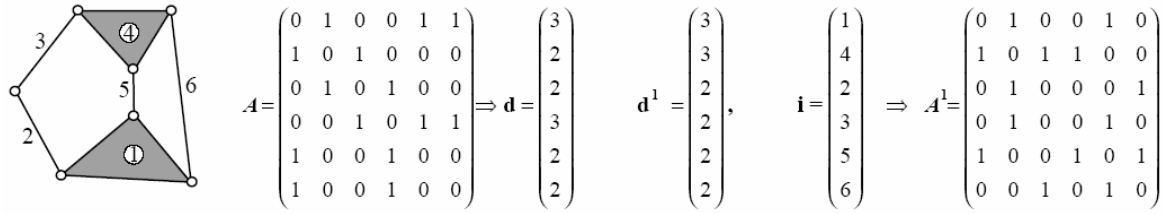

Figure 2.2 — De la chaîne cinématique aux permutations de la matrice adjacente

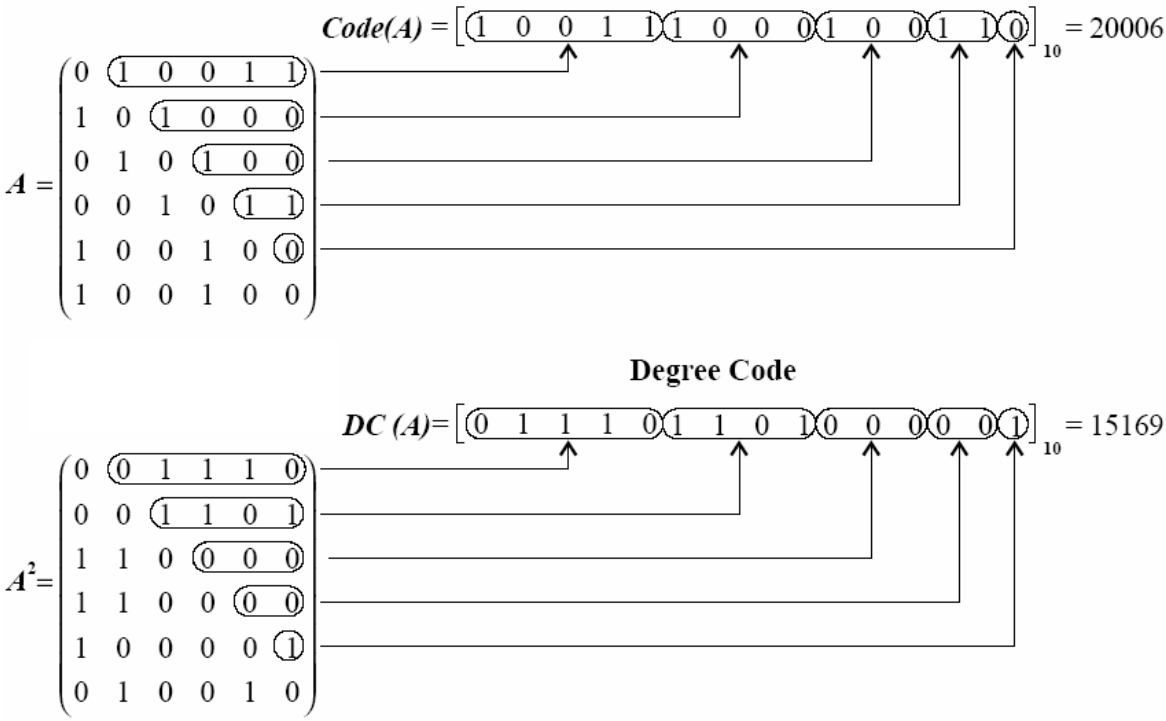

Figure 2.3 — *Degree Code*

par une chaîne de bits et donc par un entier (*CODE(A)* de la figure 2.3). En parcourant toutes les permutations possibles d'une matrice *A*, on peut trouver la matrice A^j ayant le plus grand *Code* et associer ce *MAXCode* à cette classe d'équivalence qui contient tous les mécanismes structurellement isomorphes. Ainsi, tester l'isomorphisme structurel de deux mécanismes est simplement réalisé par un test d'égalité de leur *Degree Code*.

2.1.1.2.2 Génération d'alternatives. Pour chacune des chaînes cinématiques sélectionnées, toutes les alternatives dérivées (en instanciant les joints) doivent être étudiées et un premier dimensionnement des éléments obtenu. Pour ce faire, chaque chaîne cinématique est tout d'abord représentée sous forme d'arbre ("Spaning Tree") et décomposée en diades, triades, etc. (cf. (Pucheta et Cardona, 2003, p. 10-13)). Il est ensuite possible de déterminer mathématiquement, pour chaque alternative, les dimensions du mécanisme pour respecter les spécifications fonctionnelles. Si le système

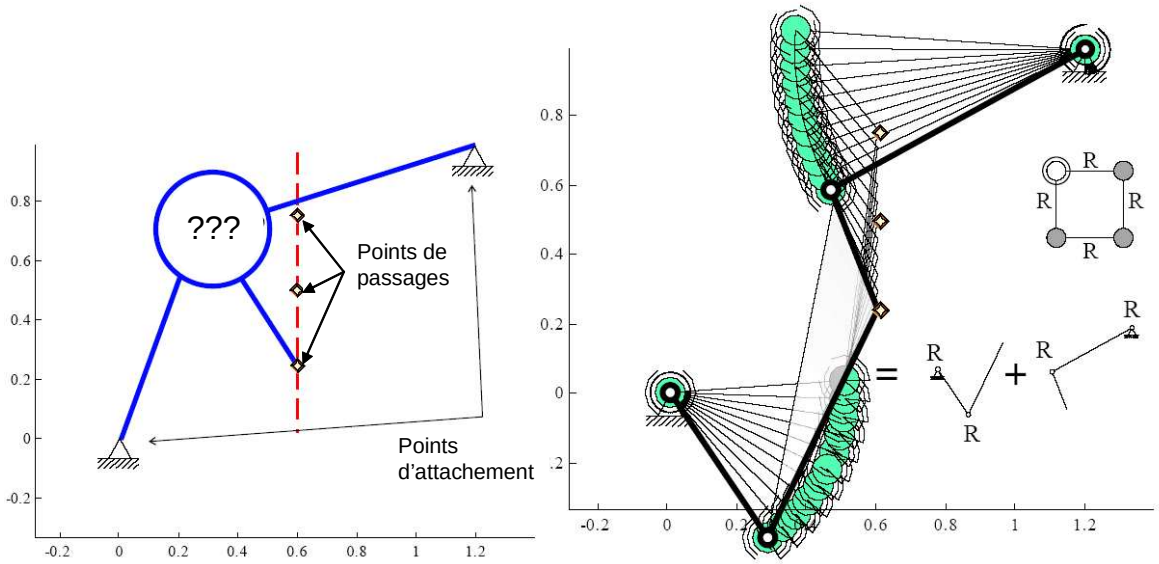

Figure 2.4 — Exemple de problème et solution 4-barres (RRRR)

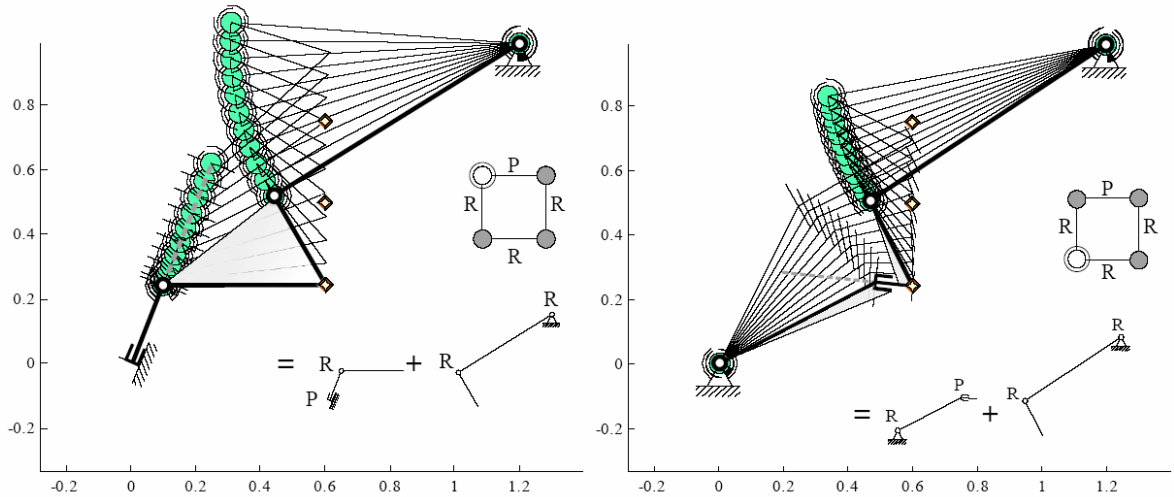

Figure 2.5 — Solution 4-barres (PRRR) (RPRR)

d'équation n'a pas une solution unique (s'il y a trop de paramètres libres relativement aux contraintes, il existe une infinité de solutions) un algorithme génétique est utilisé pour obtenir une solution.

La figure 2.4 donne un exemple de l'application de cette méthode pour résoudre le problème d'un mécanisme plan à un degré de liberté passant par trois points de passages spécifiés en ayant prédéfini deux points d'attachement. Une solution trouvée (partie droite de la figure) est un mécanisme 4-barres avec quatre charnières pour joints (rotoïdes). D'autres solutions peuvent être générées pour cette topologie avec des joints différents (**R**otoïdes ou **P**rismatiques), comme l'illustre la figure 2.5.

2.1.1.2.3 Évaluation et optimisation. Générer les mécanismes de manière exacte par cette méthode mathématique ne résout toutefois pas certains problèmes de cinématique : comme le nombre de points de passage prescrit est généralement faible, les solutions trouvées peuvent passer par des singularités ou des modifications de sens de mouvement. En effet, la génération exacte de mécanisme permet de respecter les points de passage mais ne contraint pas la cinématique en dehors de ces points. Pour améliorer les mécanismes obtenus, ils sont traités par un algorithme génétique.

2.1.1.3 Analyse

La pierre angulaire de cette méthode est donc l'énumération des structures cinématiques relativement aux caractéristiques fonctionnelles des spécifications. Cette énumération s'appuyant sur la théorie des graphes (utilisée pour représenter les structures cinématiques) et l'analyse combinatoire, elle permet de définir un processus systématique et automatisable menant à la génération d'atlas contenant les chaînes cinématiques adéquates vis-à-vis des spécifications. A partir de cet atlas, la génération exacte et la sélection de mécanismes peuvent être effectuées, elles aussi, de manière automatique en exploitant la propriété d'isomorphisme de structures et la résolution de systèmes d'équations pour obtenir les dimensions et positions des corps rigides. Pour pallier les défauts de l'analyse discrète (du point de vue de la fonction objectif), un algorithme génétique est utilisé pour améliorer les mécanismes résultants grâce à la modifications des dimensions, ce qui permet d'éviter les singularités, modifications de sens de mouvement, etc.

Le problème majeur de cette méthode est tout de même l'explosion combinatoire associée à la génération des atlas et des alternatives. Premièrement, la taille de l'atlas généré relativement aux spécifications fonctionnelles est conséquente : pour exemple, si l'on considère simplement l'atlas pour des mécanismes plan à un degré de liberté et avec au plus huit corps rigides, on obtient déjà 19 chaînes cinématiques possibles (Tsai, 2001, p. 261-273). Ensuite, lorsque l'on dérive les alternatives (en instanciant les joints), ce nombre s'accroît encore : en restant dans le domaine des mécanismes plan et en se restreignant aux seuls joints prismatiques et rotoïdes, on obtient 2^n mécanismes où n est le nombre de joints (le concept d'isomorphisme permet toutefois de réduire une part de cette explosion combinatoire (Pucheta et Cardona, 2003)).

2.1.2 A-Design

L'approche *A-design* se définit comme *"a new search strategy for the conceptual stages of engineering design that incorporates **A**gent collaboration with an **A**daptive selection of **Design**s"* (Campbell *et al.*, 1999). Le processus de résolution s'appuie sur un algorithme évolutionniste dont les phases de génération, sélection et reproduction sont effectuées par des systèmes multi-experts basés sur le

paradigme agent (un agent par expert). L'ensemble de la *théorie du A-Design* est présentée en détail dans la thèse de Matthew Ira Campbell (Campbell, 2000).

2.1.2.1 Un processus itératif

La figure 2.6 décrit le processus global de conception. A partir de la spécification du problème à résoudre (description formelle des comportements d'entrée et de sortie), le système génère une première population d'alternatives grâce à un ensemble d'agents nommés *maker-agents*. Ces solutions sont évaluées selon les critères définis par l'utilisateur et classées en trois catégories (cf. partie 2.1.2.2 ci-dessous). Les alternatives classées dans les deux meilleures catégories sont retenues pour l'itération suivante et sont donc transmises à un module de modification. Un ensemble de *modification-agents* en interaction vont alors essayer de raffiner les solutions retenues en ajoutant, éliminant ou altérant les éléments qui les composent avant de recommencer le cycle d'optimisation avec le nouvel ensemble d'alternatives obtenu. Les solutions de la meilleure catégorie (les *pareto designs*) sont présentées à l'utilisateur pour obtenir un feedback qui permet d'améliorer les traitements grâce à l'apprentissage : les agents assurant cette interface entre l'utilisateur et le système (les *manager-agents*) injectent les préférences exprimées par l'utilisateur lors de la communication pour modifier les lois d'activation ainsi que les populations des *maker-agents* et des *modification-agents*.

2.1.2.2 Sélection

L'étape de sélection consiste en la classification des mécanismes générés par les *maker-agents* en trois classes :

1. Les *pareto designs* qui correspondent aux meilleures alternatives (celles qui sont pareto-optimales du point de vue des spécifications).

2. Les *good designs* qui correspondent aux meilleures alternatives relativement aux préférences courantes du concepteur. Généralement, ces alternatives sont un sous-ensemble ($\approx$ 50%) des *non-pareto designs* qui correspondent au mieux aux préférences du concepteur.

3. Les *poor designs* qui correspondent aux alternatives restantes (les plus mauvaises).

L'utilisation du critère de Pareto optimalité permet de déterminer l'ensemble des meilleures alternatives du point de vue des objectifs et des contraintes spécifiés sans les réduire à un simple vecteur, une somme pondérée, etc. Cet ensemble est défini comme suit (Eschenauer *et al.*, 2000) :

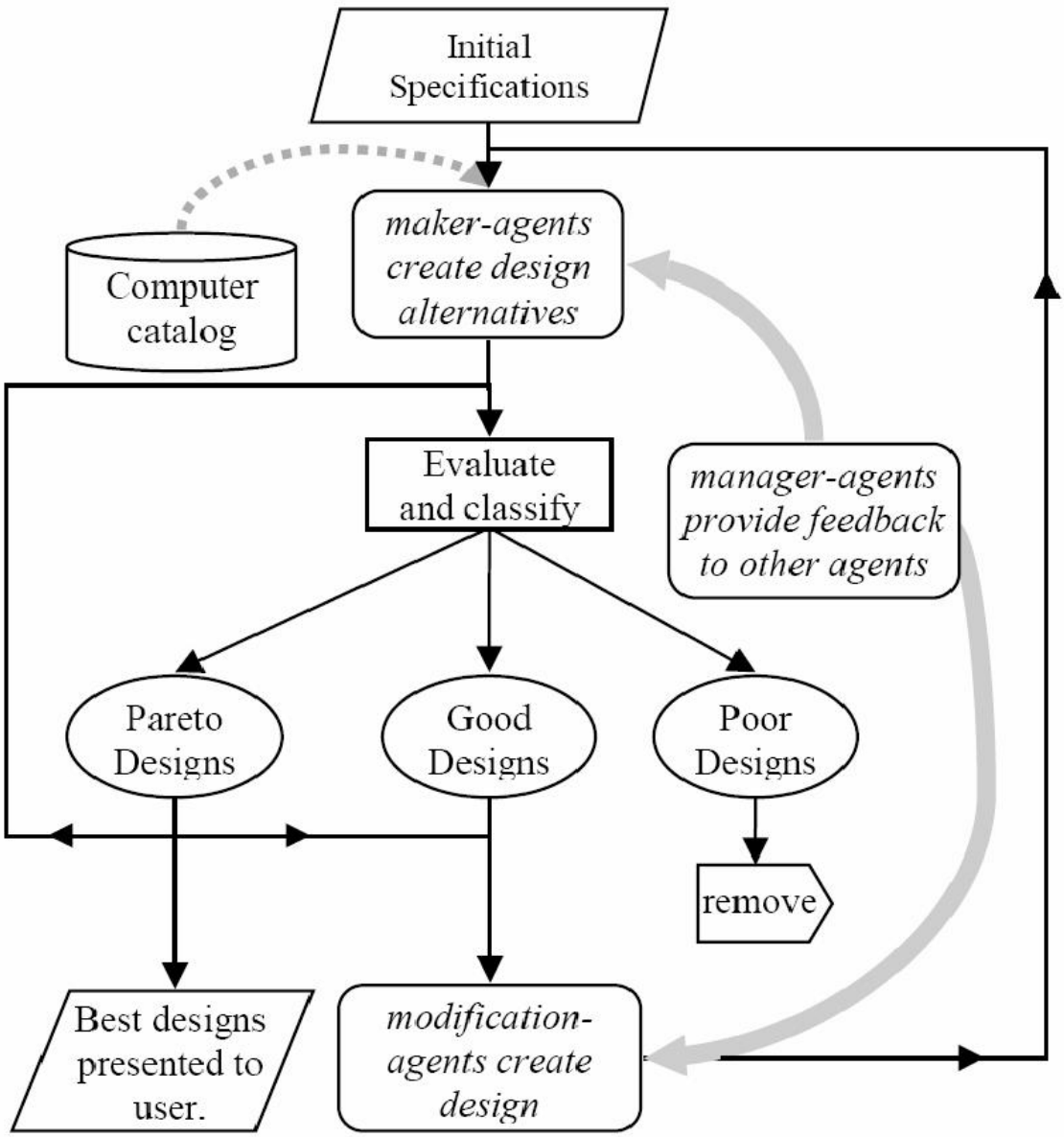

Figure 2.6 **—** Processus global du A-Design

$x^* \in X$ is a pareto optimal point if and only if

there is no other vector $x \in X$ such that

$f_j(x) \leq f_j(x^*) \forall j \in \{1, ..., m\}$

and

$f_j(x) < f_j(x^*) \exists j \in \{1, ..., m\}$

où f_i correspondant à la fonction d'évaluation relative à l'objectif i, m est le nombre total d'objectifs, et X l'ensemble des alternatives. Après cette classification, les *poor designs* sont écartés de l'ensemble des solutions pour la suite du processus : la population des alternatives retenues ne contient donc que des mécanismes qui sont intéressants soit du point de vue des préférences de l'utilisateur (par le biais des *good designs*), soit du point de vue des spécifications et contraintes (par le biais des *pareto designs*) assurant ainsi une diversité selon ces deux axes.

2.1.2.3 Représentation

Le processus du A-Design faisant intervenir des agents "experts" pour modifier les mécanismes, une attention particulière a été apportée à la représentation. Traditionnellement, une simple représentation, telle qu'un vecteur de variables représentant des paramètres physiques, est adoptée ; ce vecteur étant ensuite, au cours de l'optimisation, modifié de manière stochastique (algorithmes génétiques) ou bien selon les états passés (méthodes basées sur le gradient). Ainsi la manière dont évolue la représentation n'est généralement pas "logique" du point de vue des objets qu'elle modélise. C'est pourquoi la représentation des éléments mécaniques dans le A-Design est "'plus "naturelle que pratique" dans le sens où elle ne dépend pas directement de la manière dont les objets seront manipulés durant le processus d'optimisation.

Les éléments sont donc représentés en terme d'interfaces d'entrée/sortie (*Functional Parameters*) et de comportements (*Embodiment*) reprenant ainsi le type de représentation que l'on retrouve dans les travaux de Welch (Welch, 1992). Afin de pouvoir étudier analytiquement le comportement des mécanismes ainsi modélisés, l'ensemble des équations comportementales correspondantes peut être généré automatiquement. En parallèle au développement de cette représentation, une méthode de construction et de mise à jour des mécanismes a été développée en se basant sur les travaux de Schmidt et Cagan (Schmidt et Cagan, 1998) sur les grammaires fonctionnelles.

Functional Parameters (FP). Cette structure de données fournit une description tant qualitative que quantitative des comportements électro-mécaniques des composants. Elle donne ainsi une représentation des relations entre les composants du mécanisme et est considérée comme une interface

	Translational	Rotational	Electrical	Hydraulic
Through Variable	Force (f [Newtons])	Torque (T [N-m])	Current (I [amps])	flow rate (m [kg/s])
Across Variable	Velocity (v [m/s])	angular speed (Ω [rad./s])	Voltage (v [volts])	Pressure (P [N/m2])
Through α d(Across)/dt	mass	Rotational	inertia	capacitor reservoir
Through α Across	Damper friction	Damper friction	resistor	Valve viscous drag
Through $\alpha \int$(Across) dt	spring	Rotational spring	inductor coil	long piping

Tableau 2.2 — Variables par domaine

d'entrée/sortie. A chaque élément sont donc associés deux *FP* : l'un pour décrire l'entrée (FP_{input}) et l'autre pour décrire la sortie (FP_{output}). Les *FP* sont composés de sept champs :

1. Through {number, range, bound, unbound}
2. Across-integral {number, range, bound, unbound}
3. Across-none {number, range, bound, unbound}
4. Across-differential {number, range, bound, unbound}
5. Class {power, signal, material}
6. Domain {translational, rotational, electrical, hydraulic}
7. Coordinate {[4 x 4] transformation matrix}
8. Interface {standard size}
9. Direction {source, sink}.

Les variables de type *Through* correspondent aux paramètres représentant des phénomènes qui "traversent" les éléments. A contrario, les variables de type *Across* sont celles qui opèrent selon leur différence d'un côté à l'autre des éléments. Une énumération des variables par type pour chaque domaine est donnée dans le tableau 2.2 (Campbell, 2000, p. 51).

Embodiment. Cette structure de données contient les informations permettant de décrire, par le biais d'équations et de contraintes, les relations et lois d'évolution des entrées et des sorties des éléments. La structure contient les champs suivants :

1. Variables : nom des paramètres internes du composant.

2. Constraint Parameters (CP) : contraintes relatives aux *FP* pouvant être connectés au composant.
3. Magnitude Change(MG-Change) : fonctions décrivant les lois d'évolution entre les variables des *FP*.
4. Position Change (PO-Change) : matrice de conversion des coordonnées d'un port vers un autre port. Ces matrices permettent d'évaluer le champs *Coordinate* des *FP* connectés au composant.

Le but est de représenter le comportement de classes de composants et non pas de composants spécifiques : par exemple, l'*Embodiment* d'un ressort contient les équations et contraintes qui permettent de décrire le comportement général d'un ressort mais n'intègre pas les valeurs numériques spécifiques comme la raideur. Le champ *Constraint Parameters* est essentiellement exploité par les *Maker-Agents* qui sont chargés de connecter les composants ensemble pour générer de nouvelles configurations de mécanismes : ces contraintes permettent d'éviter les combinaisons non viables d'éléments. Les autres champs de la structure sont, pour leur part, utiles aux *Instantiation-agents* (cf. 2.1.2.4) : les variables seront instanciées par ces agents et les fonctions comportementales permettront la génération des équations nécessaires pour juger du comportement global du système.

Mise à jour des mécanismes. La représentation des composants à l'aide des deux structures de données que sont les *Functional Parameters* et les *Embodiments*, permet la construction itérative des mécanismes (un nouvel élément est rajouté à la fois). Quand un agent connecte un nouvel élément dans un mécanisme incomplet, tous les *FP* doivent être mis à jour au travers du système. Cette notion de mise à jour des mécanismes s'appuie sur les travaux de Schmidt et Cagan (Schmidt et Cagan, 1998) dans lesquels les mécanismes sont générés à l'aide de grammaires fonctionnelles.

Dans le modèle du A-Design, lorsqu'un composant est connecté au mécanisme cela engendre des modifications dans le *FP* auquel il se connecte puis, de proche en proche en "traversant" les *Embodiments*, dans les autres *FP* du système. La figure 2.7 illustre cette mise à jour lors de l'ajout d'un ressort lié au sol (modélisé par un FP_{gnd}) (Campbell, 2000, p. 55-59). Alors qu'initialement (figure du haut) il reste deux objectifs non satisfaits sur des variables *across-integrated* dans les FP_{input} et FP_{output}, suite à l'introduction du ressort et des mises à jour des différents *FPs* ces deux objectifs sont atteints.

Équations. Lorsqu'une configuration est complète, des équations sont extraites du modèle pour déterminer l'effet analytique de l'ensemble des *Embodiments* sur le comportement du système. Pour chaque variable de but (variables des FP_{input} et FP_{output}) une équation décrivant sa loi d'évolution est générée. L'extraction de ces équations se fait grâce aux informations contenues dans les *Embodiments*. Ces équations sont essentiellement utilisées par les *Instanciation-agents* pour dériver les

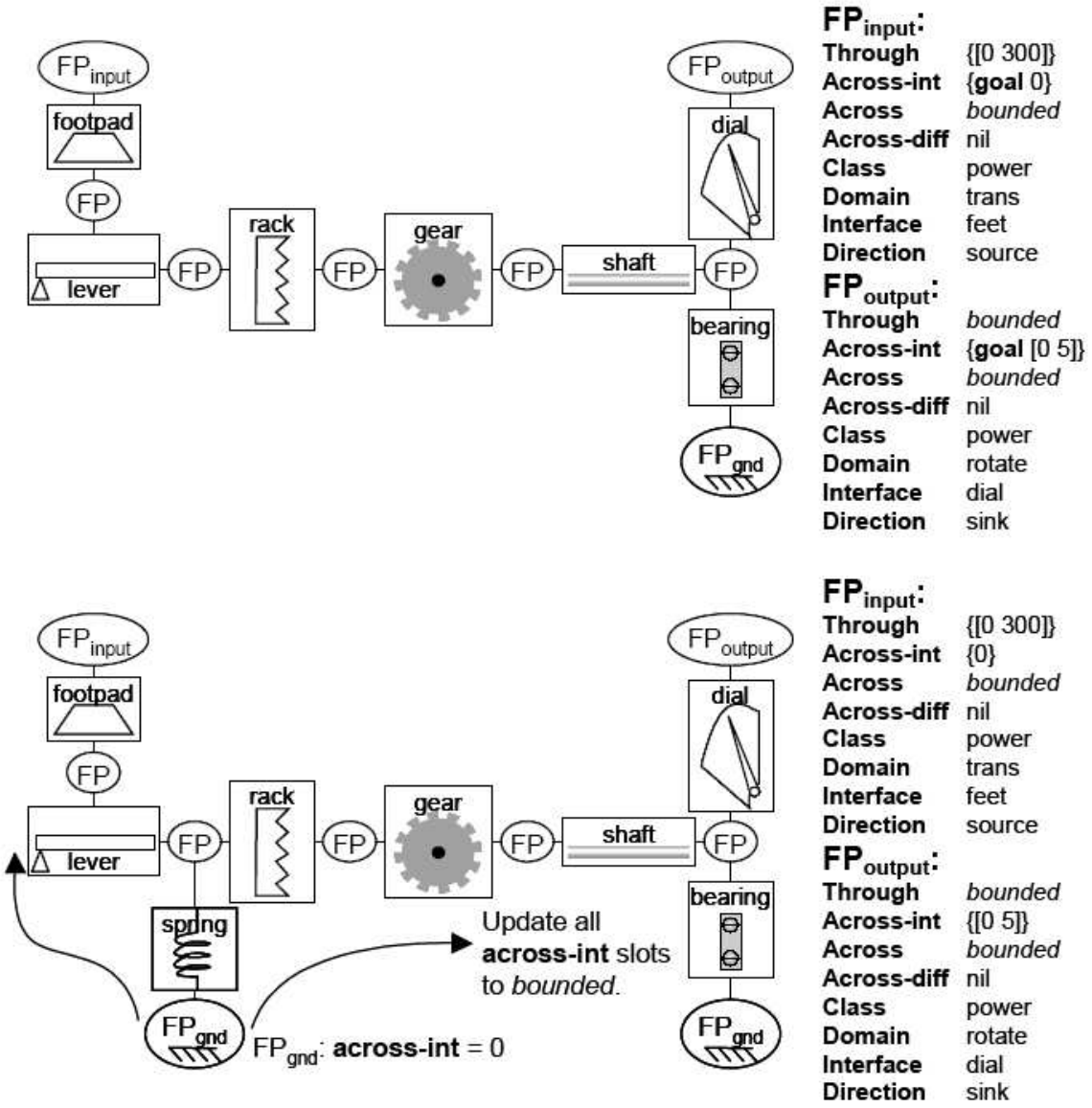

Figure 2.7 — Exemple d'introduction d'un ressort dans un mécanisme incomplet

C-Agents (48 agents)	I-Agents (18 agents)	F-Agents (72 agents)
Domain Preference electrical EBs translational EBs rotational EBs hydraulic EBs	**Objective Preference** prefer inexpensive components prefer lightweight components prefer efficient components	**Design Preference** modify expensive designs modify inexpensive designs modify heavy designs modify light designs
Parallel vs. Series connect EBs in parallel connect in series	**Variable Preference** select component based on variables present in behavioral equations select component based on variables present in FP interfaces	**Objective Preference** remove expensive components remove heavy components remove inefficient components
Source vs. Sink connect to source FPs connect to sink FPs	**Value for Instantiated Variable** High Medium Low	**Degree of Fragmentation** remove component from design remove EB from design
Other EB connections link new EB to existing FPs link new EB to new FP link new EB to ground FP		**EB Fragmentation** remove repeated EBs remove dangling EBs remove ground connections

Figure 2.8 — Préférences des différents types d'agents

Embodiments en composants réels et pour choisir les valeurs des variables de ces composants.

2.1.2.4 Les agents

Les agents ont un rôle central dans le A-Design : ce sont eux qui gèrent les créations et modifications de mécanismes. Chaque agent est capable d'effectuer une tâche précise (par exemple, ajouter un type de composant donné) selon ses propres stratégies et préférences. Dans le A-Desgin pour les systèmes électro-mécaniques, il existe trois grands types d'agents : les *Maker-agents* qui rajoutent des éléments, les *Modification-agents* qui modifient les mécanismes (notamment en retirant des éléments) et les *Manager-agents* qui contrôlent les populations et les exécutions des autres agents. Chaque classe d'agent peut se décliner en sous classes spécifiques d'agents : par exemple, pour les systèmes électro-mécaniques il a été conçu deux sous-classes de *Maker-agents* : les *Configuration-agents* (il en existe 48 sortes différentes) et les *Instantiation-agents* (déclinés en 18 types).

Maker-agents. Les *Maker-agents* effectuent deux types d'actions : ils créent de nouveaux mécanismes à partir de la spécification qui leur a été fournie et ils complètent les mécanismes qui leur sont retournés par les *Modification-agents*. Ces activités sont réalisées grâce à deux sous-classes : les *Configuration-agents* qui ajoutent de nouveaux composants (en fait de nouveaux *Embodiments*) dans la représentation du A-Design ce qui permet d'explorer l'espace des configurations possibles de mécanismes, et les *Instantiation-Agents* qui instancient chacune des configurations en mécanismes (chacun des *Embodiments* est instancié en un composant réel avec des valeurs précises pour ses variables).

Lorsqu'un *Configuration-agents* est activé, il examine une configuration incomplète et essaie de déterminer, en fonction de ses préférences, quel est le meilleur endroit où améliorer la configuration et avec quel type de composant. Comme l'ajout d'un *Embodiment* peut être effectué en le connectant à n'importe quel *FP* présent, les *Configuration-agents* analysent l'utilité de l'ajout de tous les types d'*Embodiment* possibles sur chacun des *FP* du point de vue des spécifications et de ses préférences (cf. figure 2.8).

Les *Instantiation-agents* affectent des composants réels (depuis une librairie) à chacun des *Embodiments*. Chacun des *Instantiation-agents* activé sélectionne ainsi un *Embodiment* et l'instancie par un composant réel, et ce au gré de ses préférences (cf. figure 2.8) et de l'utilité par rapport aux spécifications (en utilisant les équations extraites de la configuration).

Modification-agents. La fonction des *Modification-agents* est de modifier les configurations. Contrairement aux *Maker-agents* qui créent et complètent les configurations, les *Modification-agents* sont voués à diriger le processus global d'exploration vers de meilleurs sous-ensembles de l'espace de recherche. Un seul type de *Modification-agents* a été implémenté dans (Campbell, 2000, p. 83) : les *Fragmentation-agents*. Ces agents ont pour tâche de déterminer, dans une configuration donnée, quels sont les *Embodiments* qui réduisent le plus, de leur point de vue (cf. figure 2.8), l'utilité globale : ces parties sont alors retirées de la configuration.

Évidemment, d'autres types de *Modification-agents* sont envisageables : des agents remplaçant directement des *Embodiments* ou des composants, des agents effectuant des croisements entre différentes configurations (de la même manière que les croisements dans les algorithmes génétiques), etc.

Manager-agents. Les *Manager-agents* ont deux activités principales : l'intégration des préférences de l'utilisateur (en l'approximant par une fonction d'utilité) et le contrôle des populations de *Maker-agents* et de *Modification-agents*.

L'intégration des préférences de l'utilisateur au cours du processus s'effectue par un dialogue entre les *Manager-agents* et l'utilisateur lorsque ce dernier se voit présenter en fin de boucle les *pareto designs*. Lors de cette présentation, les *Manager-agents* demandent à l'utilisateur d'évaluer certaines des solutions en leur attribuant une note. A partir de ces données, les *Manager-agents* mettent à jour la fonction d'utilité associée aux préférences de l'utilisateur. C'est cette fonction d'utilité qui est utilisée pour séparer les *non-pareto designs* en *good design* et *poor design*.

La gestion des populations et de l'exécution des *Maker-agents* et des *Modification-agents* est basée sur l'analyse des contributions de ces agents dans la construction des mécanismes retenus en fin de cycle (les *pareto design* et les *good design*). Pour cela les *Manager-agents* construisent deux listes : une liste *TODO* contenant les tendances identifiées dans les meilleures configurations, et une liste *TABOO* contenant les tendances identifiées dans les pires configurations. Trois types de tendances ont été prises en compte dans la version du A-Design pour les systèmes électro-mécaniques (Campbell, 2000, p. 91-92) :

1. Les fragments de configuration (petits ensembles de *Embodiments* et de *FPs* connectés ensembles) : l'idée est d'identifier des "sous-fonctions" dans les solutions.
2. Les groupes d'agents ayant participé ensemble à la génération des configurations des meilleures et pires configurations : l'idée est d'identifier la pertinence des différents types de *Configuration-agents* pour résoudre le problème courant.
3. Les groupes de composants qui instancient les configurations : l'idée est d'identifier la pertinence des différents types d'*Instanciation-agents* pour résoudre le problème courant.

Ces listes permettent de modifier les populations d'agents (en maintenant ou rajoutant des agents qui sont dans la liste *TODO* par exemple) et leur fréquence d'exécution (les agents présents dans la liste *TABOO* seront moins souvent activés que ceux de la liste *TODO*). Les *Manager-agents* communiquent aussi ces deux listes aux *Maker-agents* et aux *Modification-agents* pour que ceux-ci fassent évoluer leurs préférences (c'est une sorte d'apprentissage par renforcement sur les préférences).

2.1.2.5 Analyse

Le *A-Design* se présente donc comme une approche automatique pour la conception de systèmes mécaniques. Globalement le A-design s'appuie sur un processus itératif de raffinage en se basant sur les solutions élaborées lors du cycle précédent et en gardant toujours un potentiel de création de nouveauté (ensemble des *good designs* pour maintenir la diversité, *maker-agents* dont l'activation est régie stochastiquement...). De plus, l'intégration de l'interaction avec le concepteur (présentation des *pareto designs*, discussion via les *manager-agents*, apprentissage...) introduit le double avantage d'un

contrôle extérieur transparent (l'utilisateur peut participer à la conception de manière simple) et de l'exploitation dynamique du potentiel créatif humain.

2.2 Optimisation

Aprés avoir sélectionné un ensemble d'alternatives potentiellement intéressantes, les dimensions des divers mécanismes sont optimisées. Pour ce faire, on recourt classiquement à des outils basés sur des méthodes semi-analytiques. Nous proposons, dans cette section, une présentation des méthodes d'optimisation de manière générale et plus particulièrement les métaheuristiques auxquelles les systèmes multi-agents adaptatifs ont souvent été comparés.

2.2.1 Considérations générales

De manière générale, devant tout problème d'optimisation il convient de le traiter en définissant les quatre points suivants :

1. Variables du problème. Quels sont les paramètres intéressants à faire varier ? Ces paramètres sont généralement étiquetés dans un vecteur $(x_1, x_2, ..., x_n)$ où chaque x_i correspond à une variable.
2. Espace de recherche. Dans quelles limites faire varier ces paramètres ? Il est généralement possible de limiter l'espace de variation de chaque variable. On borne ainsi chaque variable x_i par des valeurs $[x_{imin}, x_{imax}]$.
3. Objectifs. Quels sont les objectifs à atteindre ?
4. Méthode d'optimisation. Quelle méthode choisir ?

Les objectifs sont définis par une *fonction objectif* (*fonction de coût*, *fonction d'évaluation*) $f(x_1, x_2, ..., x_n)$ généralement à valeurs dans **R** ou dans **N**. Lorsque les objectifs sont multiples, une fonction $f_i(x_1, x_2, ..., x_n)$ est associée à chaque objectif i. On peut choisir de simplifier un problème multi-objectif en combinant ces objectifs à l'aide d'une simple somme pondérée ($f = \sum_i \alpha_i x_i$), d'un produit ($f = \prod_i x_i^{\alpha_i}$) ou de techniques plus raffinées telle que l'optimalité de Pareto (Eschenauer *et al.*, 2000).

Pour certains problèmes, des contraintes sont rajoutées et sont notées sous la forme :

$$g_j(x_1, x_2, ..., x_n) \leq 0$$

où j dénote la j^{ime} contrainte. Ces contraintes restreignent le domaine des solutions et le séparent en deux parties : les solutions réalisables (qui respectent les contraintes) et les non réalisables.

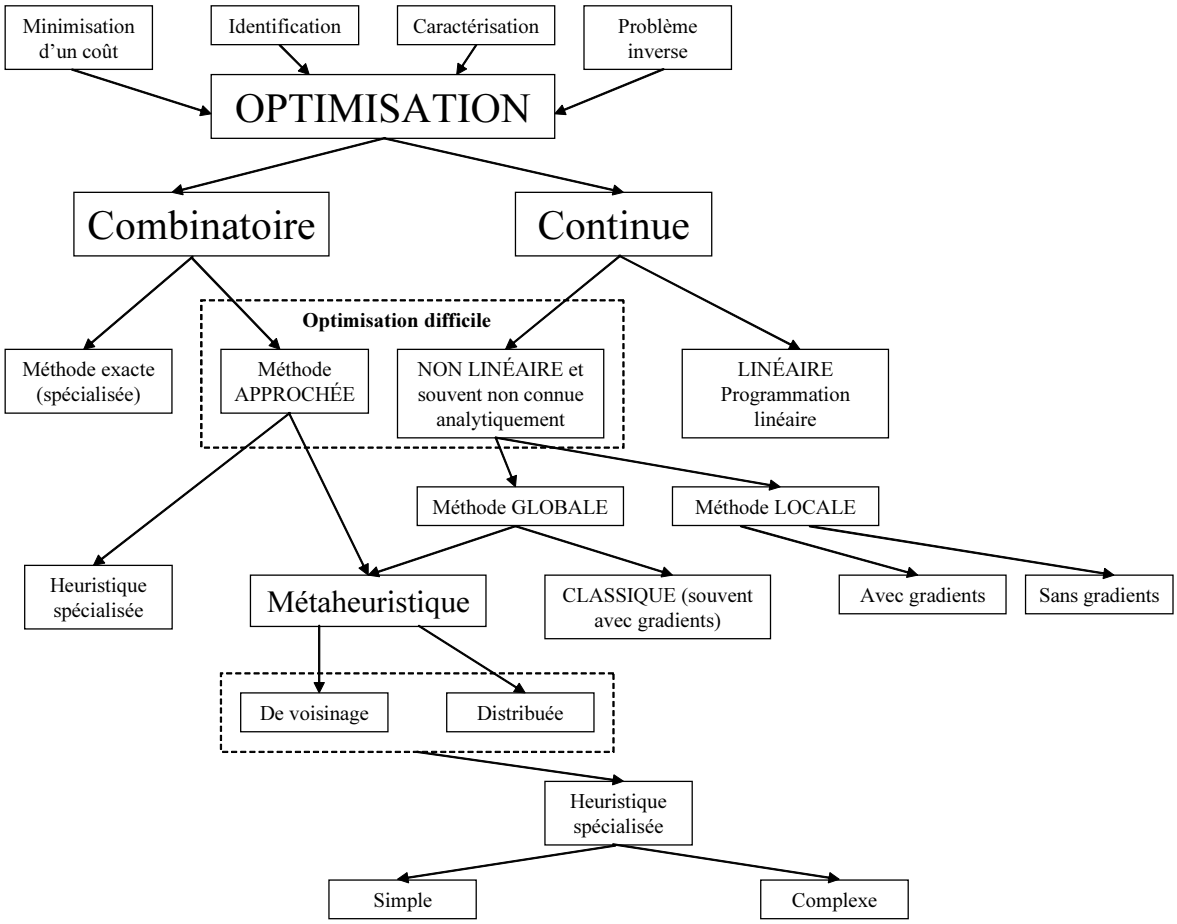

Figure 2.9 **—** Classification générale des méthodes d'optimisation mono-objectif

Dans le cas général de l'optimisation d'une fonction f, le problème se ramène à déterminer le minimum de f (ou le maximum, mais ce cas revient à chercher le minimum de $-f$).

De fait, le but de toute méthode d'optimisation consiste à fournir une stratégie pour parcourir l'espace de recherche afin de déterminer la "meilleure" valeur des variables dans l'espace de recherche du point de vue des objectifs. La recherche du minimum peut être faite de façon analytique ou de façon numérique mais, dans la grande majorité des cas réels, la fonction à optimiser n'est pas minimisable de façon analytique. On recourt donc à des méthodes numériques, et plus particulièrement à l'échantillonnage et à la descente de gradient. La première de ces méthodes consiste à évaluer la valeur de la fonction pour chacun des jeux de paramètres possibles. La seconde se base sur l'amélioration d'une solution approchée en examinant la fonction aux environs de la solution partielle courante (classiquement en suivant "la pente" de la fonction). Dans le domaine de la mécanique, on recourt plus généralement à des méthodes semi-analytiques (programmation linéaire séquentielle, programmation convexe) qui seront présentées ci-dessous.

Une autre alternative est l'utilisation de métaheuristiques. Comme le montre la figure 2.9 représentant une classification générale des méthodes d'optimisation mono-objectif (Collette et Siarry, 2002), les métaheuristiques ont pour objectif de traiter les problèmes dits d'*optimisation difficile*. Les systèmes multi-agents adaptatifs étant fortement liés aux systèmes complexes, une attention particulière a été portée sur l'étude des métaheuristiques - elles aussi proches des problématiques des systèmes complexes - et des travaux portant sur leur analyse.

2.2.2 Méthodes semi-analytiques

L'idée est de transformer un problème non linéaire contraint en un sous-problème linéaire non contraint, problème que l'on pourra ainsi résoudre avec des méthodes classiques efficaces. Le sous-problème fournissant une approximation de la solution recherchée, cette procédure est appliquée itérativement.

Pour exemple, la méthode de *programmation séquentielle linéaire* utilise ce principe. Partant d'un vecteur de variables arbitraires $X^0 = (x_1^0, x_2^0, ..., x_n^0)$, on recherche à chaque itération k une meilleure valeur pour $X^k = (x_1^k, x_2^k, ..., x_n^k)$ en effectuant une approximation linéaire de f (ou de la fonction objectif et des contraintes) au voisinage de X^{k-1}. Pour ce faire, la technique couramment utilisée est de linéariser f par expansion de premier ordre d'une série de Taylor autour de X^{k-1} :

$$f(X) \approx f(X^{k-1}) + \sum_i \frac{\partial f(X^{k-1})}{\partial x_i}(x_i - x_i^{k-1})$$

Cette approximation n'étant valable que pour le voisinage de X^{k-1}, on contraint donc l'espace de recherche autour de X^k :

$$x_i^k - \Delta x_i \leq x_i \leq x_i^k + \Delta x_i$$

Les Δx_i évoluant au cours du processus de résolution : ils peuvent être augmentés lorsque le processus converge convenablement et diminués lorsque des oscillations surviennent.

Il existe des variantes comme les méthodes de *programmation convexe* (algorithme de CONLIN ou la Méthode d'Asymptotes Mouvantes), utilisant elles aussi des linéarisations du premier ordre mais en effectuant des substitutions des variables qui permettent d'obtenir des approximations plus conservatrices (elles ont tendance à sous-estimer la taille du domaine et convergent donc généralement en passant par des solutions valables). Par exemple, dans l'algorithme de CONLIN (le plus ancien), certaines variables x_i sont remplacées par leur réciproque $1/x_i$ si $\partial f/\partial x_i < 0$.

Enfin, une dernière classe de méthodes utilisent des techniques d'approximation quadratique plus coûteuses en temps de calcul mais permettant l'obtention de sous-problèmes plus précis, ce qui amé-

liore la convergence. On obtient ainsi des approximations de la forme :

$$f(X) \approx f(X^{k-1}) + \sum_i \frac{\partial f(X^{k-1})}{\partial x_i}(x_i - x_i^{k-1}) + \frac{1}{2}(X - X^k)^T H (X - X^k)$$

où H est la matrice hessienne contenant les dérivées secondes de f :

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Une caractéristique intéressante de ces méthodes est que si H est positive, l'approximation "augmente" (s'incurve vers l'intérieur) quand on s'éloigne du point courant. Ceci implique que les limites Δx_i n'ont plus à être adaptées en cours de résolution (le but étant de minimiser, ces solutions plus éloignées auront tendance à être moins bonnes). Par contre, le calcul des dérivées d'ordre deux et les opérations mises en oeuvre sur H (inversion, factorisation) sont coûteuses en temps de calcul (notamment lorsque le nombre de variables croît).

Ces problèmes sont en partie résolus par les méthodes dites quasi-Newtoniennes. Ces méthodes sont aussi d'ordre deux, mais s'appuient sur l'approximation de la matrice hessienne : soit au fil des itérations (en partant de la matrice identité pour H puis en l'approximant au fil des itérations), soit en approximant directement l'inverse de H (éliminant ainsi les opérations d'inversion et de factorisation par la suite).

2.2.3 Métaheuristiques

Comme énoncé précédemment, les métaheuristiques ont pour objectif de résoudre les problèmes d'*optimisation difficile* (Dréo *et al.*, 2003). En fait, elles tentent d'apporter de nouvelles approches pour résoudre les deux écueils classiques de l'optimisation : les minima locaux et le coût en temps de calcul. Ces approches sont généralement inspirées de mécanismes présents dans la nature, comme le montre le tableau 2.3, et intègrent souvent une composante stochastique au niveau de la construction des nouvelles solutions.

2.2.3.1 Avantages, inconvénients et développements

Pour s'extraire des pièges que représentent les minima locaux, plusieurs mécanismes sont mis en oeuvre dans les métaheuristiques. Le premier d'entre eux, présent dans les méthodes de voisinage (recherche tabou, recuit simulé), est de permettre des dégradations de la solution (on "remonte les parois du puits représentant le minimum local"). Dans les métaheuristiques distribuées (dans lesquelles

Méthode	Inspiration
Recuit simulé	Physique
Recherche tabou	Mémoire humaine
Essaims particulaires	Comportements sociaux
Algorithmes évolutionnaires	Évolution darwinienne
Colonie de fourmis	Éthologie

Tableau 2.3 **—** Métaheuristiques et domaines d'inspiration

on gère des populations de solutions comme dans les algorithmes génétiques), des perturbations sont introduites pour extraire les individus bloqués dans les "puits les moins profonds" : c'est le rôle de l'opérateur de mutation dans les algorithmes génétiques, par exemple. La nature distribuée de ces méthodes est elle-même un autre facteur qui vise à augmenter les chances de trouver l'optimum global (les populations couvrent une grande partie de l'espace des solutions). Certains mécanismes, tel que la *redistribution* ou *sharing* dans les algorithmes génétiques, visent à renforcer cette propriété. Enfin, les principes mêmes qui sont au coeur des métaheuristiques sont sans doute une source d'efficacité : en s'inspirant des mécanismes présents dans les systèmes complexes naturels, ces méthodes cherchent à exploiter ce qui rend ces systèmes si efficaces quand bien même leur compréhension reste limitée.

Malgré tout, aucune des métaheuristique n'est à même de garantir sérieusement que l'optimum sera atteint. Toutes les études théoriques de ces méthodes n'ont fourni, jusqu'à présent, que de maigres résultats sur les propriétés de convergence si ce n'est d'affirmer que pour être (presque) sûr de trouver l'optimum il convient d'examiner un nombre de solutions supérieur au nombre total de solutions du problème, i.e. utiliser une méthode exacte de type balayage.

Du point de vue des coûts en temps de calcul, l'apport des métaheuristiques est plus contrasté. Le premier paradoxe se situe au niveau de l'exploitation de populations de solutions : la couverture quasi-totale du paysage des solutions couplée à l'évolution parallèle des individus permet intuitivement de réduire le nombre d'itérations nécessaires pour converger (les chances qu'un ou plusieurs individus soient rapidement aux alentours de l'optimum sont plus importantes que lorsque l'on ne fait évoluer qu'une seule solution très dépendante de sa position initiale), mais le coût de chaque itération est d'autant plus important que cette population est importante. Ce coût augmente de même lorsque les mécanismes d'évolution régissant les mouvements des individus se complexifient alors que ces mécanismes ont pour but d'optimiser les déplacements locaux et donc de diminuer le nombre d'itérations nécessaires pour converger (la première loi de la thermodynamique nous rattrape ...). Ce dernier point est aussi valable pour les méthodes de voisinage : le développement d'algorithmes permettant le calcul des voisinages efficaces (souvent liés à la structure de l'espace des solutions) permet a priori

de réduire le nombre d'itérations mais augmente généralement leur coût. Une partie de ces coûts est tout de même résorbée en amont du calcul : le point crucial de l'application des métaheuristiques (en dehors du choix de la métaheuristique à appliquer pour un problème donné) est la détermination des valeurs des paramètres (taux de mutation et croisements, taille des populations manipulées, taille de la mémoire, etc.). Ces réglages paramétriques sont en fait souvent effectués de manière empirique à l'aide de jeux de fonctions analytiques dont les optima sont connus, à l'aide d'analyses statistiques sur des espaces spécifiques ou, plus simplement, en se basant sur l'expérience de l'utilisateur, par analogie avec d'autres problèmes et par essais-erreur.

2.2.3.2 Recherche tabou

La particularité principale de cette méthode est de s'inspirer de la mémoire humaine : à chaque itération, la solution courante s évolue en prenant en compte les m mouvements précédents pour éviter *les erreurs du passé*. La référence la plus importante au sujet de la méthode tabou est sans doute l'ouvrage de F. Glover (Glover et Laguna, 1997) auquel je conseille de se référer pour toute information complémentaire. Le principe de l'algorithme est très simple comme le montre la figure 2.10. L'algorithme démarre d'une configuration arbitraire. L'évolution à chaque itération de cette configuration se déroule en deux étapes :

Évaluation des N voisins. L'ensemble des voisins de s est l'ensemble noté $V(s)$ des configurations atteignables en un seul mouvement élémentaire à partir de s (dans les cas où cet ensemble est trop important, une technique quelconque de réduction est mise en oeuvre - par exemple en extrayant un sous-ensemble de manière aléatoire). Est exclue de $V(s)$ toute configuration dérivée de s par l'application d'un mouvement tabou.

Sélection du meilleur voisin. Chaque voisin de $V(s)$ est évalué du point de vue de la fonction objectif f afin de déterminer quel est le meilleur (que l'on notera t). Cette configuration devient la configuration courante pour l'itération suivante.

Suite à cette phase d'évolution, le mouvement inverse de celui ayant permis la transition de s vers t est rajouté à la liste des tabous (cette liste ayant une taille fixée et fonctionnant en FIFO). Cette liste tabou est une mémoire à court terme des solutions visitées récemment afin de ne pas les revisiter. Des mécanismes permettant d'introduire des mémoires à plus long terme existent : l'*intensification* (concentration de l'exploration sur une "zone" identifiée comme prometteuse) et la *diversification* (réorientation périodique de la recherche vers des "zones" encore trop peu explorées).

Ces deux processus se retrouvent par ailleurs dans l'algorithme ECTS (*Enhanced Continuous Tabu Search*) qui propose une approche "fidèle à la méthode tabou combinatoire" pour son application sur des problèmes d'optimisation continue (Dréo *et al.*, 2003, p. 179-181) (Chelouah et Siarry, 2000).

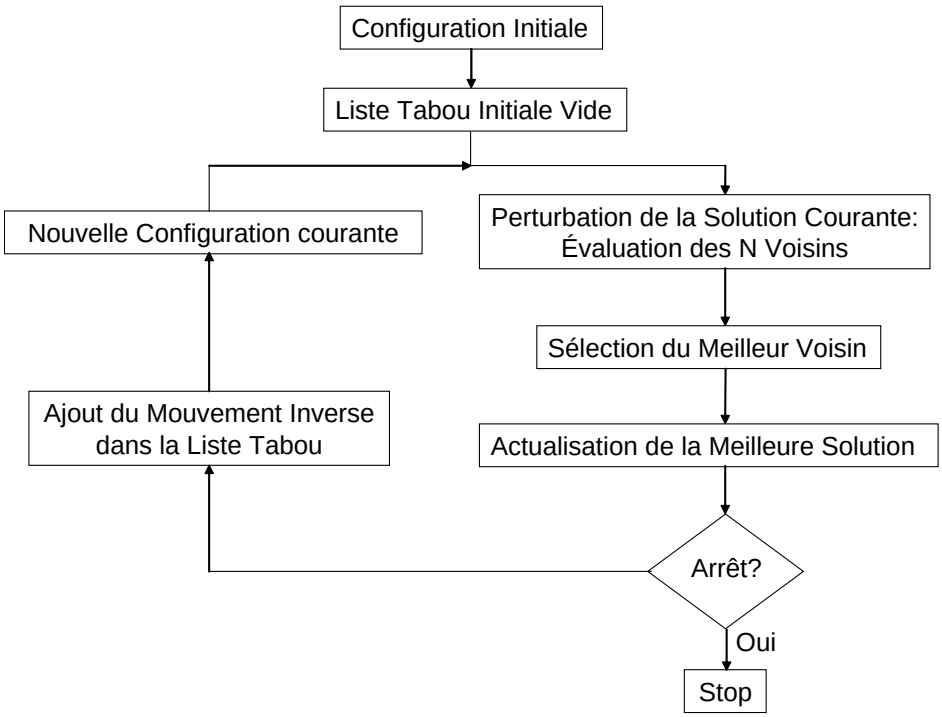

Figure 2.10 **—** Algorithme tabou simple

Dans cet algorithme, dont la structure est décrite par la figure 2.11, la phase de *diversification* consiste à déterminer les "zones prometteuses" de l'espace des solutions. Pour construire ces zones, on attribue en premier lieu le statut de "prometteuse" à des solutions jugées nettement meilleures (relativement à un seuil déterminé par ECTS) que ses N voisins évalués pendant la recherche tabou. La liste des "zones prometteuses" est alors construite comme la liste des "boules" centrées autour de ces solutions prometteuses. Durant la phase d' *intensification*, une nouvelle recherche tabou est lancée à l'intérieur de la meilleure "zone prometteuse".

De manière générale, les deux points cruciaux lors de la mise en oeuvre d'une recherche tabou sont l'évaluation du voisinage de la solution courante et les mécanismes gérant les interdits. Ces deux axes conditionnent grandement la "convergence" [4] et la qualité des solutions trouvées. L'évaluation du voisinage est le moteur de la recherche et sa performance dépend d'une part de la qualité des types de mouvement pour transformer la solution courante (a priori uniquement justifiable par le bon sens et de manière empirique) qui conditionnent "l'intelligence" de la recherche, et d'autre part l'efficacité de l'évaluation des solutions atteignables (il faut essayer de choisir le meilleur voisin) qui peut être étudiée et accélérée par des considérations algébriques. La gestion des interdits (la liste tabou) est basiquement une mémoire à court terme dont le but est d'interdire d'effectuer certains mouvements

[4]En fait il n'y a pas réellement convergence dans les recherches tabou puisque la terminaison est souvent arbitraire (la recherche s'effectue un nombre donné d'itérations ou selon des critères *ad-hoc*, la meilleure solution rencontrée étant retenue) et la confluence est très rarement garantie en pratique

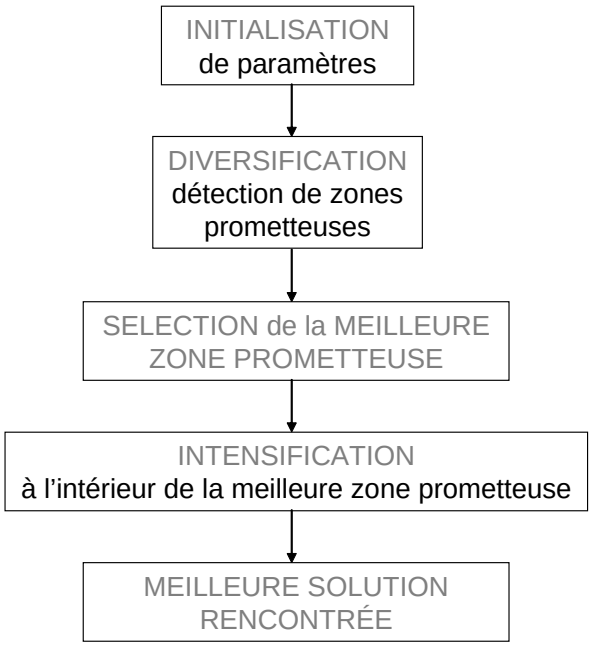

Figure 2.11 **—** Structure générale de l'algorithme tabou continu ECTS

ou de revisiter d'anciennes solutions. Le paramètre critique est ici la taille de la mémoire : si celle-ci est trop petite, le processus de recherche aura tendance à rester bloqué sur un petit ensemble de solutions (piégé autour d'un minimum local), mais si elle est trop grande, le processus explorera un grand nombre de solutions mais ne trouvera probablement pas un bon optimum car il n'explorera pas correctement les alentours des meilleures solutions (il trouvera "des vallées" mais pas forcément leur "fond"). Des études statistiques ont été menées pour juger de l'influence de la taille de la mémoire et de nombreux mécanismes ont été développés afin de pallier ces problèmes (mémoires à plus long terme, interdictions à durées aléatoires, critères d'aspiration ...).

2.2.3.3 Recuit simulé

Cette méthode s'inspire de la technique expérimentale de *recuit* qui s'oppose à la technique de la *trempe* utilisée par les métallurgistes pour obtenir des états solides "bien ordonnés" : le matériau est porté à haute température puis celle-ci est abaissée lentement par paliers. L'idée a été développée au début des années 80 par des ingénieurs d'IBM travaillant sur les verres de spins[5] (Kirkpatrick *et al.*, 1983). Tout comme la recherche tabou, le recuit simulé est une métaheuristique non distribuée et s'appuie donc sur l'évolution d'une seule solution par exploitation du voisinage. La métaphore vis-à-vis du processus physique est simple : la fonction objectif est assimilée à l'énergie libre d'un système physique, les paramètres du problème correspondent aux "coordonnées" des particules et les

[5]matériaux magnétiques désordonnés

minima représentent les états de basse énergie. La pierre angulaire du principe étant le contrôle d'une température virtuelle T dans l'algorithme : la valeur de cette température conditionne le nombre d'états accessibles (qui sont d'autant moins nombreux que cette température baisse). L'algorithme décrit dans la figure 2.12 explicite le processus global.

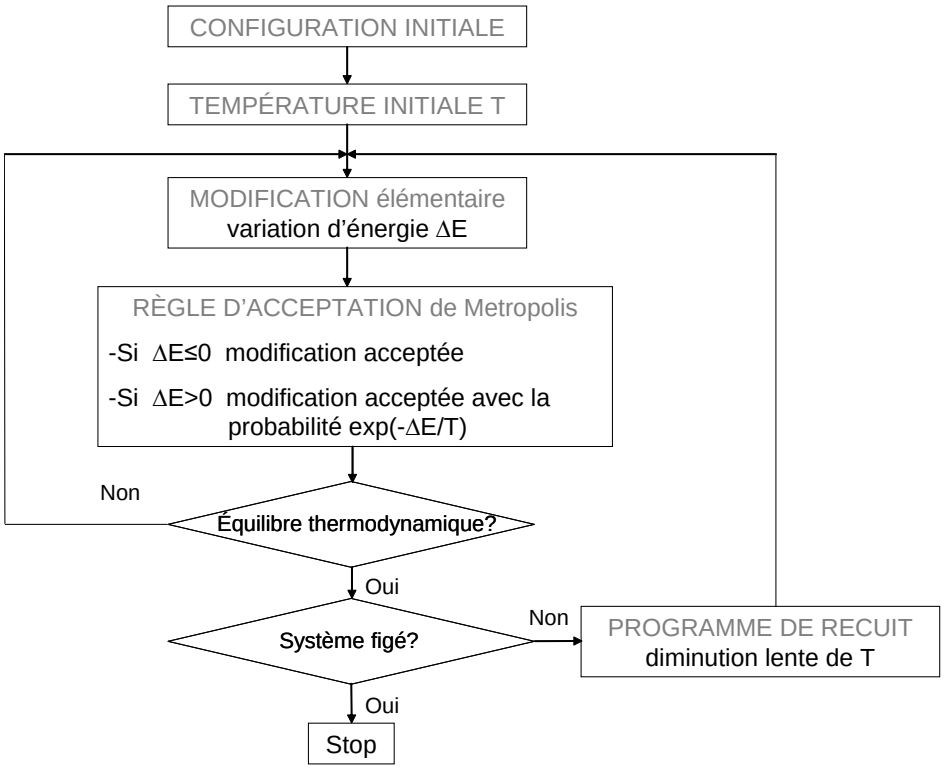

Figure 2.12 — Algorithme du recuit simulé

Pour chaque palier de température, la solution cherche son "équilibre thermodynamique" en exploitant l'algorithme de Metropolis : à partir d'une configuration donnée, on effectue une modification élémentaire ce qui entraîne une variation de la fonction objectif (assimilée à l'énergie E du système). Le critère d'acceptation de cette nouvelle configuration est alors simple : si la variation de la fonction (donc δE) est négative on l'accepte (la configuration est meilleure), sinon (si la configuration est donc moins bonne du point de vue de f) on l'accepte tout de même avec la probabilité $e^{\frac{-\delta E}{T}}$. Cette règle induit que lorsque la température T est élevée, $e^{\frac{-\delta E}{T}}$ est proche de 1 et donc la configuration évolue rapidement et de manière permissive (les configurations dégradant la valeur de f sont facilement acceptées). En abaissant T au cours du processus, $e^{\frac{-\delta E}{T}}$ se rapproche de 0 et la recherche se focalise donc de plus en plus sur les configurations minimisant f. Globalement, le processus peut être vu comme une évolution "continue" ou progressive d'une phase de *diversification* à celle d'*intensification*.

L'adaptation du recuit simulé aux problèmes à variables continues nécessite essentiellement des modifications au niveau de la dynamique des particules : la "discrétisation" de l'espace de recherche

doit maintenir l'efficacité du processus d'optimisation lors de la descente de température (Dréo *et al.*, 2003)(p. 177-179). Une solution simple consiste à associer à chaque variable x_i un pas de discrétisation $STEP_i$ représentant la variation maximale que peut subir x_i pour une modification élémentaire, $STEP_i$ étant modifié périodiquement pour maintenir constante l'efficacité de l'optimisation. Les lois d'ajustement des $STEP_i$, le mode d'évaluation de l'efficacité et les fréquences d'ajustement restent toutefois des paramètres généralement définis empiriquement en s'appuyant sur des études statistiques. L'algorithme 1 illustre la forme générale d'un recuit simulé à variables continues (Dréo *et al.*, 2003, p. 179).

Algorithme 1 : Recuit simulé en variables continues

- $STEP_i$ initial : $\frac{1}{4}$ du domaine de variation de *xi* ;
- $STEP_i$ modifié à l'issue de chaque palier de température ;
- On exploite le taux d'acceptation A_i des mouvements tentés, au cours du palier, pour x_i :
  - si $A_i > 20\%$, $STEP_i$ est doublé,
  - si $A_i < 5\%$, $STEP_i$ est divisé par 2,
- *Si* le domaine de variation de x_i est "peu étendu" :

$$x_i' = x_i \pm y \cdot STEp_i$$

  où y désigne un nombre aléatoire, tiré dans $[0,1]$. *Sinon* (cas de plusieurs décades), la perturbation est opérée selon une loi logarithmique ;
- Les n variables du problème sont modifiées par groupes de p, formés aléatoirement : typiquement, $p \cong \frac{n}{3}$. Les fréquences de mouvement des différentes variables sont égalisées.
- Plusieurs critères complémentaires sont utilisés pour l'arrêt automatique du programme.

Le recuit simulé a suscité beaucoup d'intérêt dès son apparition du fait de la nouveauté du processus d'optimisation. De nombreuses études théoriques ont été menées permettant notamment d'établir que, sous certaines conditions, le recuit simulé converge en probabilité vers un optimum global. Par exemple, les travaux de Aarts (Aarts et van Laarhoven, 1985) s'appuient sur l'hypothèse de la décroissance de la température par paliers[6] (permettant une modélisation du processus en utilisant un ensemble de chaînes de Markov homogènes). Les conditions de convergence du recuit sont alors que les changements soient tous réversibles et que le graphe d'état du système soit connexe (tous les états accessibles). Outre le fait que cette étude montre l'efficacité de la décroissance par paliers, Aarts établit que l'on peut obtenir une solution "de qualité" (optimale à quelques pourcents près) en un temps polynomial pour certains problèmes NP-difficiles.

Des études de Hajek (Hajek et Sasaki, 1989), en se plaçant dans le cadre plus général des chaînes

[6] De nombreux autres travaux se sont intéressés à différentes lois d'évolution pour T

de Markov inhomogènes, résulte une condition de convergence différente portant sur la loi d'évolution de T au cours du temps (notée $T(t)$) : la probabilité que le recuit converge vers l'optimum global, lorsque le temps tend à l'infini, est l'unité si $T(t)$ ne décroît pas plus vite que $\frac{C}{log(t)}$ où C est une constante dépendante de la profondeur des "puits d'énergie" du problème. Ces "puits d'énergie" correspondent aux minima du "paysage d'énergie" (chaque point de ce paysage représente une solution x, l'altitude associée étant l'énergie $f(x)$). Ces "paysages" ont eux aussi été largement étudiés dans le cadre du recuit simulé et des liens ont été établis entre certaines propriétés topologiques de ces espaces de configuration et l'efficacité de l'algorithme. Pour plus de détails sur ces éléments théoriques relatifs à la convergence du recuit simulé, je renvoie le lecteur à l'ouvrage didactique de Saït (Saït et Youssef, 1999)(Chap. 2).

2.2.3.4 Algorithmes évolutionnaires

Les algorithmes évolutionnaires s'inspirent des théories Darwiniennes de l'évolution des espèces : une population d'individus soumis à une pression environnementale s'adapte, au fil des générations, par croisement génétique et sélection naturelle. Seuls les individus les mieux adaptés survivent et ont des chances de se reproduire dotant ainsi leurs descendants des caractéristiques qui leur ont permis de survivre. Depuis les années 50, nombre de chercheurs ont essayé d'utiliser cette mécanique évolutionnaire à des fins d'optimisation numérique mais ce n'est qu'à la fin des années 80, avec le livre de D.E. Goldberg (Goldberg, 1989), que la discipline se développera réellement avec comme figure de proue les *algorithmes génétiques*.

Le schéma 2.13 donne un aperçu de la structure d'un algorithme évolutionnaire : partant d'une population aléatoire de solutions, chaque individu est évalué à l'aide d'une fonction d'évaluation (*fitness*) et une partie d'entre-eux (les meilleurs) génèrent de nouveaux individus à l'aide d'opérations de croisement et de mutation tandis que les moins performants sont éliminés. Cette nouvelle population est ensuite prise pour base d'un nouveau cycle d'évolution.

La définition des opérateurs de sélection et de variation conditionne fortement la qualité de la solution trouvée ainsi que la convergence (tant la terminaison que le temps de calcul nécessaire) des algorithmes évolutionnaires. De manière générale, les individus ayant obtenu les meilleures performances doivent se reproduire plus souvent que les autres. Toutefois, si l'opérateur de sélection d'un algorithme donné permet au meilleur individu de se reproduire trop rapidement, il y a un grand risque que sa descendance envahisse rapidement toute la population et, par conséquent, que l'algorithme converge prématurément vers une solution non optimale (si le super individu est non optimal). Cette observation a conduit à la définition de la pression de sélection : le temps de domination τ^* (*takeover time*) est défini comme le nombre de générations nécessaires au meilleur individu pour remplir l'en-

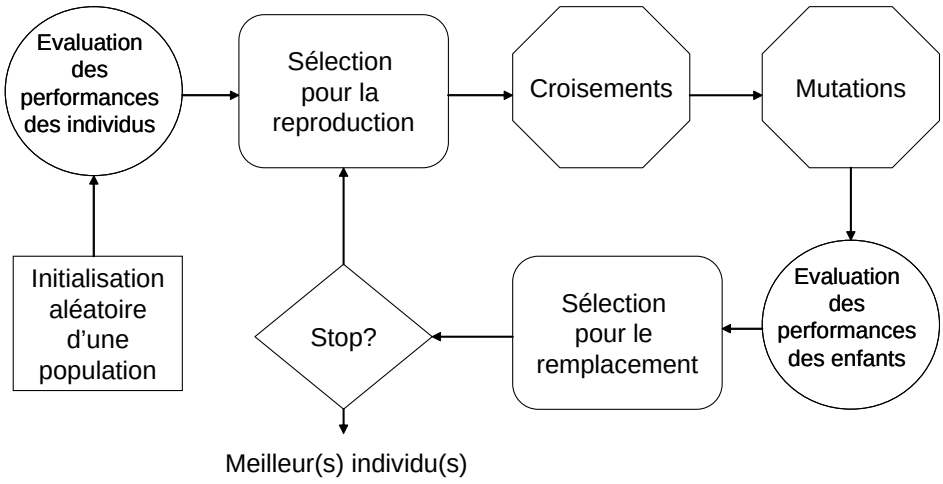

Figure 2.13 **—** Principe des algorithmes évolutionnaires

semble de la population à l'aide des seuls opérateurs de sélection. Le réglage de τ^* influe grandement sur la convergence de l'algorithme : si τ^* est trop faible, l'algorithme convergera probablement rapidement mais vers une solution de piètre qualité alors qu'un τ^* trop élevé peut entraîner un temps de calcul très important, voire même ne pas permettre à l'algorithme de converger.

Un autre phénomène doit être pris en compte dans le réglage des opérateurs de sélection : la dérive génétique. En effet, à partir d'une population d'individus différents mais dont les performances sont similaires, un algorithme évolutionnaires sans opérateur de variation converge vers une population homogène. Lorsque cette dérive génétique est trop importante, la population perd en diversité et l'algorithme tend à converger prématurément. Un moyen vérifié d'allonger le temps de convergence vers un état homogène dû à la dérive génétique est d'éviter de travailler avec des populations comprenant un nombre important d'individus. D'autre part, il est à noter que l'effet de la dérive génétique est prédominant si la pression de sélection est trop faible. Le dernier paramètre d'importance pour la sélection est évidemment la définition de la fonction de performance (*fitness*).

De nombreuses heuristiques évolutionnaires différentes ont été développées comme, par exemple, celles dans lesquelles on ne choisit pas les opérateurs génétiques appliqués de manière aléatoire mais en appliquant des règles de transition fixes entre générations. Il existe plusieurs variantes de ce principe :

- Le "space partition algorithm" (SPA), qui réalise une partition de l'espace en raffinant les régions où la fonction à optimiser est la meilleure et en fusionnant les autres. Les individus de la population sont alors des blocs de tailles changeantes dont l'évaluation est néanmoins moyennée aléatoirement en leur sein.
- Le "macro-evolutionary algorithm" (MGA), inspiré par la dynamique des espèces, qui est une heuristique dont le principe est de relier les individus par des poids mesurant leurs influences

mutuelles. Il y a, au cours des générations, extinction ou bien application d'une sorte de *crossover* continu entre individus.

– L' "évolution différentielle" (DE), où les individus donnent naissance à de nouveaux points de l'espace de recherche grâce à une sorte de tétra-crossover.
– Le "self organizing migration algorithm" (SOMA), dont l'idée principale est de déterminer des groupes à l'intérieur desquels chaque individu réalise des "sauts" vers le meilleur du groupe et en s'arrêtant sur la meilleure position rencontrée. Les groupes sont modifiées à chaque génération.

2.2.3.5 Essaims particulaires

L'optimisation par essaim de particules (OEP) est une technique d'optimisation parallèle développée par Kennedy et Eberhart (Kennedy et Eberhart, 1995), en s'inspirant du comportement social des individus qui ont tendance à imiter les comportements réussis qu'ils observent dans leur entourage, tout en y apportant leurs variations personnelles. A la différence des algorithmes évolutionnaires, qui miment les mécanismes génétiques de l'évolution, l'OEP s'inspire plutôt de la formation des cultures. Un ouvrage complet sur les racines sociales de l'OEP et les techniques mathématiques mises en oeuvre est le livre *swarm intelligence* (Kennedy *et al.*, 2001).

L'algorithme OEP classique peut être décrit sous forme vectorielle de la façon suivante :

$$\begin{aligned}\vec{v}_{t+1} &= \vec{a}\otimes\vec{v}_t+\vec{b}_1\otimes\vec{r}_1\otimes(\vec{p}_1-\vec{x}_t)+\vec{b}_2\otimes\vec{r}_2\otimes(\vec{p}_2-\vec{x}_t)\\ \vec{x}_{t+1} &= \vec{c}\otimes\vec{x}_t+\vec{d}\otimes\vec{v}_{t+1}\end{aligned}$$

Le symbole $\otimes$ signifie ici la multiplication des vecteurs élément par élément. A l'itération $t+1$, la vitesse $\vec{x}_{t+1}$ d'une particule est modifiée à partir de sa valeur courante, affectée d'un coefficient d'inertie $\vec{a}$, et de deux forces qui attirent la particule vers sa propre meilleure position passée $\vec{p}_1$ et la meilleure position de tout l'essaim $\vec{p}_2$. L'intensité de l'attraction est donnée par les coefficients $\vec{b}_1$ et $\vec{b}_2$. La position de la particule $\vec{x}_{t+1}$ est modifiée à partir de la position courante et de la nouvelle vitesse calculée $\vec{v}_{t+1}$, affectées des coefficients $\vec{c}$ et respectivement $\vec{d}$. L'expérience montre qu'une bonne exploration du domaine de recherche est obtenue en introduisant les nombres aléatoires $\vec{r}_1$ et $\vec{r}_2$, en général avec une répartition uniforme entre 0 et 1.

2.3 Discussion

Les travaux dans le domaine de la conception automatique de mécanismes sont donc relativement limités. Toutefois les deux méthodes de synthèse de mécanismes exposées ici suivent deux axes

bien distincts dans leurs centres d'intérêts. La première propose une méthode systématique, suivant une approche relativement formelle, qui présente l'avantage de l'exhaustivité (ou presque) des mécanismes évaluées au détriment de l'efficacité de traitement (le nombre de mécanismes finalement testés que ce soit partiellement ou totalement reste important) et de la flexibilité d'utilisation vis-à-vis des utilisateurs d'un tel processus. Le *A-Design* est, a contrario, bien plus centré sur l'utilisateur et s'appuie largement sur son jugement afin de parfaire les mécanismes (et donc guider sa recherche), en essayant de maintenir la génération de nouveautés grâce au paradigme évolutionnaire utilisé. Malgré tout, ni l'une ni l'autre ne parvient à définir de réelle heuristique pour le parcours de l'espace gigantesque et complexe des mécanismes possibles. Dans les deux cas, un processus stochastique est mis en oeuvre pour pallier ce problème et éviter un parcours aveugle, au prix d'une consommation en terme de temps de calcul qui s'avère conséquente (comme pour tout algorithme évolutionnaire) et sans garantie de ne pas échouer dans un minimum local (ne pas trouver le meilleur mécanisme possible).

Les techniques relatives à l'optimisation des mécanismes ainsi générés, sont historiquement mises en oeuvre dans des outils spécialisés qui s'appuient sur des algorithmes d'optimisation semi-analytiques. Rares sont les utilisations industrielles de méthodes alternatives comme les métaheuristiques. Cette phase d'optimisation peut donc s'avérer relativement coûteuse en temps d'autant plus que le nombre de mécanismes devant être testés est important. C'est l'une des raisons pour lesquelles les deux solutions que nous avons exposées dans cette partie incluent elle-mêmes un processus d'optimisation dans leur résolution.

L'application du paradigme des systèmes multi-agents adaptatifs à ce problème peut, selon nous, constituer une alternative très intéressante. D'une part cette nouvelle approche permet de fournir un processus flexible et interactif avec l'utilisateur (l'adaptation étant la pierre angulaire de ces systèmes, cette interactivité est "naturelle"), mais elle a aussi l'ambition de proposer une heuristique locale pour le parcours de l'espace de recherche : la coopération.

Deuxième partie

Mechanical Synthesis Solver : conception automatique de mécanismes

3 Architecture

POUR résoudre le problème de conception de systèmes mécaniques, nous proposons une approche basée sur la théorie des systèmes multi-agents adaptatifs. Le principe est, pour des objectifs donnés, de découvrir un mécanisme adéquat à l'aide d'un processus d'auto-organisation des composants mécaniques. Le *Mechanical Synthesis Solver* est un prototype développé dans le cadre du projet européen SYNAMEC[1] afin de valider cette application des *AMAS* au domaine de la conception de mécanismes. Ce chapitre est dédié à la présentation du *Mechanical Synthesis Solver* au travers des étapes de sa conception.

La méthode ADELFE fournissant un support au développement des systèmes multi-agents adaptatifs, nous présenterons, dans ce chapitre, la conception du *Mechanical Synthesis Solver* en nous appuyant sur le processus de cette méthode (Picard et Gleizes, 2004)(Picard, 2004). La figure 3.1 illustre les quatre premières phases du processus d'ADELFE. Ce processus -basé sur le RUP (*Rational Unified Process*)- a été enrichi d'un certain nombre d'étapes spécifiques à la conception des *AMAS* (en gras sur la figure 3.1).

Nous présenterons, en premier lieu, la conception générale du système en appliquant les trois premières étapes de la méthode. Nous détaillerons ensuite la conception et l'implémentation des agents en se basant sur le modèle d'agent coopératif proposé par *ADELFE*.

3.1 Le système

Dans cette partie, sont décrites les trois premières phases de conception de la méthode ADELFE appliquée à notre problématique : les besoins préliminaires, les besoins finals et l'analyse. A l'issue de ces trois phases, le domaine de l'application est bien défini, l'adéquation des AMAS vérifiée et les entités qui doivent être des agents identifiées.

[1] Synthesis Tool for Aeronautical Mechanism Design. Proposal number : GRD1-2001-40202.

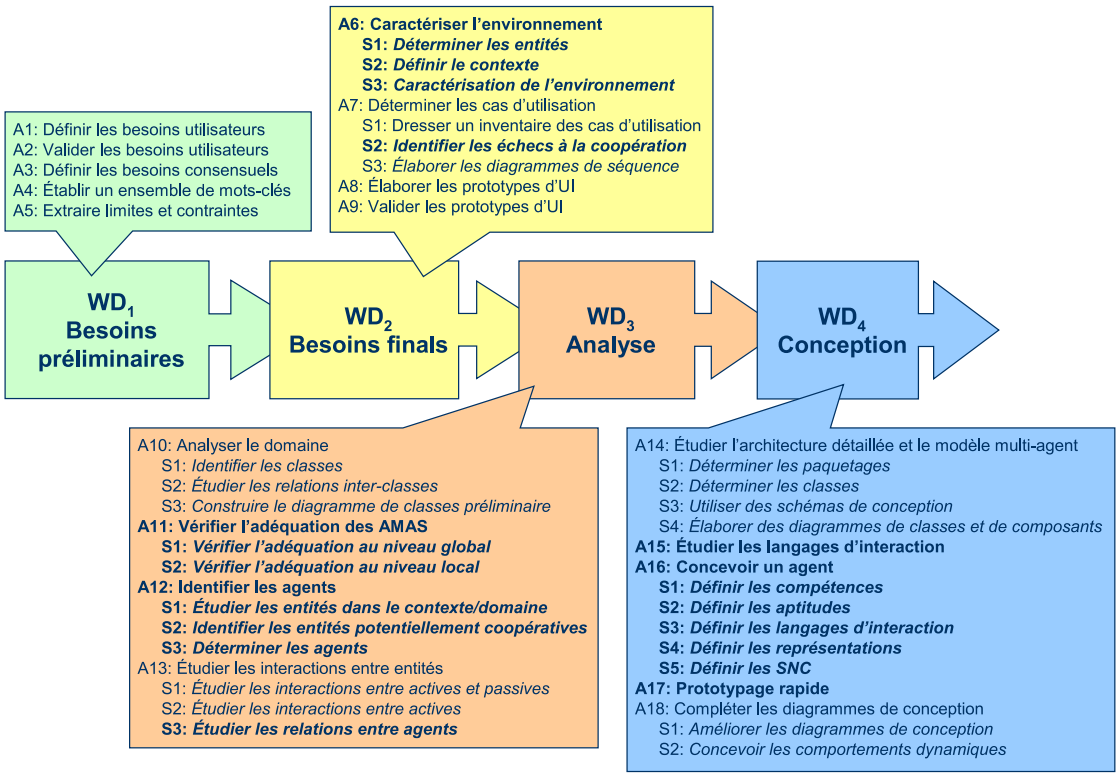

Figure 3.1 — Processus d'ADELFE

3.1.1 Besoins préliminaires et finals

Durant ces deux premières définitions de travail (*WorkDefinition* ou *WD*), l'objectif est de définir et de valider un cahier des charges précis de l'application entre les clients, les utilisateurs et les concepteurs à propos des fonctionnalités requises. Dans le cadre du projet SYNAMEC, et donc de notre travail, ces différents acteurs étaient réunis :

- Les *utilisateurs* étaient représentés par des ingénieurs d'entreprises du domaine aéronautique : ALENIA (de Turin en Italie), SNECMA (de Paris) et SABCA (de Bruxelles en Belgique).
- Le principal *client*, avec qui nous avons étroitement travaillé, étant une entreprise belge de développement de logiciels d'optimisation pour les systèmes mécaniques : SAMTECH. De plus, deux universités (l'INTEC en Argentine et l'université de CRANFIELD en Angleterre) ont participé en tant que conseillers dans le domaine de la conception de mécanismes.
- Les *concepteurs* et *développeurs* de ce système étant le laboratoire de l'IRIT.

Mises à part les activités classiques du RUP liées à l'expression des besoins des utilisateurs, des contraintes et limites du système, des descriptions des cas d'utilisation, etc., ADELFE met en exergue la caractérisation de l'environnement. En effet, le but étant de concevoir des systèmes adaptatifs, la qualification des caractéristiques de l'environnement dans lequel il sera plongé lors de sa mise en oeuvre est de première importance pour l'étude de sa dynamique.

3.1.1.1 Les besoins des utilisateurs

Le système devant être réalisé est un "solveur" : à partir de la définition d'objectifs (typiquement une ou des trajectoires à suivre), le MSS doit générer des mécanismes permettant de les satisfaire. Une ou des solutions doivent être proposées à partir des éléments spécifiés initialement par l'utilisateur. Ces éléments contiennent nécessairement :

- Les définitions des trajectoires caractérisant les objectifs à remplir : ces trajectoires peuvent être définies par le biais d'équations paramétriques, de points de passage ou de la compositions d'équations et de points de passage.
- Les composants dont l'utilisateur pense, a priori, qu'ils sont nécessaires pour la résolution du problème posé. Ces éléments peuvent être définis comme **devant** ou comme **pouvant** faire partie du système (dans ce dernier cas, des solutions dans lesquelles ces composants ont disparu sont envisageables).
- Les définitions des actuateurs du système : ces éléments correspondent à des variations angulaires prédéfinies sur certains charnières ou à des mouvements de certaines glissières.

D'autres éléments, tels que des limitations relativement au nombre maximal de composants ou aux types de composants étant disponibles, pourront être rajoutés optionnellement. De même, la définition d'une enveloppe dans laquelle le mécanisme doit être contenu, ou la disposition d'obstacles autour des trajectoires peuvent être requis pour certains problèmes.

De plus, l'utilisateur devra également pouvoir observer et modifier dynamiquement les différents éléments du problème et de la solution courante pendant la résolution : changements des objectifs et contraintes, ajout ou retrait arbitraire de composants, modifications des dimensions.

Le prototype se restreindra au seul domaine des *linkages* (cf. 2.1.1.1.4) : les solutions seront par conséquent des mécanismes plans contenant seulement des charnières et des glissières comme joints. L'objectif du MSS n'est pas de simuler l'activité d'un système mécanique mais de le concevoir et de l'optimiser. Or, des calculs de cinématique seront nécessaires pour évaluer les performances des mécanismes : ces calculs seront effectués par le solveur MECANO de la société SAMTECH[2].

3.1.1.2 Caractérisation de l'environnement

Nous pouvons identifier une entité active (l'utilisateur) et une entité passive (MECANO) qui seront en interaction avec le système. Durant la résolution, l'utilisateur peut à tout moment modifier les paramètres du problème ainsi que créer, retirer ou modifier des contraintes. Via l'interface graphique, il observe le mécanisme courant (la solution courante donc) et peut modifier les dimensions des élé-

[2] http ://www.samtech.be

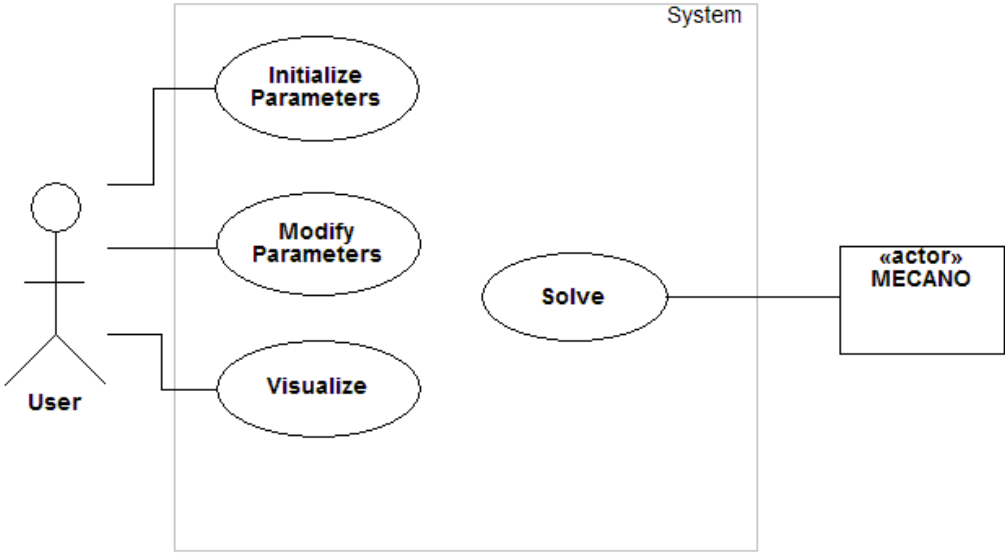

Figure 3.2 — Cas d'utilisation

ments le constituant. Pour sa part, MECANO fournit au système un outil de calcul de cinématique pour le traitement des mouvements (voir précisions en 3.1.4).

L'environnement du MSS peut se définir comme étant :

- **Dynamique** : Les contraintes et paramètres relatifs au problème peuvent évoluer n'importe quand.
- **Non accessible** : Si le calcul d'une cinématique n'est pas réalisable par MECANO, on obtient une erreur interprétable, en toute généralité, seulement par un spécialiste ou par l'utilisateur.
- **Non déterministe** : Le système n'a pas de représentation de corrélation entre l'évolution des solutions élaborées et les interventions de l'utilisateur. De plus, l'espace des solutions est fortement discontinu (des modifications topologiques dans un mécanisme peuvent parfois entraîner des changements drastiques de son comportement).
- **Continu** : Il n'y a, a priori, pas de limite au nombre de combinaisons de contraintes et objectifs que l'utilisateur peut générer.

A ce niveau d'analyse, nous pouvons définir simplement les cas d'utilisation du système. (fig 3.2. L'utilisateur doit pouvoir définir les paramètres initiaux du problème (trajectoires objectifs, parties éventuellement connues du mécanisme, contraintes de conception etc.), contrôler le déroulement de la résolution (*start*, *pause*, *resume* ...), observer le mécanisme solution courant et modifier les différents paramètres en cours de résolution. De plus, le chargement et la sauvegarde de modèles complets par le biais de fichiers doit être possible.

Parmi les interactions entre le système et les acteurs de l'environnement, les échecs potentiels à la coopération relatifs à la solution du problème posé sont possibles au niveau de l'utilisateur et de

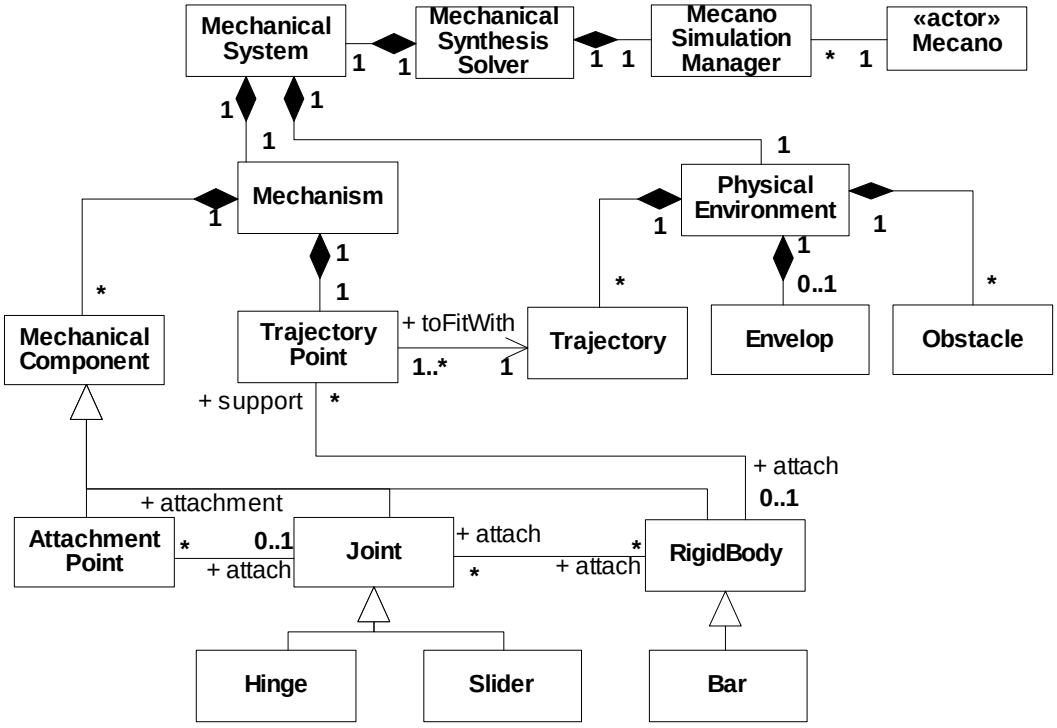

Figure 3.3 **—** Diagramme de classes préliminaire du MSS

MECANO :

1. Lorsque la solution courante ne convient pas à l'utilisateur, il a la possibilité de modifier les paramètres du problème ou de modifier directement le mécanisme afin de guider le processus.
2. A l'issue du calcul, par MECANO, d'un mouvement du mécanisme courant, l'état résultant peut être non coopératif : par exemple dans le cas d'un problème de suivi d'une trajectoire, le mécanisme peut passer d'un état -avant mouvement- dans lequel l'objectif est satisfait, à un état où celui-ci ne l'est plus.

3.1.2 Spécification du domaine

Suite à la définition des besoins, la méthode ADELFE nous amène à analyser le domaine de l'application puis à vérifier l'adéquation de l'utilisation des systèmes multi-agents adaptatifs au problème, ce qui est une phase spécifique de la méthode.

Il est aisé d'identifier un certain nombre de classes à partir des considérations précédentes (cf. 3.3). Tout d'abord une classe principale pour le coeur de l'application (*MechanicalSynthesisSolver*) est nécessaire. Le solveur doit pouvoir d'un part gérer les calculs de cinématique avec MECANO (*MecanoSimulationManager*) et d'autre part concevoir un système mécanique (*MechanicalSystem*). Nous pouvons décomposer le système mécanique en deux parties : le mécanisme lui-même (*Me-*

chanism) d'une part et les éléments constituant son environnement physique (*PhysicalEnvironment*) d'autre part.

Un mécanisme est lui-même constitué de composants mécaniques élémentaires qui peuvent être soit des corps rigides (*RigidBody*), soit des joints (*Join*), soit des points d'attachement fixes (*AttachmentPoint*). De plus, l'objectif d'un mécanisme étant de suivre une trajectoire, nous introduisons un autre élément afin de décrire les points du mécanisme qui suivent les trajectoires objectifs : *TrajectoryPoint*.

Dans la partie environnementale du système mécanique seront définies les différentes trajectoires objectifs (*Trajectory*), l'éventuelle enveloppe restreignant l'espace disponible pour le déploiement du mécanisme (*Envelop*) et les obstacles possibles que l'on peut rajouter (*Obstacle*).

On peut remarquer que le nombre de combinaisons possibles entre les composants mécaniques élémentaires étant très grand, la résolution de cette tâche est donc complexe car elle entraîne le parcours d'un espace de recherche gigantesque. Cette complexité est accrue si l'on considère que :

- Le concepteur ne connaît pas forcément le nombre et le type de composants que le problème requiert.
- Les modifications topologiques d'un mécanisme (même minimes) peuvent entraîner un grand changement pour la fonction qu'il effectue : l'espace des solutions est discontinu et il n'existe, a priori, aucune heuristique pertinente.
- L'utilisateur doit pouvoir modifier dynamiquement les paramètres du problème ainsi que la solution elle-même.

On note aussi qu'un mécanisme est naturellement composé d'entités plus simples (les composants mécaniques élémentaires) et potentiellement nombreuses (dans certains cas exposés par des utilisateurs finals, plusieurs centaines de composants peuvent être nécessaires) ce qui mène à la conclusion que l'utilisation d'un système multi-agent adaptatif est potentiellement adéquate pour traiter ce problème.

3.1.3 Identification des agents

L'identification des agents est une activité clé de la phase d'analyse de la méthode. ADELFE préconise, en premier lieu, de s'intéresser aux entités actives identifiées afin de déterminer si elles peuvent être des agents. Dans notre cas, nous n'avons identifié que l'utilisateur comme entité active or jusqu'ici il a été considéré comme extérieur au système et il n'y a nulle nécessité de lui octroyer un "représentant" au sein du systèm ; il possède seulement des moyens de contrôle, d'observation et de modification. Toutefois, la seule interaction potentiellement non coopérative identifiée précédemment est lorsque l'utilisateur juge que le mécanisme solution n'est pas satisfaisant. En analysant plus

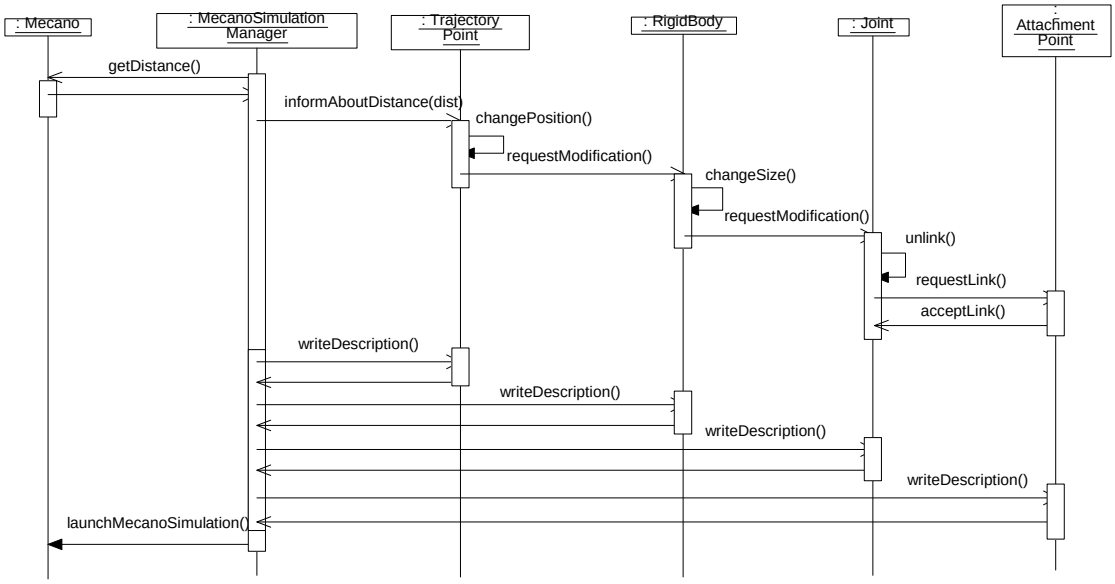

Figure 3.4 **—** Trace des interactions non coopératives issues d'un point trajectoire

précisément ce type d'interaction, nous pouvons voir que ce sont les composants du mécanisme qui devront traiter ces interactions. En effet, dans le cas où le mécanisme courant ne permet pas de remplir un objectif donné (le suivi d'une trajectoire), le traçage par le biais d'un diagramme de séquences 3.4 permet de voir que cet échec se traduit initialement par la détection d'une anomalie par le *TrajectoryPoint* associé à ce but, puis à des modifications de ce même point (il peut ajuster sa position par exemple) et/ou des autres composants du système qui peuvent eux-mêmes modifier leurs dimensions, leur position, voire même leurs relations ce qui mène à un changement topologique afin de tenter de résoudre le problème.

Les composants mécaniques (*MechanicalComponents* et ses sous-classes) et le point trajectoire (*TrajectoryPoint*) seront donc les agents de l'application. La méthode ADELFE permet de stéréotyper les classes associées avec un stéréotype *«Cooperative agent»* ce qui induit plusieurs règles à respecter quant à la conception de ces agents[3] que nous détaillerons dans la section 3.2.

3.1.4 Le processus global

L'étude des interactions entre le *MSS* et MECANO permet de décrire de manière globale le déroulement de chaque cycle de résolution. En effet, MECANO est ici utilisé pour calculer la cinématique du mécanisme, c'est-à-dire pour simuler l'activité de la solution courante afin de l'évaluer. Cette simulation est nécessaire dans le cadre de la théorie AMAS puisque le système multi-agent (ici le

[3]Pour une description détaillée du stéréotype *«Cooperative agent»* ainsi que des huit autres stéréotypes définis dans ADELFE voir (Picard, 2004, p. 134-140).

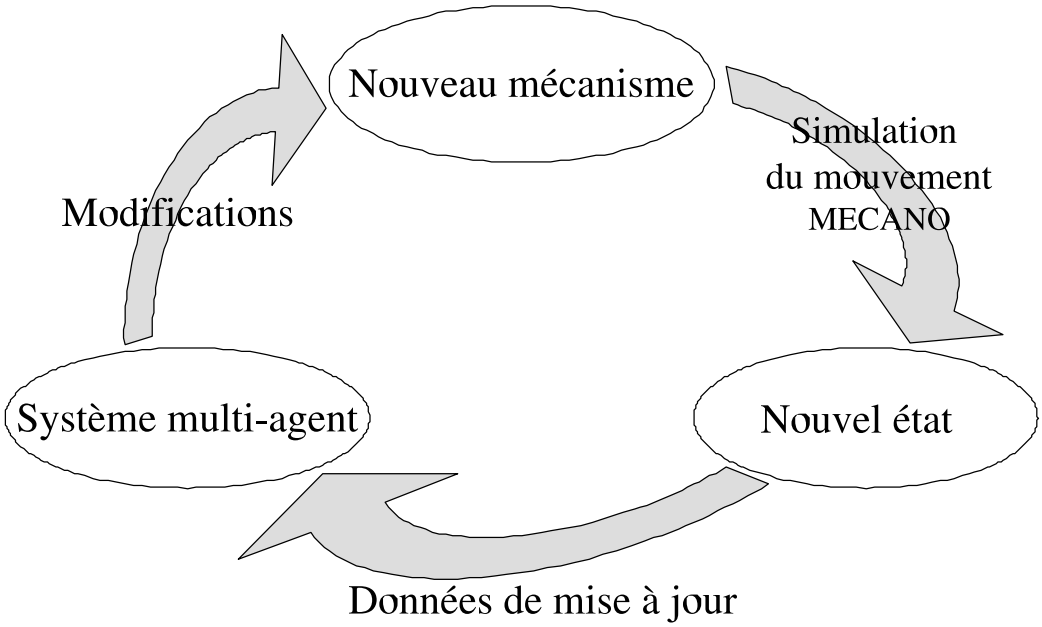

Figure 3.5 **—** Description d'un cycle de résolution

mécanisme) doit être plongé, en fonctionnement, dans un environnement afin de pouvoir s'adapter (la rétroaction système-environnement est le générateur de l'adaptation). De fait, chaque cycle de résolution est défini par trois phases comme l'illustre la figure 3.5 :

1. MECANO calcule le mouvement du mécanisme courant.
2. Les données relatives au nouvel état du mécanisme sont récupérées et communiquées aux agents pour mettre à jour leur état.
3. Les agents détectent et résolvent au mieux les situations non coopératives qu'ils détectent. Cette étape conduit à un nouveau mécanisme.

L'interface avec le simulateur se fait à l'aide de fichiers qui contiennent, en entrée, la description du mécanisme, de l'environnement et du mouvement de commande (la fonction d'actuation), et permettent de récupérer, en sortie, l'état du mécanisme après le calcul de mouvement effectué par MECANO.

3.2 Les agents

La conception des agents du système est la phase la plus délicate du processus de la méthode ADELFE. En effet, celle-ci conditionne fortement le bon fonctionnement de l'application et la définition des situations non coopératives ainsi que des actions associées pour leur correction représentent l'essentiel de l'effort (et des difficultés) de conception dans les systèmes multi-agents adap-

tatifs comme nous l'avons exposé dans la section 1.2.3.2. Pour cette raison, l'étude des situations non coopératives pour les agents du *MSS* sera détaillée dans les sections suivantes. Ne seront présentées ici que l'architecture en modules proposée par ADELFE et l'implémentation des différents types d'agents.

Partant du résultat de l'analyse qui a permis l'identification des agents (cf. 3.1.3), l'activité de conception des agents supportée par ADELFE (par le biais d'un modèle d'agent coopératif et de stéréotypes) est modulaire. La figure 3.6 illustre, de manière simplifiée, le résultat de cette activité. Nous avons, par commodité, rajouté une super classe nommée *MechanicalAgent* afin de définir de manière générique des agents coopératifs représentant soit des composants mécaniques, soit des points trajectoire. Ainsi, un mécanisme, tel que décrit dans le diagramme de classes préliminaire issu de l'analyse du domaine, est maintenant un système multi-agent (classe *MAS* non représentée) composé par l'agrégation d'*agents mécaniques* (instances de la classe *MechanicalAgent*). D'autre part, toutes les interactions entre agents ainsi que les notifications de modification de l'état des agents, issues des simulations de MECANO, seront gérées par des échanges de messages. Pour ce faire, un système de boîte aux lettres est utilisé : cette communication est gérée par la classe *MAS*.

3.2.1 Modules des agents coopératifs

ADELFE propose un modèle d'agent coopératif décomposé en sept modules qui représentent leurs capacités physiques, cognitives et sociales (Picard et Gleizes, 2004)(Picard, 2004). Dans ADELFE, chacun de ces modules est associé à un stéréotype[4] permettant d'exprimer la manière dont un agent coopératif, tel que défini sur la figure 3.7[5], est formé et/ou comment son comportement peut être exprimé.

«Perception» Le module de perception représente toutes les données qu'un agent peut recevoir de son environnement tant physique que social. La nature de ces données est relative aux stimuli perçus par l'agent : valeurs numériques (dans le cas de capteurs, par exemple), structures de données (la perception du type et de la concentration de phéromones pour des agents fourmis, par exemple), des messages (lors de communication entre agents logiciel notamment), etc. Le stéréotype associé s'applique à des attributs (pour représenter les données perçues) et aux méthodes décrivant les opérations de mise à jour de ces attributs. Les attributs stéréotypés *«Perception»* doivent nécessairement être privés ou protégés.

«Action» Ce module regroupe les capacités relatives aux moyens d'action de l'agent vis-à-vis de son environnement physique et social ou de lui-même (dans le cas où l'agent peut effectuer des

[4]Les noms des stéréotypes sont en anglais et entre chevrons.

[5]Le module caractéristique n'est pas représenté.

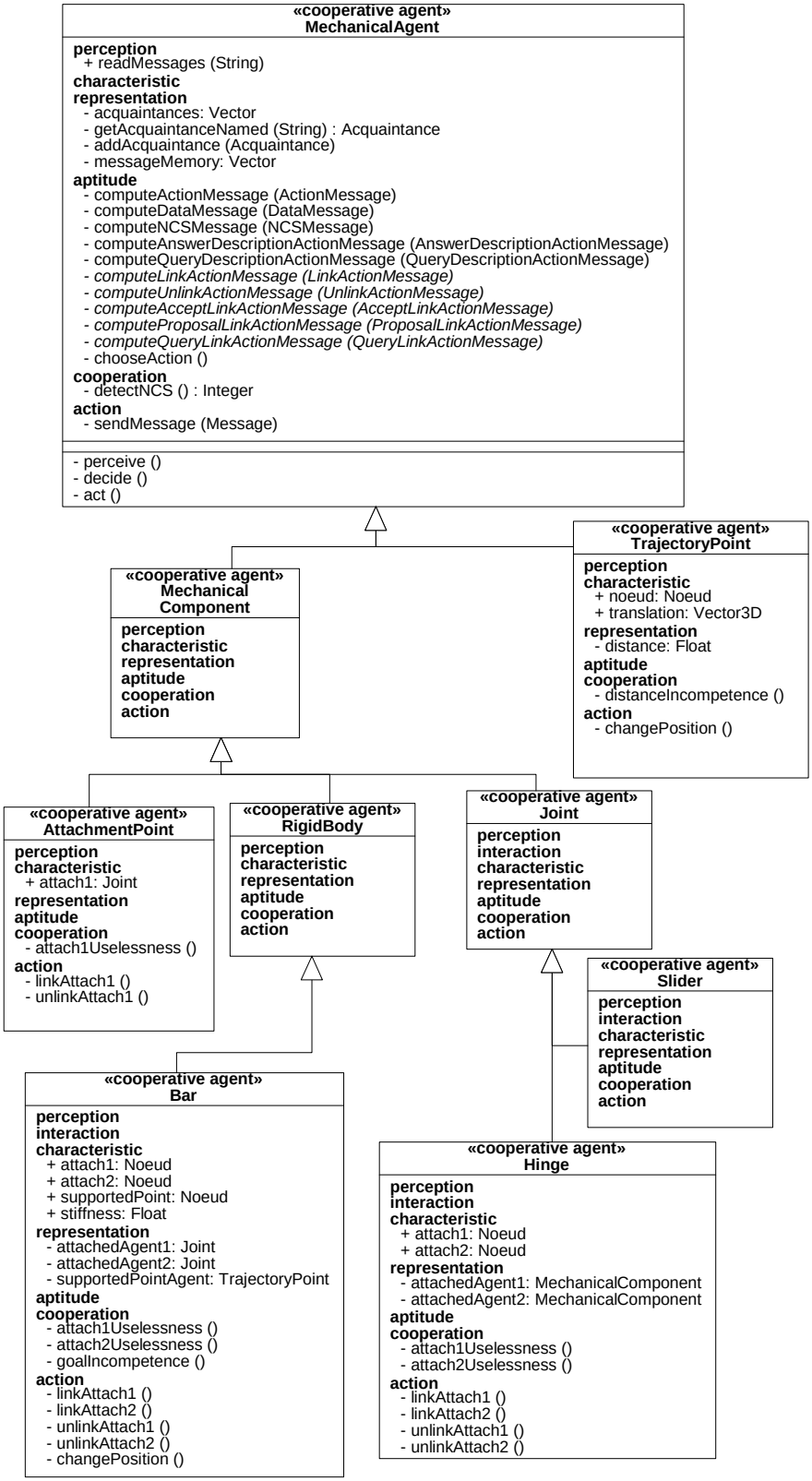

Figure 3.6 **—** Agents coopératifs mécaniques

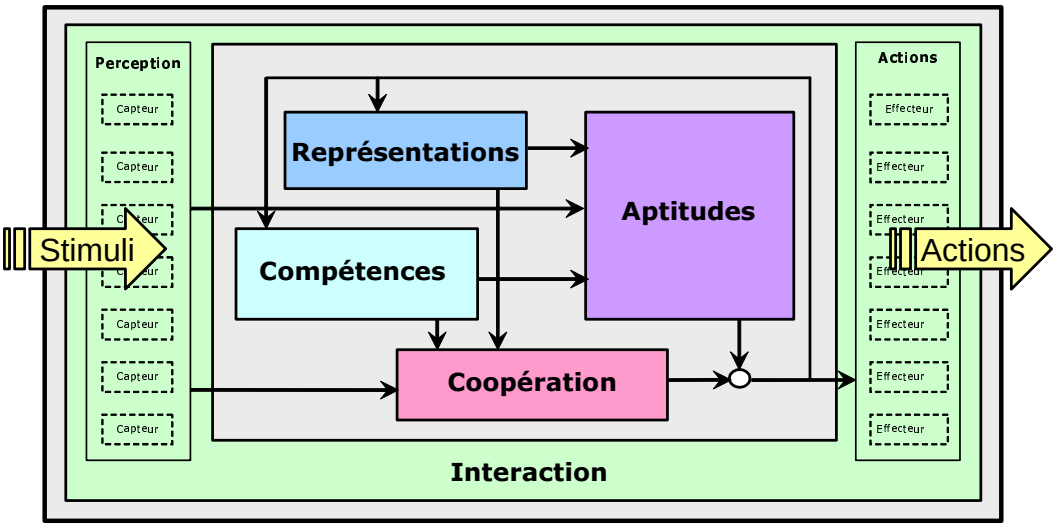

Figure 3.7 — Modèle d'agent coopératif

action d'apprentissage ou d'adaptation). Par exemple, sont considérés comme moyen d'action l'activation des effecteurs d'un robot, l'envoi de messages vers d'autres agents, la modification des dimensions d'un agent (pour des agents mécaniques), etc. Le stéréotype *«Action»* s'applique aux méthodes idoines et aux attributs qui sont les paramètres de ces méthodes. Les traits[6] ainsi stéréotypés sont nécessairement privés et utilisables seulement lors de la phase d'action de l'agent.

«Representation» Le module de représentation gère les croyances et connaissances sur l'environnement physique et social de l'agent ainsi que sur lui-même. Ce module peut, par exemple, être une base de faits et de règles dans un agent cognitif. Les attributs stéréotypés *«Representation»* seront les unités de connaissance et les opérations permettant les accès et modifications seront des méthodes *«Representation»*. Ces traits sont nécessairement privés ou protégés (préservant ainsi l'autonomie de connaissance des agents) et ne peuvent être invoqués que lors de la phase décisionnelle du cycle d'activité de l'agent. On notera que ce module est souvent dynamique (il est rare qu'un agent ait une représentation statique de son environnement) et peut, si besoin est, être lui-même conçu comme un système multi-agent adaptatif.

«Skill» Le module de compétence contient l'ensemble des connaissances permettant à l'agent de réaliser sa fonction partielle. Ainsi, ce module est au domaine d'activité de l'agent ce que les représentations sont à son environnement. Les règles associées au stéréotype *«Skill»* sont les mêmes que celles associées au stéréotype *«Representation»*.

«Characteristic» Sont considérées comme caractéristiques toutes les propriétés intrinsèques (position, type, nom, etc.) et physiques (forme, dimensions, couleurs etc.) d'un agent. Ces propriétés

[6] *Feature* dans le métamodèle UML.

seront représentées par des attributs stéréotypés *«Characteristic»*, et les méthodes d'accès à ces attributs le seront de même. Aucune règle particulière d'utilisation n'est associée à ce stéréotype.

«Aptitude» Les aptitudes représentent toutes les capacités de raisonnement sur les perceptions, les compétences et les connaissances d'un agent. Ce module contient toutes les données et opérations nécessaires pour définir quelle action effectuer pour un état donné de perception, en fonction des connaissances et compétences de l'agent. On peut noter sur la figure 3.7 que les actions déduites par ce module peuvent être subsumées par les résultats du module de coopération (voir ci-dessous). Au stéréotype associé sont rattachées les règles suivantes : les méthodes et attributs stéréotypés *«Aptitude»* ne peuvent être utilisés que durant la phase de décision de l'agent et sont nécessairement privés ou protégés ; de plus, les opérations *«Aptitude»* n'ont le droit d'accéder qu'aux traits non stéréotypés ou stéréotypés *«Characteristic»*, *«Perception»*, *«Skill»* ou *«Representation»*.

«Cooperation» La méthode générale de détection des situations non coopératives est *detectNCS()*. Cette méthode teste si chacune des conditions des situations non coopératives définies spécifiquement pour chaque agent est remplie. Elle retourne le nombre de situations non coopératives qui sont détectées.

Les deux derniers stéréotypes définis par ADELFE sont : le stéréotype *«Cooperative agent»* qui définit le cycle de vie de l'agent, et le stéréotype *«Interaction»* qui "englobe" les capacités de perception et d'action de l'agent et dont l'utilisation s'avère nécessaire dans certains cas (pour stéréotyper un attribut "boîte aux lettres" par exemple puisqu'il peut être utilisé tant pour la réception que pour l'envoi de messages).

3.2.2 Implémentation des agents

La super classe *MechanicalAgent* implémente de manière générique le modèle d'agent coopératif pour les agents mécaniques. Tout d'abord, cette classe implémente les quatre méthodes induites par le stéréotype *«Cooperative agent»* (Picard, 2004) :

1. **run()** : cette méthode simule le cycle de vie de l'agent. Elle est constituée de l'appel successif aux trois méthodes *perceive(), decide(), act()*.
2. **perceive()** : l'agent lit dans la boîte aux lettres tous les messages qui lui sont adressés.
3. **decide()** : l'agent traite les différents messages reçus, cherche à détecter d'éventuelles situations non coopératives et détermine l'action qu'il faudra mettre en oeuvre.
4. **act()** : l'agent effectue l'action décidée.

Lors de chacune de ces phases, l'agent utilise les ressources de chacun des modules du modèle conformément aux règles attachées aux stéréotypes d'ADELFE. Les principaux attributs et méthodes

définis dans la classe *MechanicalAgent* sont donc stéréotypés en fonction du module de l'agent auquel ils appartiennent.

«Perception» L'unique méthode nécessaire à l'agent est *readMessages(String)*. Elle permet de récupérer dans la boîte aux lettres tous les messages adressés à l'agent (le nom de l'agent étant précisé en paramètre).

«Action» Les agents doivent pouvoir communiquer et donc, dans *MSS*, envoyer des messages. De fait, tous les agents possèdent la méthode *sendMessage(Message)* qui permet d'envoyer un message à un autre agent.

«Characteristic» Les caractéristiques de chaque agent sont propres au type de composant qu'il représente. Par conséquent, aucune caractéristique commune n'est définie au niveau du *MechanicalAgent*.

«Representation» Le module de représentation contient d'une part les attributs et méthodes relatifs à la gestion des accointances (*acquaintances*, *getAcquaintanceNamed(String)*, *addAcquaintance(Acquaintance)*, ..., et d'autre part une mémoire des messages reçus (*messagesMemory*).

«Aptitude» Dans ce module sont définies toutes les méthodes relatives aux traitements de messages que chacun des agents doit nécessairement pouvoir traiter. Certains de ces traitements sont communs à tous les agents (*computeActionMessage(ActionMessage)*, ..., *computeQueryDescriptionMessage(QueryDescriptionMessage)*) et par conséquent entièrement définis, alors que d'autres ne sont définis que de manière abstraite et devront être définies dans chacun des agents en fonction de leur type (*computeLinkActionMessage(LinkActionMessage)*, ...,*computeQueryLinkActionMessage(QueryLinkActionMessage)*). La méthode permettant d'effectuer le choix parmi les différentes actions possibles est aussi définie dans ce module.

«Cooperation» La méthode générale de détection des situations non coopératives est *detectNCS()*. Cette méthode teste si chacune des conditions des situations non coopératives définies spécifiquement pour chaque agent est remplie. Elle retourne le nombre de situations non coopératives qui sont détectées.

«Skill» Les compétences des agents mécaniques du *MSS* sont singulières : elles sont inconnues. En effet, l'activité du mécanisme étant simulée grâce à MECANO, les agents n'ont, individuellement, aucune compétence relative à la fonction qu'ils réalisent.

A partir de cette classe, sont dérivés les différents types d'agents. Dans le prototype du *MSS*, seuls quatre agents ont été implémentés : les points trajectoire, les barres, les points d'attachement et les charnières. En effet, ces agents permettent, a priori, de pouvoir tester la faisabilité de l'utilisation de systèmes multi-agents adaptatifs sur des problèmes de conception mécanique. Pour chacun des agents définis, les modules relatifs aux caractéristiques, aux actions, aux représentations et à la coopération

doivent être précisés. De plus, la cinématique du mécanisme étant calculée par MECANO, chaque agent contient une méthode lui permettant de décrire son état en langage BACON (le langage du préprocesseur de MECANO) dans un fichier.

Voici, dans les paragraphes suivants, les différents éléments de l'implémentation de ces quatre types d'agents. Les algorithmes de traitement des situations non coopératives de chaque agent seront décrits dans la partie suivante. La description des modules de coopération ne concerne donc ici que l'énumération et les conditions de déclenchement des situations non coopératives.

3.2.2.1 Agent *point trajectoire*

L'agent *point trajectoire* (*TrajectoryPoint*) a pour but de rester sur la trajectoire qui lui est assignée. Une seule trajectoire à la fois peut être assignée à un agent *point trajectoire* bien que celle-ci puisse être modifiée durant la résolution (la trajectoire assignée peut évoluer ou une nouvelle trajectoire peut être assignée). Bien que cet agent ne représente aucun composant mécanique réel, il est assimilé, dans le *MSS*, à l'un d'entre eux par commodité mais ne possède pas de propriété d'ordre mécanique. L'agent *point trajectoire* n'a donc pour seule capacité que de s'attacher à un agent représentant un corps rigide.

«Characteristic» L'agent *point trajectoire* possède deux caractéristiques : sa position absolue (*noeud*) et sa position relative au milieu de la barre qui le supporte (*transition*). Ces deux caractéristiques sont synchronisées lorsque la barre qui supporte le *point trajectoire* modifie sa position : un message est envoyé au *point trajectoire* pour qu'il recalcule sa position absolue.

«Action» L'agent *point trajectoire* peut mettre en oeuvre trois actions spécifiques :

1. *changePosition* : permet au point de modifier sa position relative (modification de *translation*).
2. *unlinkAttach* : permet au *point trajectoire* de se détacher de la barre qui le supporte.
3. *linkAttach* : permet au *point trajectoire* de s'attacher à une barre.

«Representation» En plus des représentations d'un agent mécanique (*MechanicalAgent*), l'agent *point trajectoire* référence, s'il y a lieu, l'agent auquel il est attaché. De plus, le *point trajectoire* mémorise sous forme de vecteur la distance à laquelle il se trouve de sa trajectoire cible (*distance*). Cette distance correspond à un vecteur pour atteindre le point le plus proche de la trajectoire si celle-ci est une fonction.

«Cooperation» L'agent *point trajectoire* peut détecter deux situations non coopératives :

1. *positionIncompetence* : l'agent détecte une incompétence de position s'il ne se situe pas sur la trajectoire (voir section suivante pour les différentes variantes de cette situation).

2. *attachUselessness* : le *point trajectoire* doit être attaché à un corps rigide pour être fonctionnel sinon il est en situation d'inutilité.

Le noeud permettant de définir la position du *point trajectoire* est décrit (tout comme les autres noeuds) de la manière suivante dans le fichier permettant d'effectuer la simulation avec MECANO :

```
.NOE I i X nx Y ny Z 0.0
```

Cette commande déclare un noeud dont l'identificateur est le numéro *i* et les coordonnées sont *nx* et *ny*. Le logiciel fonctionnant en deux dimensions, la composante sur l'axe Z a systématiquement la valeur 0.0.

```
.SAI ARCHI NOEUD i comp 1 2 styp 9153
```

Cette commande permet de récupérer après le traitement, les nouvelles valeurs pour les coordonnées (composantes 1 et 2 pour x et y).

3.2.2.2 Agent *barre*

L'agent *barre* (*Bar*) représente le seul type de corps rigide disponible dans le *MSS*.

«Characteristic» Un agent *barre* est défini par les coordonnées de ses deux extrémités (*noeud1* et *noeud2*).

«Action» [7] L'agent *barre* peut s'attacher à des agents de type joint à l'aide des actions *linkAttach1* et *linkAttach2*, ou bien s'en détacher avec *unlinkAttach1* et *unlinkAttach2*.

«Representation» L'agent *barre* maintient les informations relatives à son propre état vis-à-vis de ses voisins :

1. *attach1* : référence l'agent (s'il existe) aulequel il est lié par l'extrémité définie par *noeud1* (cet agent est forcément de type joint).
2. *attach2* : référence l'agent (s'il existe) aulequel il est lié par l'extrémité définie par *noeud2* (cet agent est forcément de type joint).
3. *supportedPoint* : référence, s'il y a lieu, l'agent *point trajectoire* attaché à la barre.

«Cooperation» L'agent *barre* doit gérer les situations non coopératives d'inutilité (*attach1Uselessness*, *attach2Uselessness*) lorsqu'il n'est pas attaché à d'autres composants. Bien que ces deux situations soient dissociées, elles sont interdépendantes : une barre qui n'est attachée que sur une extrémité peut être fonctionnelle. Les situations dans lesquelles la fonction qu'il réalise n'est pas adéquate (*goalIncompetence*) doivent aussi être traitées.

[7] Dans certains algorithmes présentés dans la section suivante, les agents *barre* peuvent aussi modifier leur position.

L'agent contient aussi une méthode permettant d'écrire en langage BACON sa description afin de pouvoir effectuer une simulation avec MECANO. Dans le cas où il n'y a pas de point tajectoire attaché sur l'agent considéré, l'agent est décrit comme suit :

```
.MCE I i RIGID N n1 n2
```

Cette commande permet de créer un élément rigide dont l'identificateur est le numéro *i* et dont les extrémités sont définies par les noeuds dont les identificateurs ont respectivement les numéros $n1$ et $n2$. Si un *point trajectoire* est attaché, l'élément est défini au moyen de trois noeuds, le troisième étant celui représentant le *point trajectoire*.

```
.MCE I i RIGID N n1 n2 n3
```

3.2.2.3 Agent *point d'attachement*

L'agent *point d'attachement* (*AttachementPoint*) permet de fournir des points fixes aux mécanismes. Malgré le fait que tous les points fixes utilisés par un mécanismes puissent être considérés comme faisant partie d'un unique corps rigide fixe (cf. 2.1.1), nous avons choisi de les considérer indépendamment les uns des autres, au niveau agent, puisqu'ils sont fonctionnellement indépendants.

«Characteristic» Un point d'attachement se définit par sa position et possède donc une seule caractéristique : *noeud*.

«Action» Les actions implémentées sont seulement les actions relatives à l'attachement (*linkAttach*) et au détachement (*unlinkAttach*) avec des agents de type joint. On notera que les points fixes pourraient, lors des "phases d'ajustement", modifier leur position ; nous n'avons pas retenu cette option pour le *MSS* afin de simplifier les premières études.

«Representation» Chaque agent *point d'attachement* mémorise, s'il y a lieu, avec quel agent de type joint il est lié (*attach*)

«Cooperation» La seule situation non coopérative que peut rencontrer un agent *point d'attachement* est celle d'inutilité (*attachUselessness*) lorsqu'il n'est attaché à aucun autre agent.

La commande permettant de fixer les six composantes (x,y,z,rx,ry,rz) d'un noeud *i* pour la simulation effectuée par MECANO est la suivante :

```
.CLM FIX NOEUD i COMP 1 2 3 4 5 6
```

3.2.2.4 Agent *charnière*

Les agents *charnière* (*Hinge*) sont les seuls types de joints que nous avons utilisés dans le *MSS* car ils étaient suffisants pour les tests que nous avons effectués.

«Characteristic» La fonctionnalité d'une charnière est d'articuler deux corps rigides en un point selon un degré de liberté (rz dans le *MSS*). Les caractéristiques d'une charnière sont donc deux noeuds (*noeud1* et *noeud2*) se trouvant toujours à la même position.

«Action» [8] Les actions disponibles sont celles relatives à l'attachement (*linkAttach1*, *linkAttach2*) et au détachement (*unlinkAttach1*, *unlinkAttach2*) de l'agent avec des corps rigides ou des points d'attachement.

«Representation» L'agent *charnière* maintient les informations relatives à son propre état vis-à-vis de ses voisins :

1. *attach1* : référence l'agent (s'il existe) avec lequel il est lié par l'extrémité définie par *noeud1* (cet agent est forcément de type *barre* ou *point d'attachement*).
2. *attach2* : référence l'agent (s'il existe) avec lequel il est lié par l'extrémité définie par *noeud2* (cet agent est forcément de type *barre* ou *point d'attachement*).

«Cooperation» L'agent *charnière* ne peut correctement fonctionner que s'il est rattaché à deux autres agents. De fait, les situations où l'une de ses attaches est libre sont des situations d'inutilité (*attach1Uselessness*, *attach2Uselessness*).

Les commandes nécessaires pour décrire une charnière dans le fichier de simulation sont les suivantes :

```
.MCE I i HINGE N n1 n2 n3
```

Cette commande déclare une charnière *i* à partir de 2 noeuds *n*1 et *n*2. Le troisième noeud est un noeud spécial permettant de mesurer l'angle de rotation relatif.

```
.MCC I i HINGE AXE1 3
```

A l'aide de la commande .MCC, l'axe de rotation de la charnière *i* est défini comme étant l'axe Z.

[8] Dans certains algorithmes présentés dans la section suivante, les agents *charnière* ont la capacité de modifier leur position.

3.3 Conclusion

Nous avons présenté ici les différentes étapes de conception du *Mechanical Synthesis Solver* en suivant les étapes de la méthode *ADELFE*. Cela nous a permis, à partir des spécifications et de l'analyse du domaine, de déterminer quelle devait être l'architecture générale du système et d'identifier que les composants mécaniques seraient les agents.

La conception et l'implémentation de ces agents mécaniques ont été effectuées conformément au modèle d'agent proposé par *ADELFE* et supporté, dans la méthode, par des stéréotypes. Nous n'avons pas détaillé, dans ce chapitre, les situations non coopératives des agents. Ce travail est l'objet des deux chapitres suivants dans lesquels nous décrirons, tout d'abord, les différents algorithmes testés pour la résolution des situations non coopératives d'incompétence (chapitre 4), puis le travail effectué sur les actions de ré-organisation et d'évolution (chapitre 5).

4 Optimisation - Régulation

L'objet de cette section est de décrire les travaux effectués pour le traitement des situations non coopératives d'incompétence. Comme l'illustre le diagramme de séquences de la figure 3.4 de la section précédente, ces situations ont pour origine la détection par un agent point trajectoire d'un différentiel entre sa position et la trajectoire qui lui est affectée. Par le biais de la relaxation (lorsque l'agent ne peut traiter seul un problème il demande à ses voisins de l'aider), tous les agents du système peuvent potentiellement rencontrer ces situations. De manière générale, dans les systèmes multi-agents adaptatifs basés sur l'auto-organisation coopérative, les agents peuvent effectuer quatre types d'actions pour résoudre les situations non coopératives :

1. Modifier leurs compétences pour adapter leur comportement afin de résorber le problème.
2. Relaxer tout ou partie du problème aux agents voisins si l'agent le juge nécessaire.
3. Effectuer des actions de réorganisation.
4. Effectuer des actions d'évolution.

Les actions 1) et 2) sont compatibles : un agent percevant un problème peut identifier les aspects relevant de sa compétence (ou de son incompétence dans le cas présent) et essayer de les résoudre, tout en relaxant aux autres agents les aspects qu'il ne peut traiter lui-même : ces modifications sont de l'ordre de la régulation. A contrario, les actions de réorganisation (l'agent essaie de trouver une place plus adaptée dans le système en modifiant ses accointances) et d'évolution (l'agent disparaît du système ou provoque l'apparition d'un autre agent) sont exclusives entre elles et vis-à-vis de la régulation. Le diagramme 4.1 illustre ces interdépendances dans le cadre du *MSS*.

Ainsi, notre premier effort s'est porté sur l'implémentation des méthodes relatives aux actions de régulation : dans le cadre du *MSS* la régulation entraîne des ajustements sur les dimensions des composants. En effet, il est nécessaire que cette phase d'adaptation soit développée en premier lieu car les échecs éventuels fournissent des données nécessaires aux déclenchements et traitements des phases de réorganisation. Les situations non coopératives liées à la régulation sont, comme le montre la figure 4.1, déclenchées soit par la détection directe d'un problème d'incompétence (mauvaise position pour un point trajectoire ou blocage du mécanisme pour les autres composants), soit par la réception d'un message notifiant une incompétence. Néanmoins, plusieurs facteurs ont contribué aux difficultés que

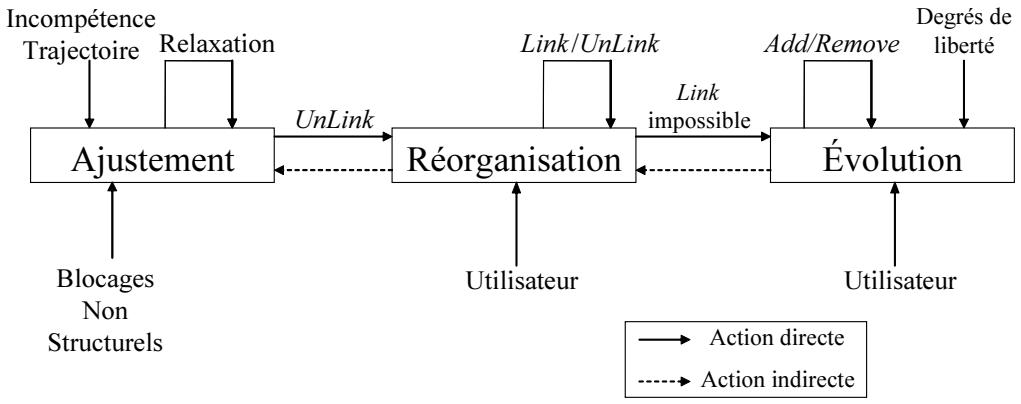

Figure 4.1 — Interdépendances des types d'action

nous avons rencontrées lors de cette phase de développement :

- Les carences de la théorie *AMAS* et de la méthode *ADELFE* quant au support pour la mise en oeuvre des actions correctrices : ce problème a été évoqué dans la section 1.2.3.2 ;
- La difficulté intrinsèque du domaine : l'espace de recherche est non linéaire et n'est continu que par parties ;
- Les compétences des composants nous sont inconnues (ainsi que les données de leur dynamique).

Les deux premières difficultés sus-citées sont les causes majeures du développement de nombreux algorithmes sur un mode essais-erreurs. Ces algorithmes sont l'objet principal de cette section et seront présentés regroupés en trois catégories[1]. Pour chacune d'elle, nous présenterons les principes sur lesquels reposent les algorithmes testés ainsi qu'un ou plusieurs exemples. La troisième difficulté relevée (le problème des compétences) est l'objet de la section qui suit.

4.1 Compétences des agents

La connaissance des compétences s'est avérée être un aspect de première importance lors de la mise en oeuvre du *MSS*. En effet, le calcul de la cinématique du mécanisme étant délégué à un logiciel tiers, les compétences ne sont pas intégrées aux agents. Or, afin de pouvoir traiter les situations non coopératives d'incompétence, les données et connaissances à propos des compétences sont nécessaires sinon aux agents mais pour le moins au concepteur des agents pour définir leur comportement. En tant que concepteur des agents, deux solutions se sont spontanément présentées pour acquérir ces informations : d'une part, l'expertise de personnes travaillant dans le domaine de la conception mécanique, et d'autre part, l'utilisation de la trace du processus de calcul effectué par MECANO.

[1]Ces catégories correspondent à la chronologie des travaux

La consultation d'experts du domaine s'est révélée peu productive du point de vue de la spécification des compétences. En effet, l'utilisation du paradigme agent semble trop éloignée de la manière conventionnelle dont sont considérés les composants mécaniques. Généralement, l'étude des mécanismes (optimisation, calcul de cinématique) est réalisé par le biais d'outils mathématiques et les comportements des différents éléments, sont de fait, retranscrits sous forme d'expressions mathématiques intégrées dans des systèmes d'équation. Les principes utilisés (comme le principe d'Hamilton) dénotent une approche résolument globale qui ne nous permet pas de définir les comportements locaux des composants mécaniques.

L'utilisation de la trace du processus de calcul effectué par MECANO nous est apparue non pertinente pour les mêmes raisons : il reflète l'évolution de variables au sein d'un processus global qui n'apporte pas d'information pour ce qui est des influences réciproques entre les éléments constituant le système.

4.2 Algorithmes d'ajustement

Durant la phase de développement, de nombreux algorithmes ont été testés pour les traitements des situations non coopératives d'ajustement. Ces algorithmes peuvent être classés en trois grandes catégories : les algorithmes réactifs, les algorithmes basés sur l'apprentissage et enfin les algorithmes basés sur une mémoire coopérative. Ces trois grandes classes se caractérisent de par les hypothèses choisies, et pour chacune d'entre elles plusieurs variantes ont été testées. Dans cette section, nous nous attacherons à exposer les principes et hypothèses qui ont présidé à l'élaboration des diverses versions ainsi que les éléments importants de leur mise en oeuvre. Des résultats, accompagnés de leur analyse, illustreront les raisons et motivations qui nous ont amené à faire évoluer la manière d'ajuster les composants.

4.2.1 Algorithmes réactifs

Cette première catégorie d'algorithmes est la plus simple et la plus proche de la manière dont est calculée la cinématique des mécanismes de manière classique. Elle se base essentiellement sur une fonctionnalité de l'outil MECANO qui permet de diffuser une raideur dans l'ensemble du mécanisme à partir d'un élément de type ressort. Ainsi, chaque composant du mécanisme subit une raideur à partir de laquelle il va pouvoir décider des modifications à mettre en oeuvre afin de la réduire.

4.2.1.1 Principe

L'idée à la base de ces algorithmes dits réactifs est donc l'exploitation, au niveau de chacun des corps rigides du mécanisme, d'une raideur induite au niveau des points trajectoire. L'induction de la raideur est réalisée par l'ajout d'un composant de type ressort ("*sping*") entre un point trajectoire et la trajectoire qu'il a pour objectif. La raideur ainsi générée dans le mécanisme n'intervient pas dans le calcul de la cinématique car la simulation est faite en mode statique. Plus la distance entre le point trajectoire et la trajectoire est grande plus la raideur diffusée dans le mécanisme est importante. Les valeurs de raideur pour chacune des barres sont récupérées à chacune de ses extrémités dans un repère local dont l'axe des abscisses est l'axe de la barre et l'axe des ordonnées la normale à l'axe des abscisses dans le sens trigonométrique. Tous les algorithmes calculent, à partir de cette mesure, les modifications à appliquer pour chacune des extrémités.

De manière imagée, cette approche peut être vue comme l'application à tout le mécanisme d'une "force" tendant à amener le mécanisme vers la trajectoire objectif sans que cette "force" n'intervienne dans la cinématique. La diffusion étant globale, l'algorithme de traitement de la raideur au niveau des barres n'applique ensuite que des actions locales d'ajustement et ne relaxe pas de problème aux autres agents puisque tous les composants rigides du mécanisme perçoivent les valeurs locales d'un même "champ" et donc perçoivent la même information.

4.2.1.2 Implémentation

Afin de récupérer les composantes de la raideur pour chacun des corps rigides, il est nécessaire de rajouter, dans le fichier de description du mécanisme destiné à MECANO, la commande suivante pour chacune des barres :

```
.sai archi element i comp 10 16
```

Il faut aussi définir, comme précisé ci-dessus, un repère local. La commande qui suit permet la définition d'un tel repère.

```
.frame I i type cartesian
origin noeud n1
V1 noeud n1 n2 V2 0 0 1
```

Ce repère est ensuite associé à la barre au moyen de la commande .MCC.

```
.MCC I i RIGID frm1 i
```

Pour finir, il faut définir un capteur de distance entre le point trajectoire et la trajectoire cible.

```
.NOE I 99
.CLM fix noe 99
.MCE I c curv n 99 j d
.MCC I c curv ncs t
```

Les deux premières commandes permettent la création d'un point fixe. La troisième définit un capteur de distance ayant pour identifiant c pour mesurer la distance entre une courbe et un point donné j (le noeud représentant le point trajectoire dans notre cas) et stocker cette distance dans la première composante du noeud d. Le dernière commande associe au capteur de distance c la fonction définissant la trajectoire (identificateur noté t).

```
.MCE I s sto1 n d
.MCC I s sto1 kr k lt 0
```

Pour finir, ces commandes permettent de définir un élément s de type ressort qui induit une raideur proportionnelle à la distance stockée dans le noeud d. La valeur k est un paramètre de la fonction décrivant la loi associée au ressort.

Les agents représentant les barres du mécanisme modifiant leurs dimensions de manière non coordonnée, le mécanisme après ajustement est *désassemblé*. Pour résoudre ce problème, une fonctionnalité de MECANO d'auto-assemblage est utilisée : avant chaque calcul, MECANO effectue des modifications afin de ré-assembler les parties qui le nécessite. Par exemple, si une barre initialement attachée par chacune de ses extrémités avec des charnières modifie la position de l'une de ses extrémités, l'outil de calcul détecte le problème et procède aux modifications minimales permettant de réajuster les positions de la charnière et/ou de la barre.

Dans ce contexte, les agents barre se considèrent en situation non coopérative si la projection de la raideur sur leur axe est non nulle (on ne considère pas la composante normale). L'algorithme de traitement des valeurs de raideur est ensuite relativement simple. Pour chacune de ses extrémités, l'agent barre effectue une modification de la position du noeud associé proportionnelle au vecteur décrit par la composante projetée.

4.2.1.3 Résultats et analyse

Les tests effectués avec ce type d'algorithme l'ont été principalement sur deux mécanismes 4-barres. La figure 4.2 montre le premier d'entre eux. L'objectif assigné à l'unique point trajectoire du mécanisme est une trajectoire verticale. L'actuation est un balayage de 0.4 radians par pas de 0.05

radians. Sur le schéma n'apparaissent que 3 barres car la quatrième est le sol qui relie les deux points fixes placés respectivement en haut à gauche et en bas à droite du schéma.

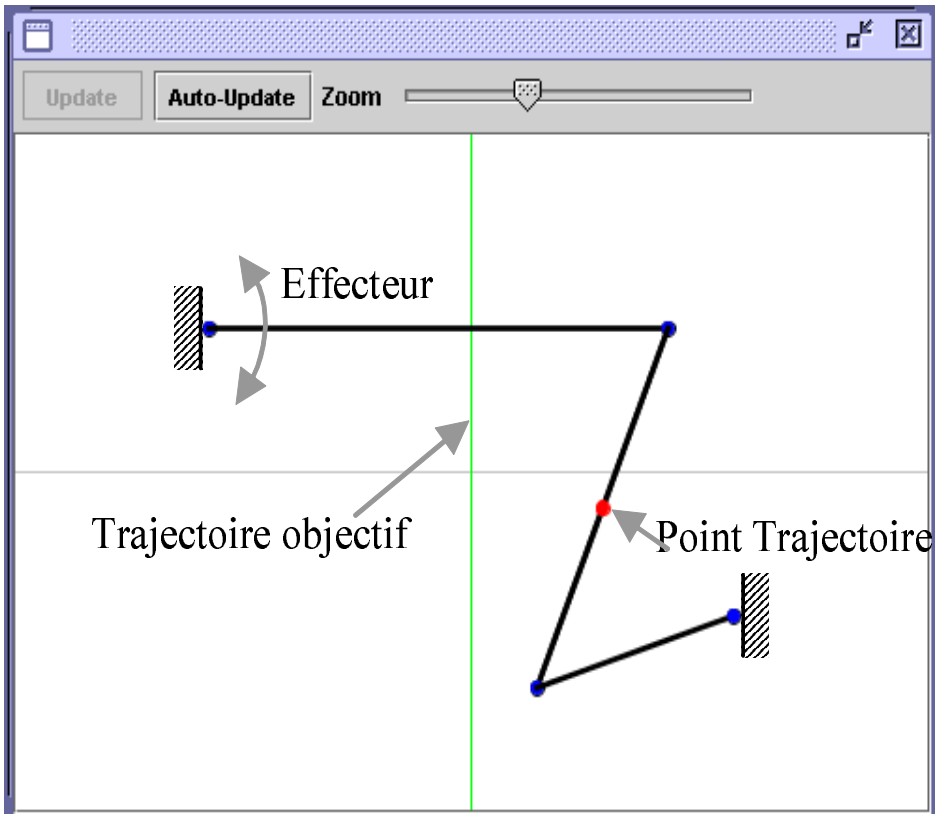

Figure 4.2 — Description d'un mécanisme 4-barres

De cette configuration, le mécanisme converge vers un état stable : les ajustements réalisés à chaque pas sont négligeables par rapport au dimensionnement du mécanisme (la différence entre les ordonnées des deux points fixes est de 40.0). Dans cet état, la distance à la trajectoire reste inférieure à 0.01. De plus la convergence est très rapide : comme l'illustre le graphique 4.3, dès le premier pas de simulation l'erreur chute d'une valeur supérieure à 10.0 à une valeur proche de 1.5. En fait, les agents atteignent une solution dès le premier cycle (un cycle représentant un balayage complet soit 16 mouvements).

Le même type de résultat est obtenu avec un mécanisme 4-barres légèrement différent dont l'objectif est une trajectoire horizontale comme illustré dans la figure 4.4. Dans cet exemple, l'effecteur est le même que précédemment et le point trajectoire n'est pas (contrairement à l'exemple précédent) directement sur la barre : le couple barre/point trajectoire est représenté par un corps rigide triangulaire par esthétisme.

La courbe de la figure 4.5 fait tout de même apparaître que le temps de convergence est significativement plus long que dans l'exemple précédent : le système n'atteint un état stable qu'après 40

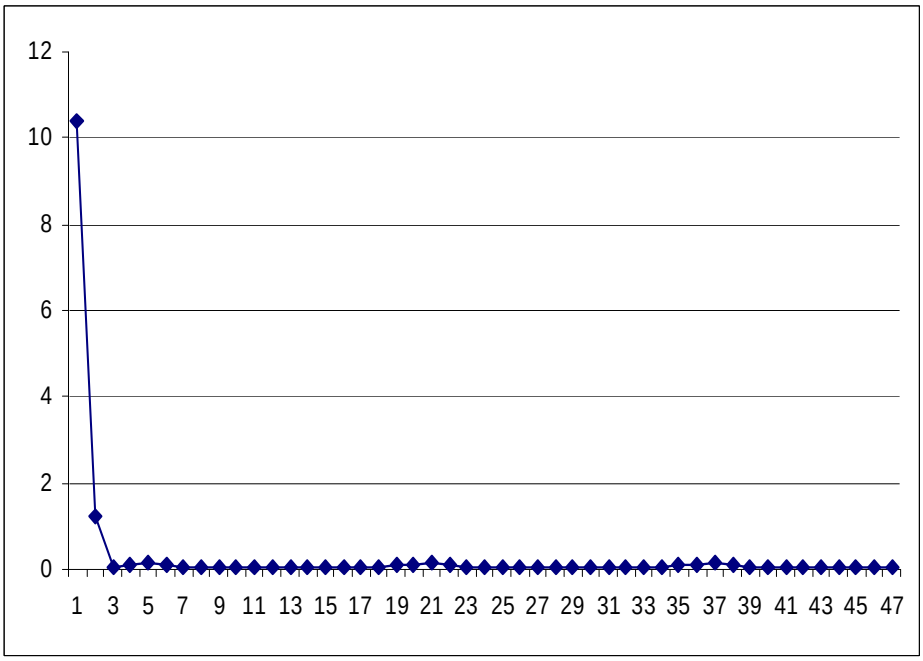

Figure 4.3 **—** Résultats d'un algorithme réactif pour un mécanisme 4-barres

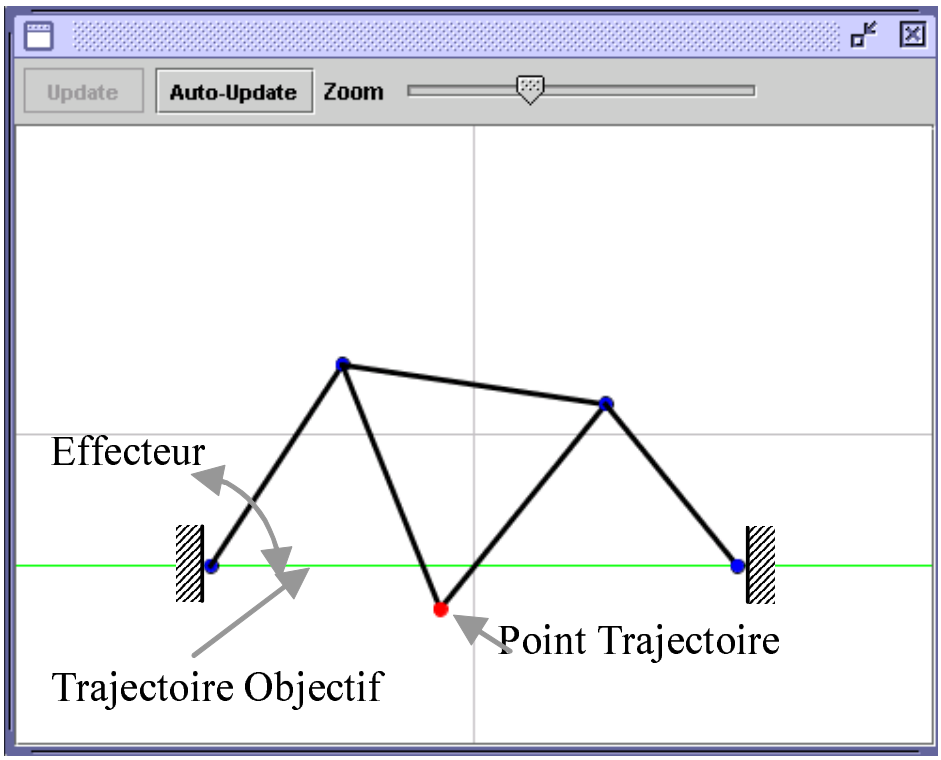

Figure 4.4 **—** Description d'un mécanisme 4-barres

cycles et ce pour une erreur de l'ordre de 0.15. Cela s'explique par le fait que le mouvement du point est "plus complexe" du fait du découplage entre le point trajectoire et la barre qui le soutient.

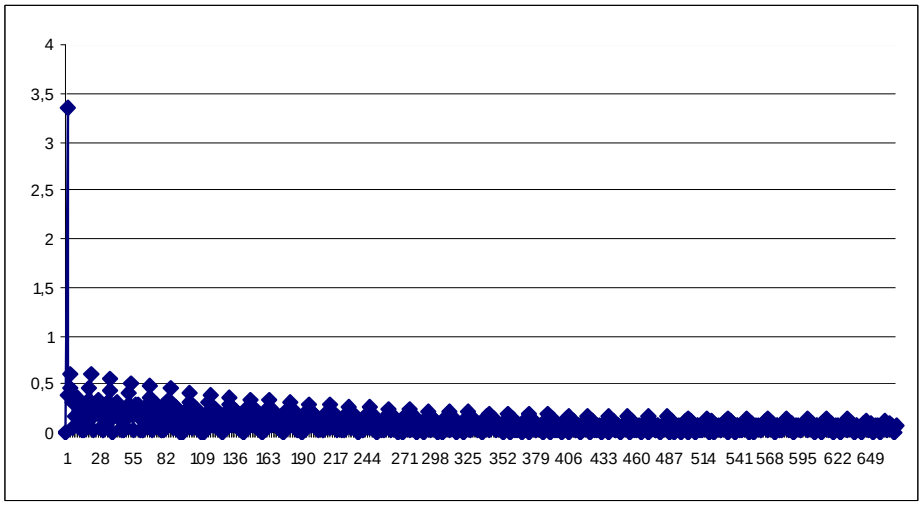

Figure 4.5 **—** Résultats d'un algorithme réactif pour un mécanisme 4-barres

Les résultats présentés ci-dessus sont caractéristiques de tous les tests que nous avons effectués sur ces deux types de mécanismes : pour toute modification mineure des dimensions de barres, le mécanismes converge vers une solution acceptable. Toutefois, des modifications plus importantes des dimensions et des positions des charnières donnent lieu à des résultats médiocres. La figure 4.6 illustre cet état de fait en mettant en parallèle deux résolutions : la première, pour un mécanisme 4-barres identique au premier exemple et une seconde, dans laquelle le mécanisme est déformé par passage de singularités.

Dans le second problème, il est nécessaire que le processus d'ajustement passe par une phase dans laquelle l'angle formé par les barres au niveau d'une charnière est plat. Dans ces positions le mécanisme est généralement bloqué mécaniquement. De plus, l'algorithme local de chaque barre n'applique que des modifications axiales, or le passage d'un point singulier en phase d'ajustement nécessite de manière évidente un déplacement normal à l'axe des barres. Au cours de nos tests, les seuls cas observés de passage de singularité en utilisant un tel algorithme se sont produits lorsque l'outil MECANO a effectué un effet miroir ou de symétrie : la résolution du système d'équations sur lequel est basé MECANO aboutit à un mouvement impossible physiquement qui s'apparente à une "téléportation" de la charnière.

En l'état, l'algorithme d'ajustement n'est donc pas satisfaisant car le processus de réorganisation provoque nécessairement des modifications structurelles qui aboutissent à des états à partir desquels le système ne sera pas capable de trouver une solution en ajustant les dimensions quand bien même cette solution existe. Afin de résoudre ce problème du passage des singularités, nous avons décidé

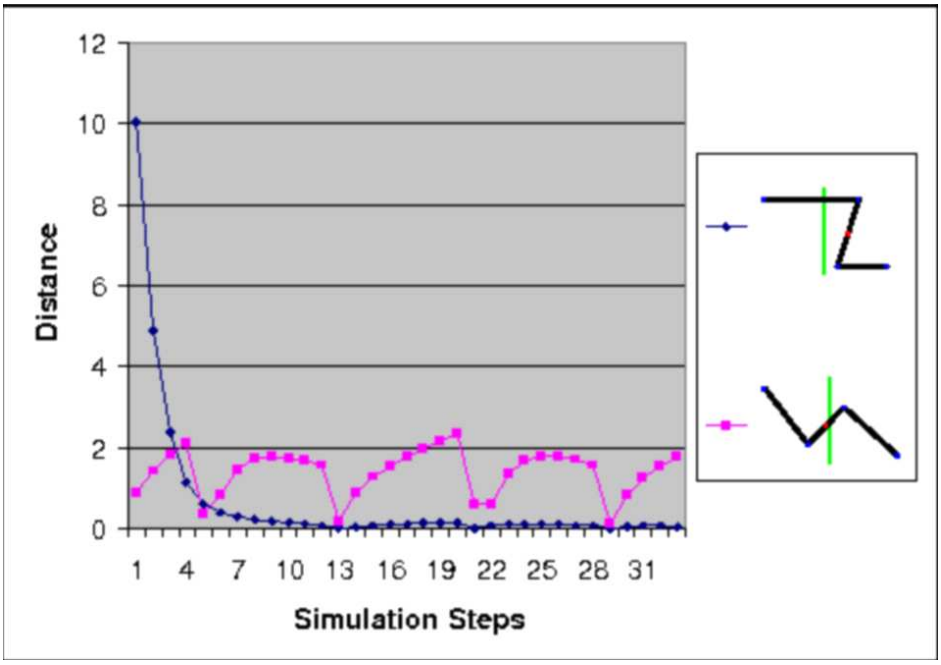

Figure 4.6 — Graphique d'illustration des limites d'un algorithme réactif

de complexifier quelque peu l'algorithme d'ajustement en ajoutant des capacités d'apprentissage aux agents.

4.2.2 Algorithmes basés sur l'apprentissage

L'idée est de pallier le problème du passage des singularités en permettant aux agents de choisir les modifications qu'ils doivent effectuer au moyen d'un apprentissage : en fonction des diminutions ou augmentations des raideurs reçues, les agents apprécient la qualité des ajustements qu'ils ont effectués aux pas précédent (apprentissage par renforcement).

4.2.2.1 Principe

De prime abord, cette capacité d'apprentissage a été rajoutée en se basant sur les algorithmes réactifs présentés ci-dessus : les agents barre établissaient des préférences sur les quatres actions possibles d'ajustement (augmenter ou réduire leur longueur en modifiant l'une de leurs deux extrémités). Mais une modification majeure est apparue nécessaire : les actions d'ajustement doivent être appliquées par les charnières plutôt que par les barres. La raison majeure de ce changement est due au fait que les opérations de réassemblage effectuées par MECANO induisent un "bruit" pour l'évaluation de la part des agents de l'influence de leurs actions dans la performance du mécanisme : un ajustement peut être entièrement annulé ou, au contraire, amplifié à l'issue du calcul de réassemblage rendant peu fiable

l'évaluation que les agents peuvent avoir des modifications qu'ils ont appliquées.

Les agents *barre* détectent donc une situation non coopérative lorsque leur raideur est non nulle, et ils envoient un message à chacune des charnières avec lesquelles ils sont liés. Ainsi, les charnières effectuent des modifications de leur position en fonction des valeurs qu'elles reçoivent des barres ce qui entraîne les ajustements au niveau des longueurs de celles-ci tout en maintenant un mécanisme assemblé. On remarque que les résultats obtenus en appliquant (sans apprentissage) comme déplacement pour chaque charnière la somme vectorielle des raideurs reçues des barres voisines donnent des résultats tout à fait similaires aux exemples montrés dans la section précédente en terme de convergence et de traitement infructueux lorsque des passages de singularités sont nécessaires.

Les algorithmes de modification des positions pour les charnières se basent donc sur la gestion de deux préférences : l'une pour l'axe absolu des abscisses (P_x) et l'autre pour l'axe des ordonnées (P_y). Les valeurs de ces préférences varient dans un intervalle centré sur 0.0 (les tests ont été effectués pour des intervalles [-100.0, 100.0]). La mesure de la norme du vecteur moyen des raideurs reçues permet l'évaluation sur laquelle se basent les agents *charnière* pour modifier leurs préférences. Ainsi, si la norme augmente, les préférences associées aux axes choisis pour effectuer le déplacement précédent sont diminuées sinon elles sont augmentées.

4.2.2.2 Implémentation

L'implémentation de ces algorithmes ne nécessite que la modification des méthodes de traitement des situations d'incompétence au niveau des agents *barre* (ils envoient maintenant aux charnières voisines des messages contenant la raideur détectée) et l'ajout, pour les agents *charnière*, de traits stéréotypés *«Cooperation»* idoines : le champ permettant de stocker les raideurs reçues et une méthode de détection d'incompétence (*goalIncompetence*).

4.2.2.3 Résultats et analyse

Nombre d'algorithmes ont ainsi été testés sur cette base mais aucun n'apporte réellement de plus-value intéressante par rapport aux algorithmes réactifs. De manière générale, ces algorithmes fournissent des solutions de qualité similaire puisque les erreurs résiduelles (en fin de résolution) sont du même ordre de grandeur. En terme d'efficacité, la résolution des problèmes présentés précédemment est quelque peu plus longue pour les premiers cas (mécanismes "en Z") car les modifications effectuées lors des premiers pas sont de moindre amplitude (les confiances sont proches de 0.0) et le système doit se stabiliser en fin de convergence (les confiances oscillent autour de 0.0). Pour les mécanismes correspondant au deuxième exemple (trajectoire horizontale et point trajectoire éloigné

de la barre), le temps de convergence est plus court car l'apprentissage permet de définir une "direction" et évite que les charnières effectuent des modifications contradictoires d'un pas sur l'autre (ce qui arrivait fréquemment avec les algorithmes réactifs).

Du point de vue du passage des singularités, les avancées sont aussi négligeables : seuls des cas spécifiques sont résolus. Pour exemple, la figure 4.7 illustre l'échec de la résolution du problème d'un mécanisme 4-barres "retourné".

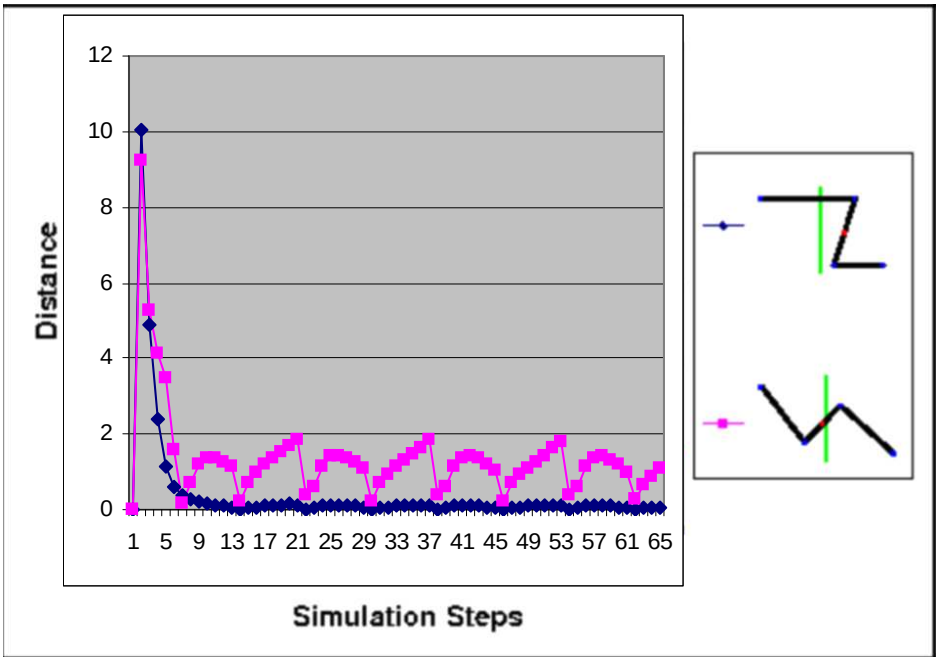

Figure 4.7 — Graphique d'illustration des limites d'un algorithme apprenant

A posteriori, ces résultats restent sans surprise : globalement, les algorithmes implémentés restent très éloignés des principes de la théorie des *AMAS*. En effet, l'utilisation de la raideur n'est que l'application à l'ensemble du mécanisme d'une fonction globale relative au but à atteindre. Or c'est précisément le type d'approche que la théorie des *AMAS* proscrit : les composants du système ne doivent pas connaître de manière explicite le but à atteindre. Ce constat nous a donc amené au développement des algorithmes suivants qui ne sont basés que sur des informations locales et non pas sur la raideur calculée par MECANO.

4.2.3 Algorithme basé sur une mémoire coopérative

Faisant table rase des précédents algorithmes, nous avons porté notre attention sur le point trajectoire puisque c'est cet agent qui doit être la source des situations d'incompétence. Ainsi, nous avons développé un algorithme de modification de position pour le point trajectoire qui ne tient compte que de la perception qu'il a du différentiel entre sa propre position et la trajectoire qui lui est affectée. Une

fois ce traitement testé et validé, nous avons, sur le même principe, développé les traitements associés aux agents barre et charnière.

4.2.3.1 Principe

Le point trajectoire. Lors de la résolution, le point trajectoire se trouve successivement confronté à des erreurs de position : il doit tenir compte de toutes les erreurs rencontrées au cours du temps et non pas seulement de l'erreur courante. La base de l'algorithme régissant les modifications de position du point trajectoire est donc une mémoire des erreurs rencontrées (on peut se représenter cette mémoire sous la forme d'un ensemble de vecteurs représentant chacun une erreur). Appliquant le principe de la théorie des *AMAS* sur le problème spécifique de la génération d'une modification à partir de la mémoire des erreurs, l'algorithme consiste à considérer chacun des vecteurs mis en mémoire comme un agent *erreur* dont le but est de diminuer sa norme.

Quand bien même l'implémentation de cette mémoire coopérative n'est pas purement "agent", les choix de traitement suivent les règles édictées par la théorie. Ainsi, si un nouvel agent *erreur* est ajouté alors qu'un (ou plusieurs) déjà présent se trouve avoir une direction similaire (un écart de 1 degré ou moins), il y a concurrence et agents *erreur* les plus anciens sont supprimés (l'information la plus récente est maintenue). Sont aussi effacées les agents *erreur* dont l'importance (la norme du vecteur par rapport à la moyenne des erreurs mémorisées) est négligeable car ils sont inutiles.

Dans le cas où la situation de l'agent *point tajectoire* change suite à un ajustement, les erreurs doivent réestimer leur importance car elle correspond à la mesure d'une erreur passée qui n'est plus juste dans le nouvel état (l'ajustement peut soit les corriger en partie, auquel cas la norme du vecteur correspondant doit être diminuée, soit les aggraver et le vecteur voit sa norme augmenter). Ces mises à jours sont effectuées par simple somme vectorielle : on somme chacun des vecteurs avec le vecteur correspondant à la modification. En terme de ré-évaluation, l'importance des erreurs diminue aussi d'un certain facteur à chaque pas de simulation afin de simuler le fait que plus une erreur mémorisée est ancienne plus elle est sujette à caution. Ces comportements correspondent à une révision des croyances des agents *erreur*.

La figure 4.8 explicite le calcul du mouvement de correction de l'erreur par opposition à un comportement non coopératif. Dans la partie supérieure, un algorithme de traitement de la plus grande erreur est appliqué : à partir d'un état donné (schéma de gauche) dans lequel il y a 3 erreurs, l'algorithme va appliquer la modification correspondante à une résorption totale de l'erreur la plus importante (vecteur "en pointillés"). Après application de cette modification (schéma de droite), on peut remarquer que parmi les erreurs réestimées en fonction de ce changement, il y en a une qui est plus importante que précédemment. Ce type d'algorithme ne correspond évidemment pas à un traitement

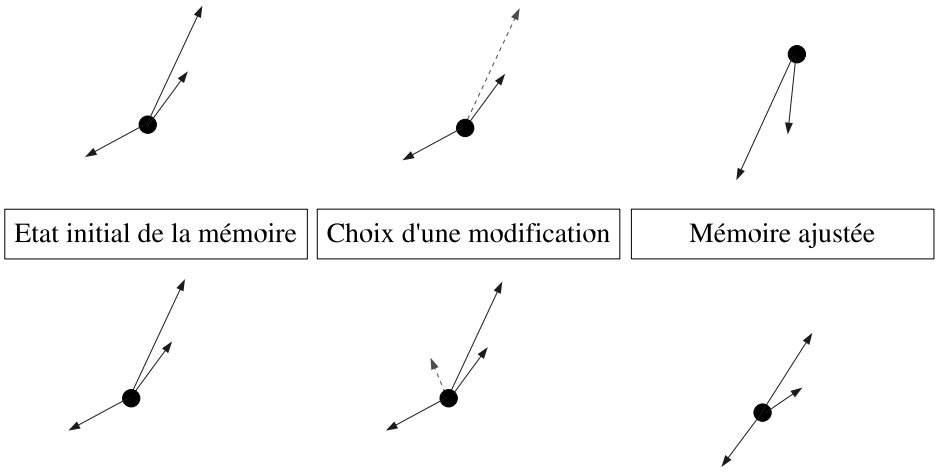

Figure 4.8 — Exemple de calcul coopératif (partie basse) et non-coopératif (partie haute)

coopératif : ce choix d'action est plus du type compétition dans lequel "le plus fort gagne". Un agent coopératif doit au contraire essayer d'atteindre son but (réduire la norme de l'erreur dans notre cas) en minimisant les effets négatifs que cela peut avoir sur l'activité des autres agents (augmentation de leur norme). Ainsi, l'algorithme que nous appliquons (voir partie basse de la figure 4.8) est le suivant :

Algorithme 2 : Calcul de la correction la plus coopérative pour un ensemble d'erreurs

- Recherche du vecteur *Max* ayant la plus grande norme ;
- Calcul du vecteur *Act* dont la direction est la même que celle de la moyenne de tous les vecteurs de la mémoire dans le même demi-plan que *Max* (angle inférieur à $\frac{\pi}{2}$), et dont la norme est égale à celle de *Max* ;
- Moyenne vectorielle de *Act* et du plus important des vecteurs du demi-plan opposé qui lui est opposé.

La première étape permet de définir quelle est l'erreur la plus importante à traiter. La deuxième établit une action coopérative en fonction de toutes les erreurs dont la résolution ne s'oppose pas à celle de l'erreur choisie. Lors de cette étape, l'algorithme ne calcule pas une simple moyenne vectorielle car nous ne considérons pas cela comme coopératif : dans le cas où interviennent beaucoup d'erreurs ayant de faibles normes par rapport à l'erreur sélectionnée à l'étape précédente, le vecteur moyen aura une norme presque négligeable ce qui est inapproprié pour corriger l'erreur. L'amplitude de la correction reste donc la norme de l'erreur initialement sélectionnée comme étant la plus importante (le comportement n'est pas altruiste). La modification de la direction en fonction de la moyenne est, a contrario, relativement coopérative : un nombre important de petites erreurs ne peut infléchir de beaucoup la direction de la correction. Toutefois, un grand nombre de vecteurs ayant des normes conséquentes peuvent influer sur la direction de l'action sans la modifier de manière drastique

(seuls sont pris en compte les vecteurs formant un angle inférieur à $\frac{\pi}{2}$ avec le vecteur référence). La troisième et dernière étape du calcul vise à prendre en compte les vecteurs opposés afin de limiter le préjudice que peut engendrer la correction.

Par ailleurs, nous avons adopté une stratégie de régulation progressive : l'agent point trajectoire n'applique que la moitié de la modification calculée à l'aide de la mémoire coopérative. Cette manière de faire comporte l'avantage majeur de modérer, en terme d'amplitude, les modifications effectuées par le point trajectoire : si cela rallonge de manière générale le temps de convergence, la robustesse du processus est renforcée surtout lors des premiers pas de simulation où les erreurs peuvent être importantes et les informations peu nombreuses et peu fiables.

Abordons maintenant le problème de la relaxation des erreurs aux autres agents. Un agent point trajectoire doit pouvoir notifier à son voisin (la barre par rapport à laquelle il se situe) les erreurs qu'il ne peut résoudre afin de déléguer leurs résolutions. En examinant le fonctionnement de la mémoire coopérative aucun élément ne nous permet de définir quelles peuvent être les informations à envoyer. De plus, ne connaissant pas les compétences des agents, il n'est pas possible d'établir un algorithme exact permettant de déduire quelle peut être la part prise par un agent donné dans l'activité d'un autre.

La solution que nous avons adoptée repose donc sur l'idée que les modifications des mouvements de la barre et du point trajectoire sont corrélées de la manière suivante : si la barre "bouge" moins, le point trajectoire bouge moins. Le point trajectoire envoie donc un facteur de modification d'amplitude. Si le mouvement effectué est plus faible en longueur que celui attendu, le facteur sera positif et dans le cas contraire négatif. Toujours dans une "optique de prudence", ce facteur est borné pour ne pas dépasser $frac12$ en valeur absolu. Une seconde information nous a paru pertinente : le différentiel de direction entre la trajectoire effectivement réalisée lors de la simulation et celle que l'on peut estimer avoir suivie. Cette mesure correspond à l'angle entre le vecteur du mouvement (on peut le calculer au moment où l'agent met à jour sa position après la simulation) et le vecteur calculé à partir des erreurs (on peut, à chaque pas, déterminer le point où l'agent aurait dû se trouver).

Comme on peut le remarquer, les caractéristiques de la trajectoire objectif assignée à un agent *point trajectoire* ne modifie en rien l'algorithme de traitement : seules comptent les mesures de distance entre le point trajectoire et la trajectoire à chaque pas de simulation.

Les charnières. On peut remarquer que l'algorithme ci-dessus peut être réutilisé pour traiter les modifications aux niveaux des charnières. En effet, du point de vue d'un agent charnière, les problèmes sont issus de la réception au cours du temps d'erreurs de la part des agents barre voisins. La problématique est donc similaire à celle à laquelle sont confrontés les agents point trajectoire : gestion d'une mémoire des erreurs reçues et calcul des corrections les plus adaptées. La solution adoptée est

par conséquent la même à ceci près que les agents charnière mémorisent les informations relativement à leurs voisins : le repère dans lequel sont stockés les vecteurs est défini dynamiquement par rapport à la position des barres voisines. La technique adoptée consiste à considérer un repère qui, à chaque pas de simulation, est mis à jour pour que l'axe des abscisses soit la bissectrice des deux segments que sont les barres (l'axe des ordonnées est calculé en conséquence).

Les barres. Les agents *barre* reçoivent soit de la part d'un point trajectoire, soit de celles des charnières avec lesquelles ils sont liés des messages de situations non coopératives contenant des erreurs. L'algorithme de traitement associé à ces situations est, contrairement à ceux des charnières et du point trajectoire, réactif : l'agent n'a pas à considérer les erreurs passées mais seulement à interpréter et relayer l'information en fonction de son l'état actuel (la prise en compte du temps est gérée dans les mémoires coopératives des agents *charnière* et point trajectoire). Cette problématique nous ramène, une fois de plus, proche de l'écueil auquel nous confronte l'ignorance des compétences exactes des composants mécaniques.

Le traitement consiste donc à calculer les modifications de position idoines pour chacune des extrémités de la barre à partir des informations suivantes :

- Les facteurs de modification d'amplitude de mouvement reçus des charnières voisines ;
- Les angles de déviations reçus des charnières voisines ;
- Les déplacements de chacune des extrémités lors de la simulation (sous forme de vecteurs) ;

Pour chacune des extrémités sont calculées :

- Une modification de position (perçue comme erreur sous forme vectorielle) qui sera envoyée à la charnière voisine correspondante pour qu'elle l'intègre dans sa mémoire coopérative ;
- L'information à relaxer en terme d'amplitude de mouvement (un facteur entre $[-\frac{1}{2}, \frac{1}{2}]$) et de rotation (angle) issues des demandes reçues par les autres extrémités.

Pour ce faire, l'algorithme calcule, pour chacun des deux types d'erreur (facteur et angle), si la barre doit se réduire ou se rallonger à l'extrémité considérée. Cette décision se fait en fonction de l'analyse des mouvements effectués par la barre durant la simulation précédente. L'étude de tous les cas (ils sont nombreux !) a été faite de manière géométrique "à la main" : nous avons considéré toutes les combinaisons possibles en fonction des directions des mouvements de chaque extrémité (divisées en quatre cadrans ayant comme base l'axe de la barre), de leur différentiel d'amplitude (quelle extrémité a effectué le mouvement le plus important), du signe du facteur de modification d'amplitude demandé, et de l'angle de rotation reçu.

4.2.3.2 Implémentation

L'implémentaion de ces nouvelles situations non coopératives n'a demandé que l'ajout ou la modification pour les agents concernés (*point trajectoire*, *barre* et *charnière*) des méthodes de détection de non coopération (les agents *point trajectoire* et *charnière* sont considérés dans un état non coopératif si leur mémoire n'est pas vide d'erreur et les agents *barre* s'ils reçoivent des messages de non coopération).

Les mesures de raideurs obtenues par MECANO n'étant plus nécessaires les commandes relatives ont été retirées.

4.2.3.3 Résultats et analyse

Nous nous sommes d'abord intéressés au fonctionnement de la mémoire coopérative. A cette fin nous avons établi un nouveau cas simple de test permettant de vérifier son bon fonctionnement pour l'agent point trajectoire. Le but est de résoudre un problème dans lequel un point trajectoire est attaché à une barre qui effectue une rotation autour d'un point fixe (l'actuation est la même que dans les cas de test précédemment exposés). La figure 4.9 montre (en haut à gauche) l'état initial du mécanisme dans lequel seul le module de coopération de l'agent point trajectoire est activé (la barre et la charnière ne peuvent modifier leurs positions pour l'aider). De manière évidente, il n'existe pas de solution parfaite mais le point trajectoire atteint la meilleure solution possible étant donné les conditions imposées (voir image du bas) et parvient à se stabiliser (les corrections effectuées à chaque pas sont négligeables car l'état de la mémoire est en équilibre).

Ce résultat encourageant obtenu, nous avons testé l'ajustement des cas de test utilisés pour les algorithmes précédents. Comme l'illustre la figure 4.10 (test d'un 4-barres de deuxième type), les résultats observés sont similaires en terme de qualité (distance à la trajectoire) à ceux obtenus avec les autres algorithmes. Du point de vue de la vitesse de convergence, nous obtenons une légère amélioration du temps nécessaire pour la résolution quand bien même le problème d'ajustement est plus complexe puisque le point trajectoire est maintenant un élément non fixé devant lui aussi ajuster sa position.

Pour ce qui est des passages de singularités, nous avons pu observer une amélioration notable : le choix du repère local associé aux mémoires coopératives est apparu ici crucial. Le repère étant ajusté en fonction de l'orientation des barres voisines aux charnières, il permet à l'algorithme de pouvoir maintenir une dynamique de modification dans la bonne direction. Toutefois deux facteurs font que cela doit pouvoir être perfectible. D'une part, le moment du retournement reste problématique : le mécanisme doit passer dans un état bloquant (l'angle entre les axes des deux barres est plat) et il semble

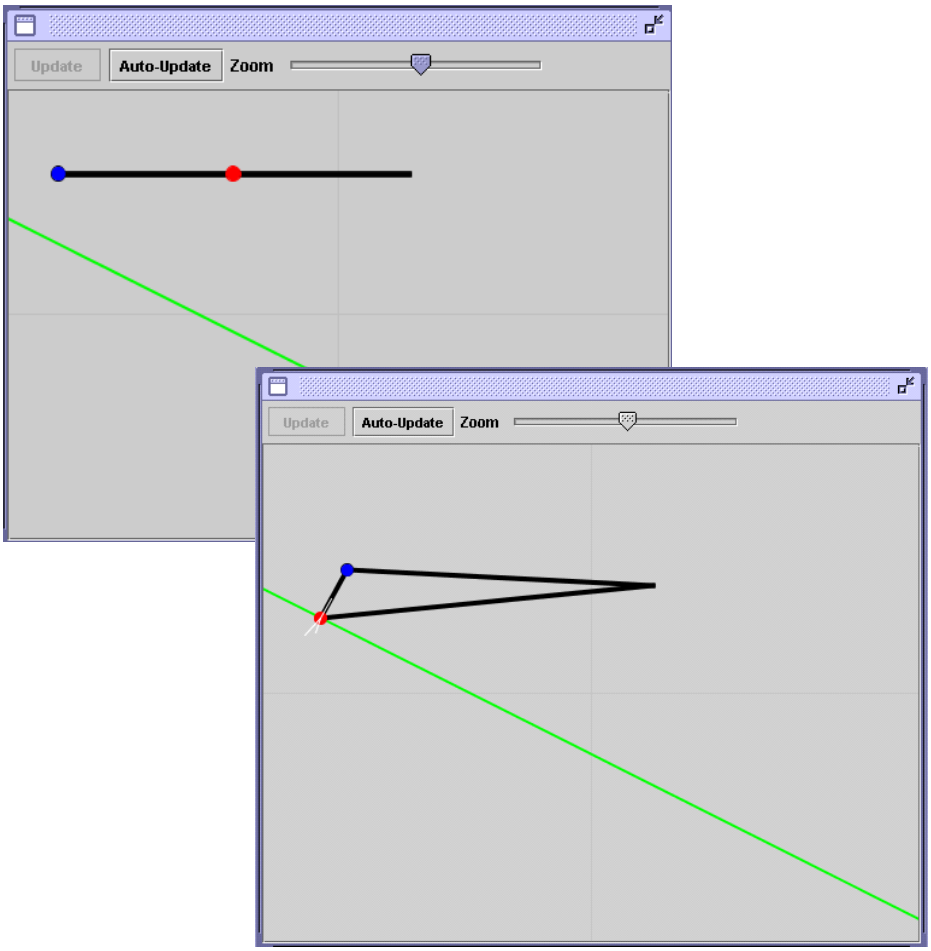

Figure 4.9 — Apprentissage du point trajectoire avec un algorithme basé sur une mémoire coopérative

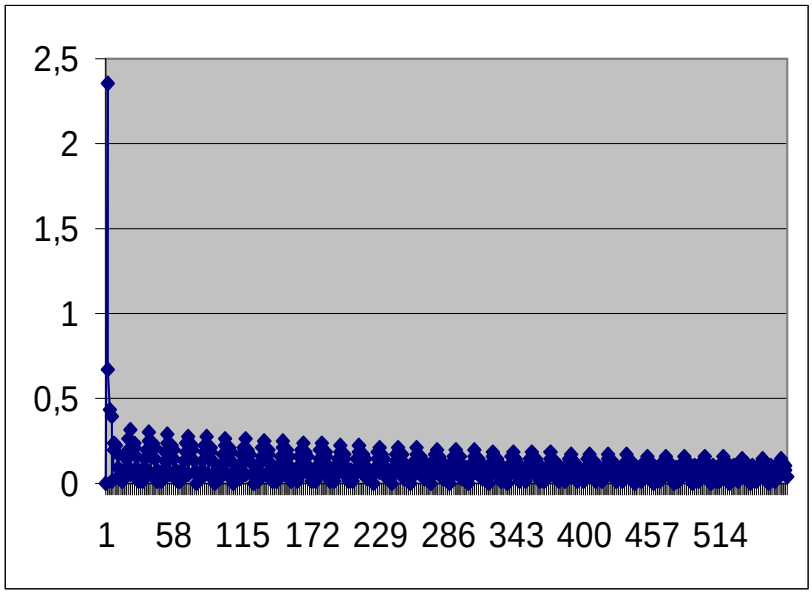

Figure 4.10 — Résultats de l'ajustement d'un 4-barres avec un algorithme basé sur une mémoire coopérative

nécessaire qu'au moment où l'angle bascule les informations stockées dans la mémoire soient modifiées pour tenir compte du changement (une idée pourrait être d'inverser les directions des vecteurs) et pour empêcher la charnière d'osciller autour de l'angle plat comme il nous est arrivé de l'observer. D'autre part, l'algorithme de traitement de l'information au niveau des barres n'est visiblement pas parfait et certains résultats restent non satisfaisants voire bloquants pour la résolution de certains cas. L'amélioration de ces calculs nécessite sûrement la connaissance des compétences des composants et des données de calcul de la cinématique afin de permettre une meilleure estimation des modifications à effectuer pour corriger les mouvements.

4.3 Analyse

Nous avons présenté, dans cette partie, le développement des algorithmes de traitement des situations non coopératives d'incompétences pour les agents mécaniques. Comme nous l'avons vu, les algorithmes basés sur l'utilisation des valeurs de raideurs n'a pas permis d'obtenir de résultats corrects. En effet, cette manière d'aborder le problème correspond à l'utilisation d'une information globale (la distance à la trajectoire) diffusée à tous les agents du système (par le biais de la raideur). Outre le fait que c'est précisément le type de fonctionnement que la théorie des *AMAS* essaie d'éviter (le but est d'utiliser l'émergence et donc la non connaissance du but global du système), il est apparu évident que, dans le cas qui nous préoccupe, l'évaluation locale de l'implication de chacun des agents dans l'activité du système est extrêmement difficile. Cette difficulté est d'autant plus exacerbée ici puisque nous n'avons pu établir les compétences des agents ni extraire une information significative quant aux données relatives à la cinématique.

D'un autre côté, du point de vue de la conception des systèmes multi-agents adaptatifs, il est clairement apparu que tant au niveau de la théorie des *AMAS* que de la méthode *ADELFE* il nous a manqué des outils de support pour les choix de conception relatifs aux situations non coopératives. En effet, quand bien même la théorie spécifie que l'on ne doit utiliser au niveau des agents que des informations locales, il s'avère que divers éléments intervenant lors de la conception d'une application peuvent rendre l'application de ce principe non trivial (pour exemple l'utilisation de la raideur dans les premiers algorithmes) du fait de la non connaissance du domaine.

Malgré les résultats obtenus, nous n'avons pu établir d'algorithmes parfaitement satisfaisants pour les actions de régulation. De fait, le travail portant sur les actions de réorganisation et d'évolution ne peut être mené de manière complète puisque toutes les informations issues des phases d'ajustement des mécanismes sont sujettes à caution quant à leur qualité. Nous avons néanmoins pris le parti de développer les actions de réorganisation et d'évolution afin d'illustrer le principe de construction

automatique de mécanismes.

5 Type Synthesis - Évolution

APRÈS avoir présenté les travaux portant sur les actions de régulation, nous allons maintenant aborder ceux relatifs aux deux autres types d'actions qui peuvent être mis en oeuvre par les agents : les actions de réorganisation et celles d'évolution. En examinant le schéma 4.1 présenté au début de la section suivante, nous pouvons voir que ces deux types d'actions qui participent à la modification de la structure du mécanisme sont liés :

- Lorsqu'un agent doit chercher un nouveau partenaire et qu'il n'en trouve pas de satisfaisant, il va effectuer des actions d'évolution ("*link* impossible").
- Une action d'évolution (apparition ou disparition d'agent) peut engendrer de nouvelles réorganisations au sein du système.

Ces liens de dépendance ne sont pas spécifiques au problème de conception mécanique mais communs, de manière générale, à tous les systèmes multi-agents adaptatifs. Le même type de dépendance apparaît entre les actions de régulation (ajustement) et de réorganisation : lorsque le processus d'ajustement échoue en un point du système (un agent perçoit des erreurs qu'il ne peut résoudre, par exemple), cela déclenche une réorganisation (initiée par le ou les agents détectant un problème insurmontable) par rupture d'une relation (dans le *MSS* la mise en oeuvre d'une action *UnLink* de détachement entre deux agents). Réciproquement, les actions de réorganisation permettent de produire des systèmes dont la fonction est différente et donc induisent, par la suite, la mise en oeuvre de nouvelles actions de régulation.

Dans le cadre du dévelopement du *MSS*, le travail effectué a été conduit dans un esprit de démonstration car la mise en oeuvre stricte des algorithmes de réorganisation et d'évolution nécessite au préalable que toutes les actions de régulation soient parfaitement finalisées, ce qui n'a pas été le cas ici. Nous présenterons donc ici ce travail illustré par les résultats obtenus sur le cas test de la construction d'un 4-barres pour le suivi d'une trajectoire verticale.

5.1 Situations Non Coopératives

Les situations non coopératives développées sont celles relatives à l'échec de la régulation (incompétence), aux blocages structurels et à l'inutilité.

Echec de l'ajustement L'objectif est ici de détecter les cas dans lesquels les agents ne peuvent plus améliorer le mécanisme à l'aide d'ajustement de dimensions. Cette situation peut se présenter dans deux cas :

1. Si le mécanisme a atteint la meilleure configuration possible relativement à l'agencement courant des composants.
2. Si un agent se retrouve confronté à des erreurs antinomiques.

Le premier cas ne peut être détecté que par les agents point trajectoire et charnière : cela correspond à un état stable dans lequel au fil des pas de simulation les ajustements deviennent négligeables car les demandes de modification qu'ils gèrent sont en conflit (il reste des erreurs en mémoire mais celles-ci s'annulent mutuellement car elles sont contradictoires). Pour ce faire, les agents enregistrent un historique des n[1] dernières mofications effectuées et vérifient que la dernière action calculée a une amplitude supérieure à un pourcent de la somme des amplitudes des n dernières actions enregistrées. Lorsqu'un agent détecte cette situation, il se détache de ses voisins par application de l'action *UnLink*.

Le deuxième cas porte uniquement sur les agents barres. L'idée est de détecter, pour une barre donnée, que les erreurs communiquées par ses deux voisins sont contradictoires et ne peuvent permettre de définir un comportement cohérent (allongement ou raccourcissement des extrémités). Nous n'avons pas développé cette situation dans le *MSS* pour la simple raison que nous n'avons pu trouver de cas où elle s'applique. Ceci peut être dû soit au fait que de tels cas ne peuvent se présenter dans le *MSS* à cause des choix opérés au niveau de la régulation soit, de manière plus probable à la vue des résultats présentés dans la section précédente, que l'algorithme de traitement des erreurs au niveau des agents barres est incomplet.

Blocages structurels. Les cas de blocages structurels se produisent lorsque l'organisation même des composants d'un mécanisme bloque son mouvement. Ce problème est un problème de manque de degrés de liberté du mécanisme. Dans les exemples sur lesquels nous avons travaillé, ces cas sont rares et facilement détectables : dans les exemples 4-barres, cela correspond à la détection d'une rotation inférieure à l'actuation demandée au niveau de la charnière sur laquelle elle s'applique alors qu'aucune charnière ne détecte un angle plat entre ses deux barres voisines. La mise en oeuvre *add-hoc* de cette situation consiste donc en la mesure de la variation d'angle pour la charnière d'actuation, puis à l'envoi de messages à travers le système. Si aucune charnière ne détecte un angle plat, le message arrive par relaxation à une charnière fixe (liée avec un agent *point d'attachement*) et cette dernière effectue une action de détachement (*UnLink*) et crée deux agents : une nouvelle barre et une nouvelle charnière.

[1] Nous avons arbitrairement pris une valeur de 23 pour n dans le *MSS* c'est-à-dire 1.5 fois supérieure au nombre de pas par cycle de résolution dans nos exemples.

Inutilité. Une situation d'inutilité correspond, pour un agent, à un état dans lequel il ne peut effectuer la fonction qu'il est censé réaliser. Pour les agents mécaniques cela correspond à la situation dans laquelle ils ne sont pas liés avec d'autres agents (ils ne peuvent donc pas participer au mouvement du mécanisme). Ainsi, un agent charnière doit être lié avec deux agents barre différents, un agent point trajectoire doit être attaché à une barre, un agent barre doit être lié à une ou deux charnières[2] et enfin un agent point d'attachement doit être lié à une charnière. la détection de ces situations est donc simple à réaliser : il suffit que les agents testent les caractéristiques associées (les *attach*).

Le principe pour résoudre le problème consiste en la recherche de partenaires. Pour cela, les agents envoient aux agents qu'ils connaissent un message de requête spécifiant le type d'agent avec lequel ils peuvent se lier. Les agents ayant un type compatible peuvent alors, si eux-même sont disponibles, envoyer une proposition à l'agent demandeur, lequel agent peut ainsi choisir une des propositions et effectuer l'action *Link*. Le point crucial de cette résolution réside dans le choix (s'il y a plusieurs réponses) de l'agent avec lequel se lier. C'est ici qu'intervient la coopération. Tout agent envoyant un message dans le cadre de cet "appel d'offre" y incorpore une évaluation de son degré de coopération. La stratégie consiste par la suite à choisir l'agent étant le plus coopératif possible. Chaque agent fait cette évaluation en fonction de son activité :

Agents *charnière.* Son degré de non coopération est évalué par le nombre d'erreurs présentes dans la mémoire non coopérative. La mémoire ne pouvant contenir que 360 erreurs au maximum (les erreurs ayant moins de 1 degré d'écart), la valeur de coopération envoyée correspond au complémentaire à 360 du nombre d'erreurs.

Agents *barre.* N'ayant pas de mémoire des erreurs qu'elle gère, une barre évalue son degré de coopération comme étant la moyenne des degrés de coopération des charnières voisines.

Agents *point trajectoire.* Le degré de coopération est calculé de la même manière que celui des agents *charnière* : à partir de la mémoire coopérative.

Dans le cas où aucune réponse n'est reçue pour résoudre la situation, les agents créent un agent du type recherché et se lient avec lui. De plus, les agents raisonnent aussi par rapport aux degrés de liberté que leur liaison peut restreindre. Les messages contiennent donc, comme information, les capacités de mouvement des agents : un point d'attachement se considère toujours comme fixe, une charnière se considère non fixe par défaut mais fixe si l'un de ses voisins est fixe, une barre n'est fixe que si ses deux voisins sont fixes et le point trajectoire se considère libre sauf s'il est attaché à une barre elle-même fixe. Cette information est prise en compte et permet aux agents de refuser certaines liaisons si celles-ci entraînent leur immobilisation (les barres et les points trajectoires refusent systématiquement de tels liens pour éviter les blocages de mouvement) ; c'est une condition nécessaire pour que le mécanisme

[2] S'il est indirectement lié à l'actuation une charnière lui suffit, sinon il lui faut s'attacher à une deuxième.

qui se constuit soit fonctionnel (il réalise une fonction), même si il n'est pas fontionnellement adéquat (il ne réalise pas forcément la "bonne" fonction).

5.2 Résultat

Ayant implémenté les situations non coopératives décrites ci-dessus, nous avons pris comme cas de test la conception d'un mécanisme 4-barres devant suivre une trajectoire horizontale. Initialement (voir figure 5.1), le mécanisme ne comporte donc qu'un point trajectoire (P), une barre (B_1), une charnière (C_1 qui est aussi l'actuateur) et les deux points d'attachement (A_1 et A_2 ce dernier n'étant pas représenté).

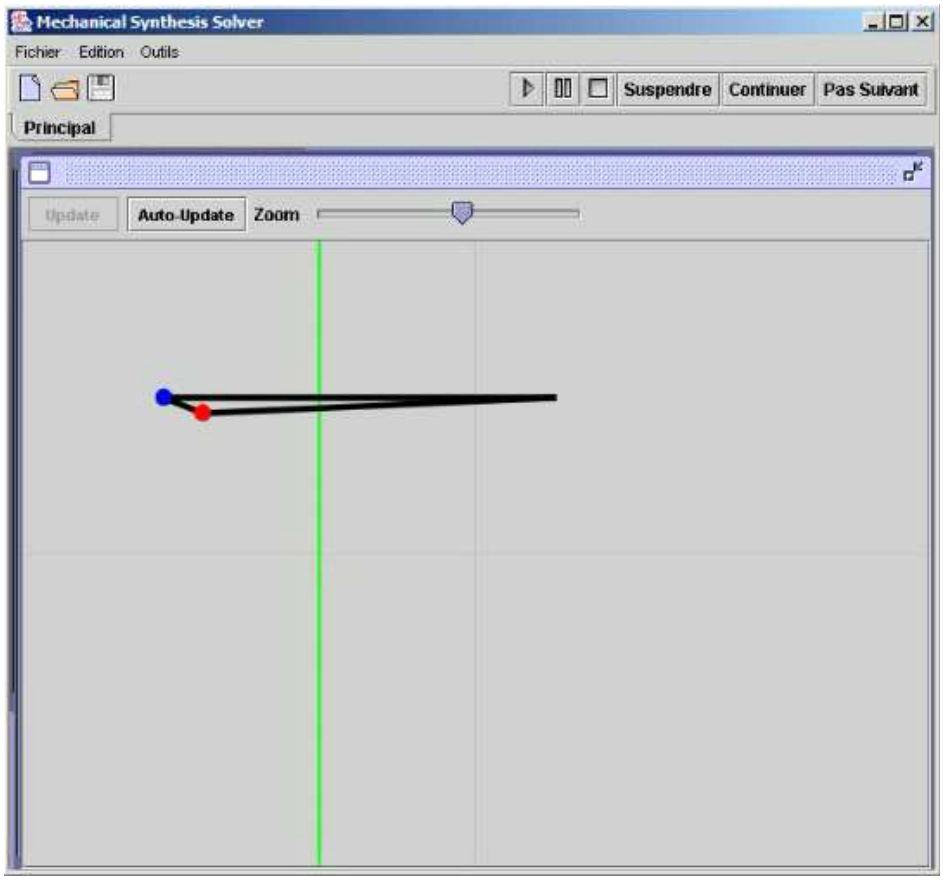

Figure 5.1 — Etat initial du système

Dès le premier pas de simulation, le point d'attachement non utilisé (A_2) est en situation non coopérative puisque non lié à un autre agent. Il n'y a dans le mécanisme aucun agent *charnière* disponible et par conséquent, l'agent *point d'attachement* crée un agent de type charnière (C_2). Ce dernier peut ainsi se lier à l'agent *point d'attachement* et se met en quête d'un partenaire de type barre (une charnière doit être attachée à deux agents). Le seul agent *barre* présent (B_1) dans le système

ne répond pas à sa requête (la nouvelle charnière est fixe car liée à un point d'attachement et la barre possède déjà une extrémité fixe) et le nouvel agent *charnière* (C_2) doit donc créer une nouvelle barre (B_2). L'agent *barre* B_2 se retrouve à son tour en situation d'inutilité puisque bien que relié à une charnière il ne bougera pas n'ayant pas de lien indirect avec l'actuateur. Aucun agent *charnière* n'étant disponible, l'agent *barre* va créer une nouvelle charnière (C_3), s'attacher avec et enfin cette nouvelle charnière pourra s'attacher avec la barre initialement présente (B_1).

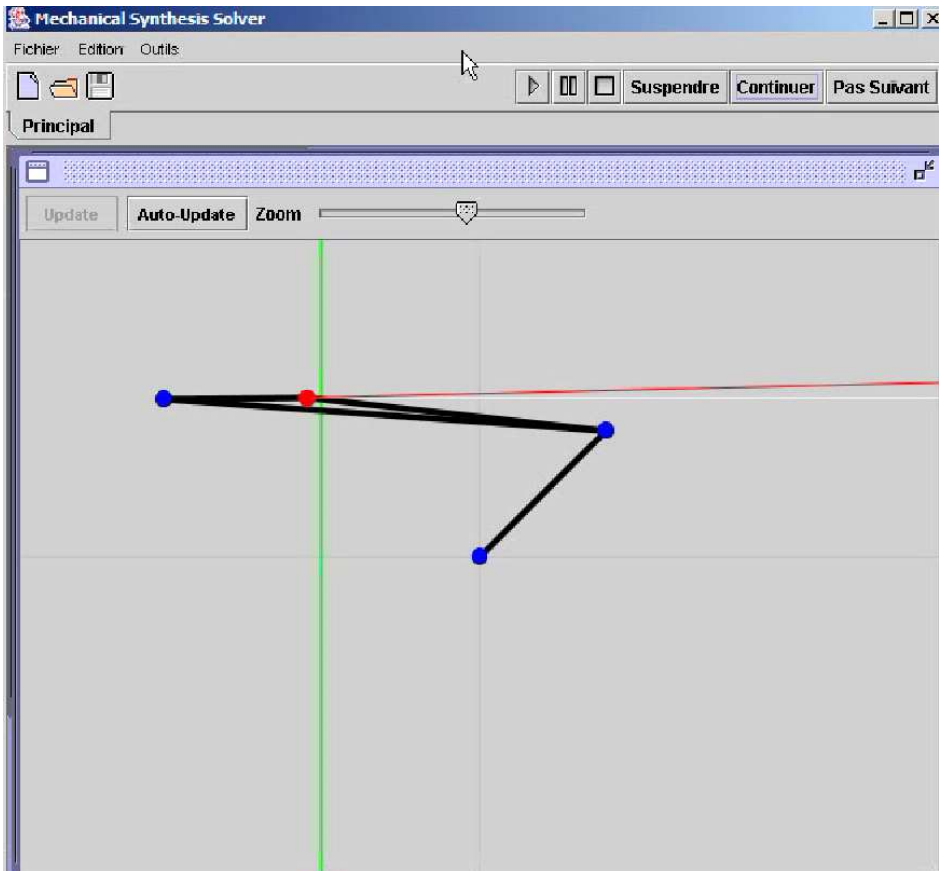

Figure 5.2 — Evolution vers un 4-barres

A partir de cet état (figure 5.2), le mécanisme essaie d'effectuer son premier mouvement. Suite au calcul de MECANO, l'agent *charnière* responsable de l'actuateur (C_1) détecte un blocage et envoie par conséquent un message à ses voisins. Aucun angle plat n'étant présent dans le mécanisme, ce message atteint la charnière (C_2) qui est fixe et qui crée en réaction une barre (B_3) et une charnière (C_4) qui s'insèrent dans le mécanisme. On peut remarquer que durant ce pas de résolution, le point trajectoire détecte qu'il n'est pas sur la trajectoire et effectue donc un ajustement de position (le trait partant du point trajectoire représente le vecteur de modification calculé par la mémoire coopérative). A l'issue de ces modifications, le mécanisme se trouve dans une configuration fonctionnelle (comme le montre l'image 5.3, le mécanisme peut se mouvoir) et les pas suivants vont consister en des actions

de régulation afin modifier les dimensions.

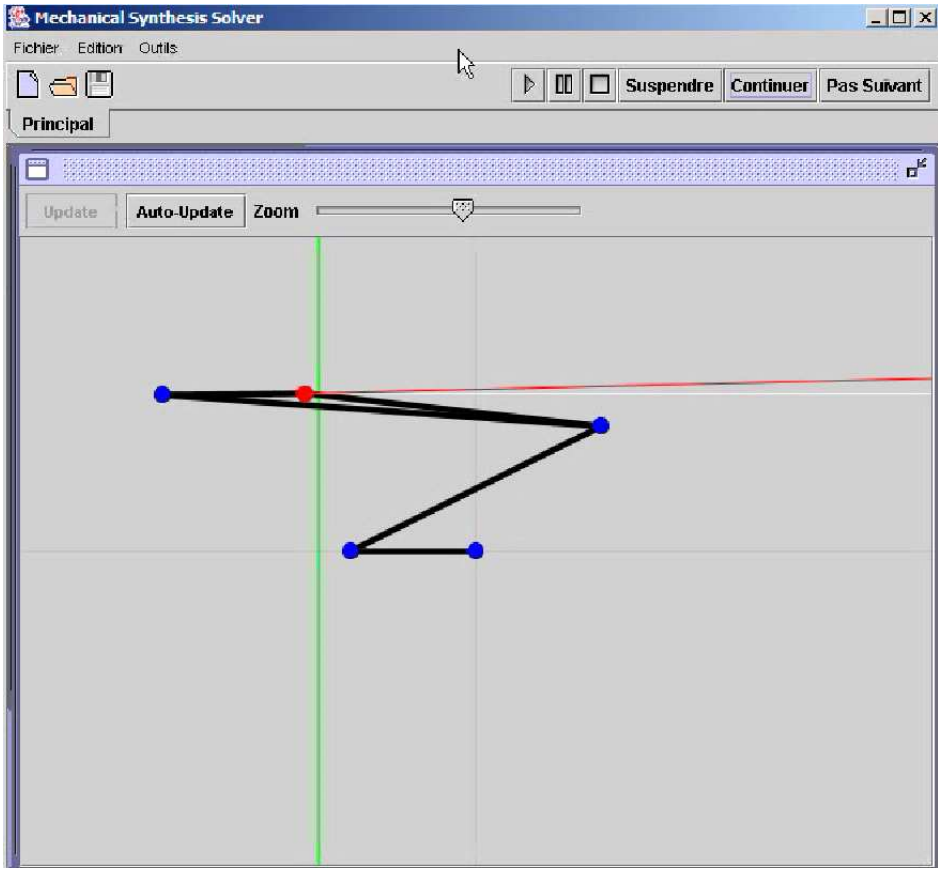

Figure 5.3 — Phase d'ajustement

De manière évidente, il n'y a en l'état pas de solution. Le point trajectoire va modifier sa position sans pouvoir trouver de position satisfaisante et les modifications effectuées par les autres agents, à partir des erreur relaxées, ne peuvent l'aider. En environ 95 pas de simulation, le point trajectoire va ainsi ajuster sa position jusqu'à atteindre un état stable et détecter l'échec de l'ajustement. Il va donc se détacher de la barre sur laquelle il se trouve (voir 5.4).

Après envoi d'une requête pour rechercher une nouvelle barre sur laquelle s'attacher, le point peut choisir entre les deux autres barres du mécanisme (B_2 et B_3). L'évaluation du degré de coopération est favorable à la barre du milieu (B_2) et le point trajectoire se lie donc à cet agent (par défaut sa position est au milieu de la barre comme le montre la figure 5.5).

La résolution entre ensuite dans une phase de réajustement pour atteindre après 350 pas de simulation un état stable avec une erreur très faible. L'image 5.6 est une capture d'écran sur laquelle on peut voir le mécanisme solution fourni par le *MSS* ainsi que la courbe de mesure de la distance entre le point trajectoire et son objectif. Sur cette courbe, le pic d'erreur au pas 90 de la simulation correspond au moment du changement de support du point trajectoire. Les traits centrés sur le point trajectoire

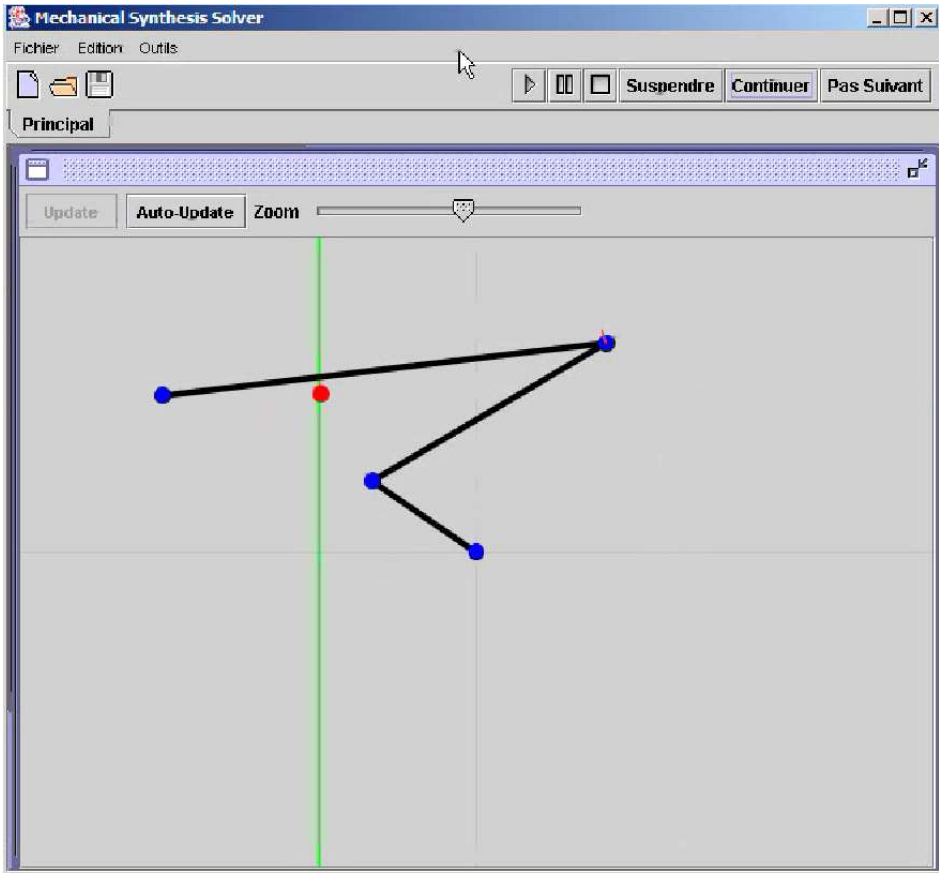

Figure 5.4 — L'echec de l'ajustement entraine une réorganisation

et les charnières centrales sont la visualisation graphique de l'état des mémoires coopératives (ces vecteurs étant très petits, la taille des traits n'est pas à l'échelle).

5.3 Conclusion

Le travail effectué sur les situations non coopératives de réorganisation et d'évolution nous a permis d'illustrer le principe du processus de construction d'un mécanisme. Dans l'exemple montré, un mécanisme contenant initialement les composants minimum définissant le problème du suivi d'une trajectoire verticale évolue, en se basant uniquement sur l'analyse des situations locales aux agents, vers un mécanisme 4-barres. Le processus se poursuit par une phase d'ajustement des dimensions pour converger vers la meilleure solution possible compte tenu de la topologie obtenue : le point trajectoire suit une trajectoire en arc de cercle coupant la trajectoire rectiligne. Cette configuration n'étant pas acceptable, le point trajectoire déclenche une phase de réorganisation en détectant que l'ajustement a convergé vers une solution non acceptable (l'erreur résiduelle est trop importante) et ce en se basant uniquement sur les informations locales dont il dispose. Après réorganisation, le mécanisme obtenu

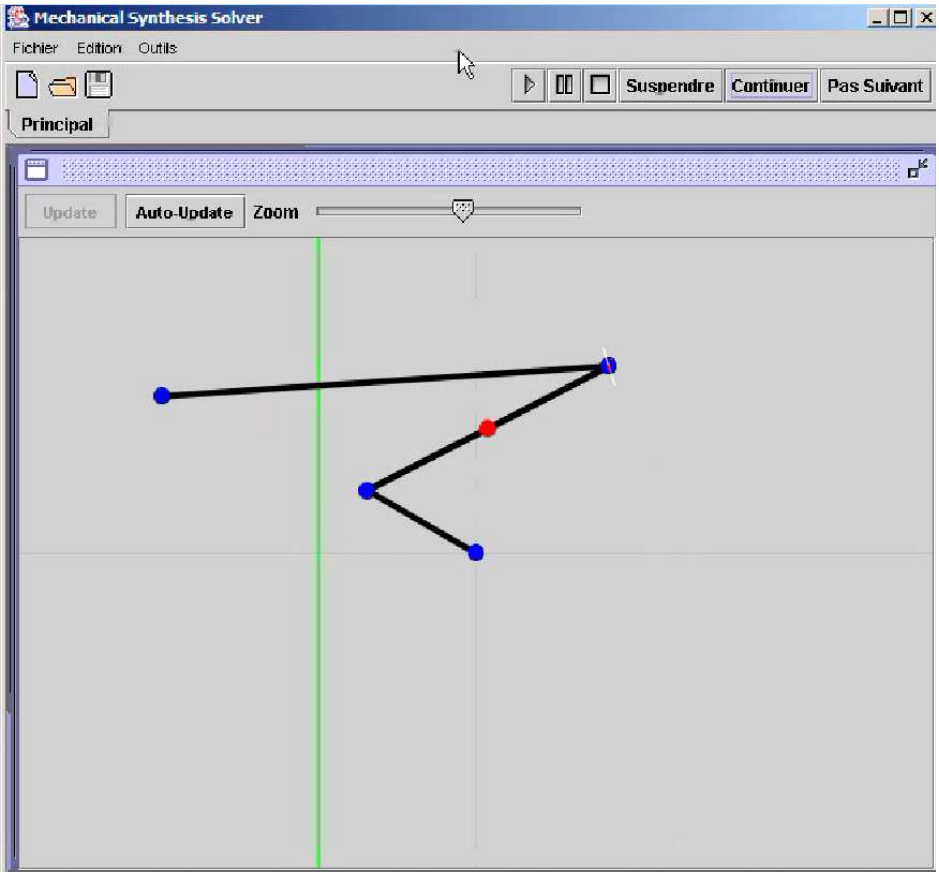

Figure 5.5 **—** Deuxième phase d'ajustement

est topologiquement bon et continue la résolution en optimisant les dimensions.

Bien que ce processus soit mis en oeuvre dans le *MSS* par le biais de situations non coopératives "add-hoc", les principes utilisés sont conformes à la théorie des *AMAS* : détection des situations non coopératives par analyse des informations locales aux agents et critères de choix pendant la réorganisation s'appuyant sur le principe de coopération. Pour généraliser ces processus de réorganisation et d'évolution afin qu'ils fonctionnent pour tout mécanisme, il serait nécessaire de se baser sur des informations plus fiables issues des actions de régulation, et sur l'analyse des degrés de liberté des composants du mécanisme.

Régulation et degrés de coopération. Nous avons déjà abordé la question des informations issues des actions de régulation auparavant. Toutefois, dans l'optique où nous pourrions obtenir des algorithmes justes dans ce domaine, les informations obtenues permettraient probablement de définir non pas une mais deux mesures de la coopération pour chacun des composants mécaniques : un degré de coopération et un degré de non coopération. En effet, les travaux les plus récents laissent à penser que, de manière générale, ces deux degrés permettent aux agents coopératifs de mieux décider des

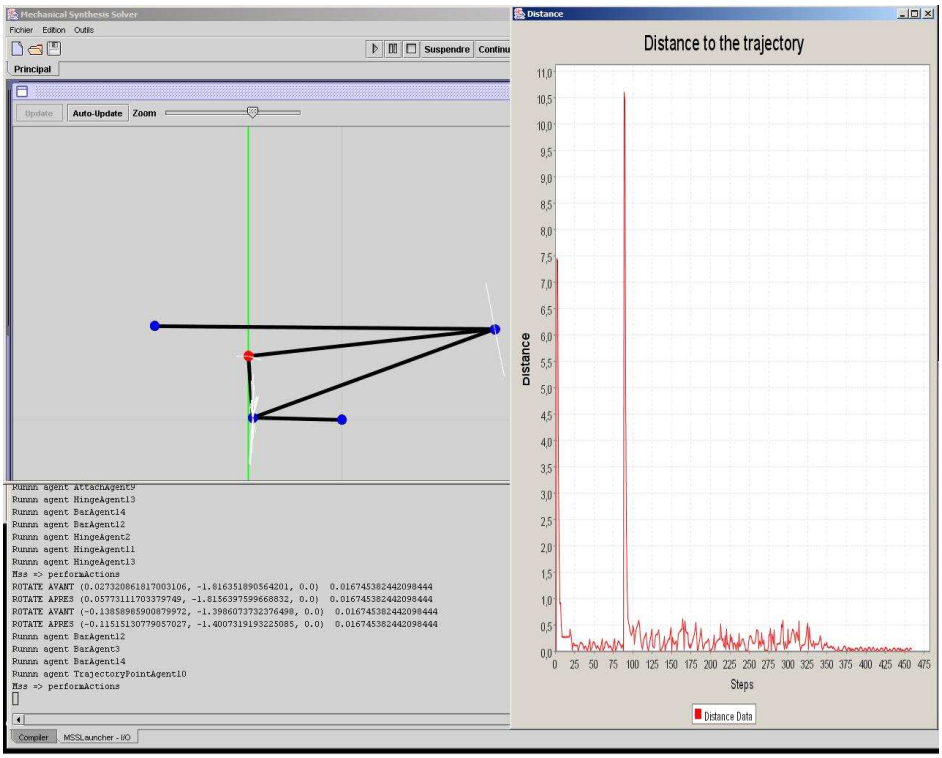

Figure 5.6 — Solution et mesure de l'erreur au cours de la résolution

actions à mettre en oeuvre (Georgé, 2004) (Picard, 2004).

Tout d'abord, ces deux degrés ne sont pas, comme pourrait le suggérer leur dénomination, inverses ou complémentaires. Le degré de non coopération correspond au degré de "gène" occasionné (ou pouvant être occasionné relativement à une action donnée) par un agent sur son voisinage et le degré de coopération à une évaluation des points positifs de son activité (ou de l'"aide" qu'il est susceptible apporter à son voisinage relativement à une action donnée). Ainsi, si l'on considère une évaluation de ces deux degrés sur un intervalle $[0,100]$ par exemple, des agents peuvent obtenir des couples de valeurs degré de coopération/degré de non coopération divers : $(0,100)$ pour un agent en fort conflit avec son environnement, $(100,0)$ pour un agent en totale adéquation avec son environnement, $(50,75)$ pour un agent ayant une activité coopérative avec certains agents et des interactions non coopératives avec d'autres, etc. Actuellement, l'application de la théorie des *AMAS* à divers problèmes, tend à montrer que contribuer à faire diminuer le degré de non coopération est plus important que de faire augmenter (ou même diminuer) le degré de coopération : dans (Georgé, 2004) J.P. Georgé nomme ce principe le "principe de moindre préjudice". L'idée intuitive est qu'il est plus important, pour un agent, de ne pas perturber le fonctionnement des autres agents du système plutôt que réussir à poursuivre sa propre activité à leur détriment.

L'application de ce principe pour les actions de réorganisation des agents mécaniques permet-

trait de guider le mécanisme vers des modifications plus judicieuses. Par exemple, sachant qu'il peut exister plusieurs topologies valides pour des mécanismes X-barres (pour $X > 5$), une telle analyse permettrait soit de passer de manière plus rapide d'une topologie valide à une autre, soit d'en éviter certaines qui ne correspondraient pas de toute manière avec la fonction que l'on attend du mécanisme.

Degrés de liberté. Parmi les situations non coopératives "add-hoc" que nous avons définies pour les agents du *MSS*, certaines sont en fait relatives aux degrés de libertés. Le blocage structurel en est le meilleur exemple : cette situation correspond à la détection d'un mécanisme sur-contraint, i.e. ayant trop peu de degrés de liberté. Le refus d'effectuer un lien avec certains agents parce que cela provoquerait un blocage est aussi une analyse, prospective dans ce cas là, de ce paramètre. Dans les mécanismes plan, les corps rigides ajoutent trois degrés de liberté (deux translations et une rotation) alors que les joints rotoïdes ou prismatiques en retirent deux (une charnière contraint les deux translations par exemple). L'ensemble de points d'attachement tels que nous les avons définis dans le *MSS* correspondent quand à eux un seul corps rigide qui ne possède aucun degré de liberté (il est "fixé au sol"). Pour finir, tout actuateur supprime encore des degré de liberté : lorsque l'on contraint la rotation d'une des charnières dans nos exemples, cela supprime un degré de liberté au mécanisme.

Raisonner systématiquement sur ces degrés pourrait fournir de nouvelles informations aux agents pour guider leurs choix. Ainsi, les composants sachant combien de degrés de liberté ils induisent ou suppriment, les agents éviteraient de manière simple les situations de blocage, des boucles inutiles ou encore des situations sous-contraintes, en détectant localement les degrés de liberté qu'il leur restent. De plus, l'analyse par le point trajectoire des mouvements erronés qu'il effectue par rapport à la trajectoire peut lui faire émettre une requête pour augmenter les degrés de libertés du mécanisme et ainsi favoriser l'apparition de trajectoires plus complexes.

Troisième partie

Modélisation des systèmes multi-agents basée sur les automates

6 Modèle basé sur les automates

L'objectif des travaux présentés ici est d'établir un modèle général pour les systèmes multi-agents. Ce travail a été réalisé en collaboration avec J. Fanchon du Laboratoire d'Analyse et d'Archictecture de Toulouse et s'intègre dans le projet nommé MASE[1]. Ce modèle repose sur une forme particulière de produit d'automates (Arnold, 1992) se situant au croisement des automates communiquants, des algèbres de processus (Milner, 1989) et de techniques de réétiquetage de graphes.

La modélisation d'un système d'agents et de ses comportements débute par la donné d'un ensemble I d'identificateurs d'agents, souvent défini par un intervalle d'entiers $I = [n] = \{1, ...n\}$. Les principes d'évolution du modèle reposent ensuite sur les choix suivants :

1. Chaque agent au cours d'un comportement se trouve dans un état particulier. L'état global du système est représenté par le vecteur des états des agents le constituant.

2. Les états de chaque agent sont étendus par les connaissances que peut avoir cet agent à propos des autres et de lui-même, ainsi que par une relation de voisinage. Ceci conduit à la définition de configurations locales.

3. Le comportement global du système est défini à partir des composants locaux : une configuration globale est un vecteur de configurations locales et l'on passe de l'une a l'autre au moyen de transitions compatibles avec les relations locales.

Nous décrirons tout d'abord la modélisation des agents à l'aide d'automates élémentaires, puis les notations relatives à l'ajout des connaissances et voisinages associés à chaque agent. La seconde partie décrira la manière dont est modélisé le système multi-agent dans son ensemble ainsi que sa dynamique. Nous aborderons la définition des règles de transition, outils permettant d'écrire plus facilement les nombreuses transitions possibles d'un système multi-agent. Pour finir, nous expliciterons comment peuvent être décrits les différents mécanismes mis en oeuvre dans les systèmes multi-agents.

[1]Modélisation par Automate de Systèmes multi-agents à Emergence. Projet développé dans le cadre du groupe de laboratoires FERIA. http ://www.feria.cnrs.fr)

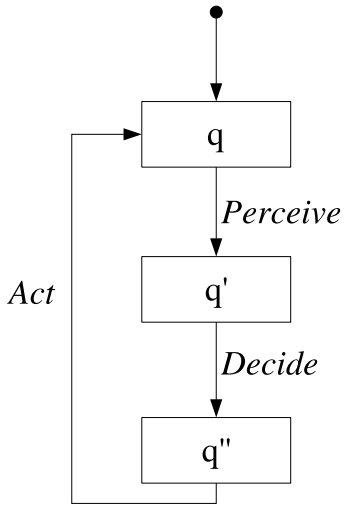

Figure 6.1 — Graphe d'états d'un agent

6.1 Composants élémentaires

6.1.1 Agents et automates

A partir de l'ensemble I d'identificateurs, une famille d'automates $A = \{A_i = (Q_{i,}\Sigma_i, \rightarrow_i), i \in I\}$ indexés par I représente l'ensemble des agents. Pour chaque automate A_i nous avons :

1. Q_i, un ensemble d'états élémentaires auxquels peuvent être associées un ensemble de variables Var_i.
2. Σ_i, un ensemble d'actions élémentaires (dont l'action nulle notée $*$).
3. $\rightarrow_i \subseteq Q_i \times \Sigma_i \times Q_i$, une relation de transition notée $q \xrightarrow{a}_i q'$ traduisant le passage de l'état q à q' par l'action élémentaire a. Sur chaque état est définie une transition "vide" $q \xrightarrow{*}_i q$ (application de l'action nulle).

L'adjonction d'un ensemble de variables aux états permet de simplifier le nombre, et donc l'écriture, des états. Par exemple, la définition d'un état *Attente* auquel on désire associer une valeur de durée comprise entre t_{min} et t_{max} peut s'écrire $Attente[t]$ (ceci évite la définition de multiples états $AttenteTMin$, $AttenteTMin + 1$, ..., $AttenteTMax$).

6.1.2 Automates "types"

Agent simple. A l'aide de ces automates élémentaires, il est aisé de définir un automate type basé sur le cycle de vie Perception-Décision-Action comme l'illustre la figure 6.1 sous forme de graphe d'états.

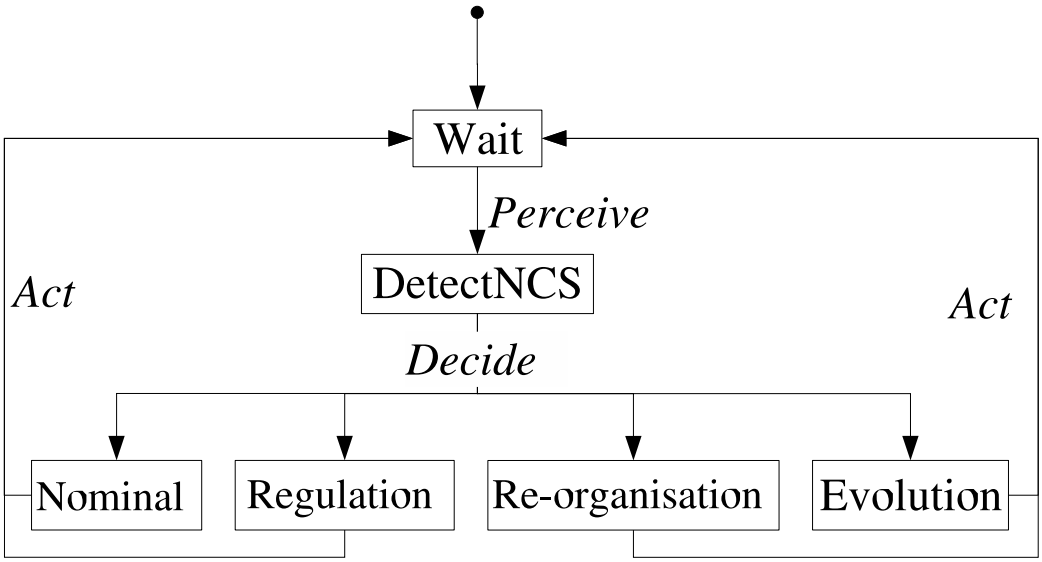

Figure 6.2 **—** Graphe d'états d'un agent coopératif

En haut du graphe, le point noir et la transition vers le premier état q (les états sont représentés par des rectangles) correspondent à la transition de création de l'agent. Les trois autres transitions sont étiquetées par les actions *Perceive*, *Decide* et *Act*.

Agent coopératif. En "dépliant" le graphe pour ajouter des états relatifs aux traitements des situations non coopératives, nous pouvons définir le graphe d'états générique d'un automate correspondant à l'activité d'un agent coopératif.

Sur la figure 6.2, sont donc ajoutés plusieurs états et transitions afin de détailler la partie décisionnelle d'un agent coopératif. Après avoir effectué une perception (action *Perceive*), l'agent se retrouve dans l'état *DetectNCS* d'analyse des situations non coopératives. Si aucune situation non coopérative n'est détectée, l'agent effectue la transition ($q_{DetectNCS} \xrightarrow{\text{Decide}} q_{Nominal}$) afin d'exécuter son comportement nominal. Dans le cas contraire, l'agent transite (selon le type de situation non coopérative détectée) vers l'état permettant de traiter la situation détectée (*Regulation*, *Re-organisation*, *Evolution*). Pour un problème donné, ce graphe peut être encore "déplié" afin de décrire plus précisément certains états.

L'environnement. Nous n'avons pas jusqu'ici abordé la question de l'environnement. Or ce dernier est souvent de première importance dans les systèmes multi-agents. Dans notre modèle, nous avons choisi de traiter l'environnement comme un agent particulier. L'agent A_0 (ou noté Env) permet ainsi d'intégrer l'environnement.

6.1.3 Configurations élémentaires

Pour chaque agent i, l'ensemble de ses configurations élémentaires (ou locales) est un ensemble $S_i \subseteq Q_i \times \Pi_{j \in I} R_i(j) \times 2^I$, dont les éléments, pour un agent i donné sont des triplets $s_i = (q_i, r_i[], v_i)$, composés comme suit :

1. $q_i \in Q_i$ est un état élémentaire ;
2. $r_i[]$ est un vecteur de vues locales, où pour tout $j \in I$ on a $r_i[j] \in R_i(j)$, l'ensemble $R_i(j)$ avec $\perp \in R_i(j)$ représentant les vues (connaissances, croyances) éventuellement vides ($\perp$) qu'a l'agent i de l'agent j ;
3. $v_i \subseteq I$ est un ensemble d'identificateurs, les voisins de i dans cette configuration.

Pour chaque agent (ou type d'agent) il est nécessaire de préciser les connaissances ou croyances qu'il peut avoir vis-à-vis des autres agents (ou types d'agents). Si l'on considère l'exemple d'un ensemble d'agents situés homogènes ayant un ensemble d'états q_i auxquels sont associées les coordonnées de leurs positions ($q[x,y]$), la définition de champs $coord_x$ et $coord_y$ aux vues locales des agents permettent d'exprimer la connaissance qu'ils peuvent avoir des positions des autres. Ainsi $r_i[j].coord_x$ contient la valeur x de la position de l'agent j pour l'agent i. $r_i[i].coord_x$ et $r_i[i].coord_y$ expriment la connaissance qu'a l'agent i de sa propre position. Ces connaissances peuvent ne pas correspondre à la réalité : on peut avoir $r_i[i].coord_x \neq q_i[x]$.

Connaissances exactes. Si l'on se donne deux ensembles Q et R, domaines de valeurs pour les états locaux et les vues, avec en particulier $\cup_{i \in I} Q_i \subseteq Q$, $\cup_{i,j \in I} R_i(j) \subseteq R$. Soit $I = [n]$, le domaine des configurations locales est $S = \cup_{i \in [n]} S_i \subseteq Q \times R^n \times 2^{[n]}$, chaque agent à un état local, une connaissance d'au plus n agents et au plus n voisins. Dans les cas où les agents ont des connaissances exactes sur les états des autres agents, les vues correspondent aux états élémentaires et l'on a $R_i(j) = Q_j \uplus \{\perp\}$ pour tout $i \in [n]$, et donc $S_i \subseteq Q \times \Pi_j (Q_j \cup \{\perp\}) \times 2^{[n]} \subseteq Q \times (Q \cup \{\perp\})^n \times 2^{[n]}$.

Restriction à un sous -ensemble d'agents. La restriction d'une configuration locale $s_i = (q_i, r_i[], v_i)$ à un sous-ensemble $J \subseteq I$ tel que $i \in J$ est le triplet $s_i/J = (q_i, r_i/J[], v_i \cap J)$ ou le vecteur de vues locales $r_i/J[j]$ n'est défini que pour les agents $j \in J$) avec $r_i/J[j] = r_i[j]$. Cette notation sera utile pour la définition des conditions d'application de règles que nous verrons dans la section 6.3.

6.2 Système, configurations et transitions globales

A partir des définitions ci-dessus, nous pouvons décrire un système multi-agent comme un produit particulier d'automates, l'ensemble des configurations globales de ce système étant le produit des configurations locales associées aux agents le composant. De plus, la modélisation de la dynamique du système peut être effectuée par l'écriture de transitions entre ces configurations globales.

6.2.1 Système et configurations globales

L'ensemble des configurations globales (ou états globaux) d'un système sur I est le produit $\Omega = \Pi_{i\in I} S_i$ des ensembles de configurations locales. Un élément $\overrightarrow{s} \in \Omega$ s'écrit $\overrightarrow{s} = (s_1, s_2....s_n)$ ou $|I| = n$.

En regroupant tous les premiers, deuxièmes et troisièmes champs des configurations locales $s_i = (q_i, r_i[], v_i)$, $\overrightarrow{s} \in \Omega$ peut également s'écrire $\overrightarrow{s} = (\overrightarrow{q}, \overrightarrow{r}, v)$ avec $\overrightarrow{q} = (q_1, q_2....q_n)$, $\overrightarrow{r} = (r_1[], r_2[]....r_n[])$, et $v : I \longmapsto 2^I$ associe à tout identificateur l'ensemble de ses voisins.

Ainsi, les configurations globales peuvent se définir par $\Omega = \Pi_{i\in I} Q_i \times (\Pi_{i\in I}\Pi_{j\neq i} R_j) \times V$ où $V = \{I \longmapsto 2^I\}$ est l'ensemble des relations de voisinage. Notons qu'une application $v : I \longmapsto 2^I$ définit un graphe orienté simple que l'on note $G_v = (I, v)$ ou simplement G lorsqu'il n'y a pas d'ambiguïté.

L'ensemble Ω représente ainsi tous les états possibles du système relativement aux définitions des agents qui le composent.

6.2.1.1 Transitions globales

Afin de pouvoir décrire la dynamique d'un système multi-agent il faut définir les transitions permettant de passer d'une configuration globale à une autre. Pour cela nous définissons *Act* l'ensemble $Act = \cup_{J\subseteq I} \prod_J \Sigma_i$ des actions globales. Un élément de *Act* est donc un vecteur d'actions élémentaires $(a_j)_{j\in J}$ pour un sous-ensemble $J \subseteq I$.

La relation de transition globale est une relation $\longrightarrow \subseteq \Omega \times Act \times \Omega$. Un triplet de cette relation est noté $(s_1, s_2....s_n) \xrightarrow{(a_j)_{j\in J}} (s'_1, s'_2....s'_n)$ où $J \subseteq I$ est appelé le sous-ensemble des agents interagissant ou actifs de la transition.

Une transition peut être également notée $(\overrightarrow{q}, \overrightarrow{r}, v) \xrightarrow{(a_j)_{j\in J}} (\overrightarrow{q'}, \overrightarrow{r'}, v')$ ou $\overrightarrow{s} \xrightarrow{a} \overrightarrow{s'}$ où $a = (a_i)_{i\in J}$.

Le graphe sous-jacent de la transition est le sous-graphe de G_v engendré par les sommets de J, $G_v/J = (J, v/J)$.

L'ensemble $\Omega \times Act \times \Omega$ désigne ici toutes les transitions que l'on pourrait écrire sans prise en compte des transitions définies au niveau des automates élémentaires.

Cohérence des transitions globales avec les transitions élémentaires. Seules les transitions compatibles avec les transitions élémentaires définies au niveau des agents doivent pouvoir être mises en oeuvre. Donc, pour une transition globale $(s_1, s_2....s_n) \xrightarrow{(a_j)_{j\in J}} (s'_1, s'_2....s'_n)$, on requiert que pour tout $i \in J$, si $s_i = (q_i, r_i[], v_i)$ et $s'_i = (q'_i, r'_i[], v'_i)$, il existe une transition locale $q_i \xrightarrow{a_i}_i q'_i$, et que $s_i = s'_i$ si $i \notin J$.

Soit $\Pi_{i\in I}A_i$ l'automate $\Pi_{i\in I}A_i = (\Pi_{i\in I}Q_i, Act, \longrightarrow_I)$, où la relation de transition produit $\longrightarrow_I$ est définie par $(q_1, q_2....q_n) \xrightarrow{(a_j)_{j\in J}}_I (q'_1, q'_2....q'_n)$ si et seulement si pour tout $i \in J$, il existe une transition locale $q_i \xrightarrow{a_i}_i q'_i$, et que $q_j = q'_j$ si $j \notin J$. Cet automate est appelé le produit libre des automates $(A_i)_{i\in I}$ et il ne contient, par définition, que des transitions compatibles avec les transition élémentaires.

De fait, la condition de cohérence s'exprime par : $(\overrightarrow{q}, \overrightarrow{r}, v) \xrightarrow{(a_j)_{j\in J}} (\overrightarrow{q'}, \overrightarrow{r'}, v')$ seulement si $\overrightarrow{q} \xrightarrow{(a_j)_{j\in J}}_I \overrightarrow{q'}$, i.e. la relation de transition globale est compatible avec la relation de transition du produit libre.

Projection d'une configuration sur un sous-ensemble d'agents. Soit $J = (i_1, i_2....i_m) \subseteq I$ un sous ensemble d'identificateurs, la projection d'une configuration $\overrightarrow{s} = (s_1, s_2....s_n)$ sur J est une configuration d'un système d'agents notée $\overrightarrow{s}/J = (s_{i_1}/J, s_{i_2}/J....s_{i_m}/J)$ où chaque configuration élémentaire s_{i_k}/J est la restriction de s_{i_k} à J définie en 2.1 ci-dessus.

Si l'on écrit $\overrightarrow{s} = (\overrightarrow{q}, \overrightarrow{r}, v)$ avec $\overrightarrow{q} = (q_1, q_2....q_n)$, $\overrightarrow{r} = (r_1[], r_2[]....r_n[])$, et $v : I \longmapsto 2^I$, alors $\overrightarrow{s}/J$ peut s'écrire $\overrightarrow{s}/J = (\overrightarrow{q}/J, \overrightarrow{r}/J, v/J)$ où $\overrightarrow{q}/J = (q_{i_1}, q_{i_2}....q_{i_m})$, $\overrightarrow{r}/J = (r_{i_1}/J[], r_{i_2}/J[]....r_{i_m}/J[])$, et $v/J : J \longmapsto 2^J$ est telle que pour tout $i \in J$ on a $v/J(i) = v(i) \cap J$.

6.3 Générateurs ou axiomes de la relation de transition

Le nombre de configurations locales étant généralement très élevé, la définition des transitions globales d'un système donné est difficilement réalisable. Nous allons donc définir ces relations de transition à partir d'un ensemble fini de règles (d'axiomes) qui génèrent, pour une configuration initiale donnée, l'ensemble des transitions et des configurations accessibles à partir de cette configuration initiale.

6.3.1 Règles

Une règle de taille k est un triplet de la forme $\theta = ((t_1, t_2....t_k), (b_1, b_2..., b_k), (t'_1, t'_2...t'_k)) \in S^k \times \Sigma^k \times S^k$. Cette règle s'applique à une configuration $(s_1, s_2....s_n) \in \Omega$, engendrant la transition globale

$$(s_1, s_2....s_n) \xrightarrow{(a_i)_{i\in J}}_\theta (s'_1, s'_2....s'_n)$$

s'il existe une injection $\varphi : [k] \longmapsto I$ telle que si $J = \varphi([k])$, pour tout $i \in [k]$, $t_i = s_{\varphi(i)}/J$, $t'_i = s'_{\varphi(i)}/J$, $t_i = s_{\varphi(i)}/J$, $b_i = a_{\varphi(i)}/J$.

On a de plus $s_{\varphi(i)}/(I-J) = s'_{\varphi(i)}/(I-J)$ et pour tout $i \in I-J$, $s_i = s'_i$.

6.3.2 Axiomes ou règles "localisées" pour les transitions.

On note $Q = \cup_{i\in I} Q_i$, $\Sigma = \cup_{i\in I}\Sigma_i$, et l'ensemble des vues des agents est noté R, avec $\cup_{i,j\in I} R_i(j) = R$.

Le domaine des configurations locales d'un système à n agents est noté $S(n) = Q \times R^n \times 2^{[n]}$.

Les règles de transition peuvent être définies par divers prédicats portant sur la configuration de départ d'une transition et/ou les actions exécutées, et/ou la configuration atteinte. Ces règles peuvent être de taille fixe ou de taille variable.

Règles de taille fixe. Soit k un entier, une règle de transition de taille k est une fonction

$$Trans_\theta : S(k)^k \times \Sigma^k \times S(k)^k \longrightarrow \{True, False\}$$

ou bien un couple $\theta = (Precond_\theta, Postcond_\theta)$ de fonctions

$$Precond_\theta : S(k)^k \times \Sigma^k \longrightarrow \{True, False\}$$

$$Postcond_\theta : S(k)^k \times S(k)^k \longrightarrow \{True, False\}$$

Règles de taille variable. On doit imaginer des règles pour lesquelles le nombre d'agents est une variable prenant valeur dans un ensemble donné (noté $\aleph$). Dans ce cas, les prédicats $Trans_\theta$ ou $(Precond_\theta, Postcond_\theta)$ peuvent également dépendre de la taille.

Une règle de transition de taille variable est une fonction

$$Trans_\theta : \cup_{k\in\aleph} S(k)^k \times \Sigma^k \times S(k)^k \longrightarrow \{True, False\}$$

ou bien un couple $\theta = (Precond_\theta, Postcond_\theta)$ de fonctions

$$Precond_\theta : \cup_{k\in\aleph} S(k)^k \times \Sigma^k \longrightarrow \{True, False\}$$

$$Postcond_\theta : \cup_{n\in\aleph} S(k)^k \times S(k)^k \longrightarrow \{True, False\}$$

6.3.3 Application des règles

Une règle $Trans_\theta$ s'applique à une configuration $(s_1, s_2 .... s_n) \in \Omega$ s'il existe une injection φ : $[k] \longmapsto I$ avec $J = \varphi([k])$ telle que $Trans_\theta((\overrightarrow{s}/J)_{i\in J}), (a_i)_{i\in J}, (\overrightarrow{s}'_i/J)_{i\in J}) = True$.

On dit que la règle θ s'applique a l'occurrence φ. Dans ce cas, la règle génère la transition globale

$$(s_1, s_2 .... s_n) \xrightarrow{(a_i)_{i\in J}}_\theta (s'_1, s'_2 .... s'_n)$$

où $s_j = s'_j$ si $j \notin J$.

Une règle $(Precond_\theta, Postcond_\theta)$ s'applique à une configuration $(s_1, s_2 .... s_n) \in \Omega$ s'il existe une injection φ : $[k] \longmapsto I$ avec $J = \varphi([k])$ telle que $Precond_\theta((\overrightarrow{s}_i/J)_{i\in J}, (a_i)_{i\in J}) = True$. La nouvelle configuration $(s'_1, s'_2 .... s'_n) \in \Omega$ étant définie par

$$Postcond_\theta((\overrightarrow{s}_i/J)_{i\in J}, (\overrightarrow{s}'_i/J)_{i\in J}) = True$$

.

Dans ce cas, on écrit $(s_1, s_2 .... s_n) \xrightarrow{(a_i)_{i\in J}}_\theta (s'_1, s'_2 .... s'_n)$ où $s_j = s'_j$ si $j \notin J$.

6.3.3.1 Types de conditions

Ces prédicats peuvent porter sur un ou plusieurs des trois champs d'une configuration $\overrightarrow{s} = (\overrightarrow{q}, \overrightarrow{r}, v)$, exprimer des contraintes sur les états, les connaissances ou la relation de voisinage. Par exemple on peut définir un prédicat

$$Precond_\theta : 2^{[k]\times[k]} \longrightarrow \{True, False\}$$

qui s'applique à des relations binaires entre ensembles ayant k éléments, et qui est vérifié dans la configuration $\overrightarrow{s} = (\overrightarrow{q}, \overrightarrow{r}, v)$ par l'injection φ : $[k] \longmapsto I$ si la relation de voisinage v restreinte à $\varphi([k])$ satisfait $Precond_\theta$, i.e. $Precond_\theta(v/_{\varphi([k])}) = True$.

6.3.3.2 Règles globales

On peut définir des règles θ qui s'appliquent directement à des configurations $(s_1, s_2 .... s_n) \in \Omega$ avec

$$Precond_\theta((s_i)_{i\in I}, (a_i)_{i\in I}) = True$$

une nouvelle configuration $(s'_1, s'_2 .... s'_n) \in \Omega$ étant définie par

$$Postcond_\theta((s_i)_{i\in I}, (s'_i)_{i\in I}) = True$$

Dans ce cas on écrit $(s_1, s_2....s_n) \xrightarrow{(a_i)_{i\in I}}_\theta (s'_1, s'_2....s'_n)$

On a donc $\overrightarrow{s} \xrightarrow{a}_\theta \overrightarrow{s'}$ si et seulement si

$$Precond_\theta(\overrightarrow{s}, a) \ = \ True$$

et

$$Postcond_\theta(\overrightarrow{s}, \overrightarrow{s'}) \ = \ True$$

6.3.4 Système engendré par un ensemble de règles

On se donne un ensemble de règles Θ. On note $\longrightarrow_\Theta$ la relation de transition globale $\longrightarrow_\Theta \subseteq \Omega \times Act \times \Omega$ définie par $\overrightarrow{s} \xrightarrow{a}_\Theta \overrightarrow{s'}$ si et seulement si $\exists \theta \in \Theta : \overrightarrow{s} \xrightarrow{a}_\theta \overrightarrow{s'}$.

On note $\overrightarrow{s} \longrightarrow_\Theta \overrightarrow{s'}$ si et seulement si $\exists a \in Act : \overrightarrow{s} \xrightarrow{a}_\Theta \overrightarrow{s'}$.

On note $\Theta(\overrightarrow{s})$ l'ensemble des configurations directement accessibles depuis $\overrightarrow{s}$: $\Theta(\overrightarrow{s}) = \{\overrightarrow{s'} \in \Omega : \overrightarrow{s} \longrightarrow_\Theta \overrightarrow{s'}\}$ et $\Theta^*(\overrightarrow{s})$ l'ensemble des configurations accessibles depuis $\overrightarrow{s}$ par transition successives.

Soit $X \subseteq \Omega$, les ensembles de configurations $\Theta^{i+1}(X) = \Theta(\Theta^i(X)) \cup \Theta^i(X)$ pour $i \in \aleph$ permettent de définir $\Theta^*(X) = \cup_{i\in\aleph} \Theta^i(X)$

Le système de transition engendré par l'ensemble de règles Θ , l'ensemble d'agents $\{A_i = (Q_i, \Sigma_i, \rightarrow_i), i \in I\}$ et la configuration initiale $\overrightarrow{s_{in}} \in \Pi_{i\in I} S_i$ est ainsi définie comme l'ensemble de configurations $\Theta^*(\overrightarrow{s_{in}})$.

6.4 Un modèle pour les SMA

Le modèle présenté ci-dessus permet d'exprimer nombre de mécanismes et propriétés des systèmes multi-agents. Notre but n'est pas de modéliser une classe particulière de systèmes mais bel et bien un spectre le plus large possible.

6.4.1 Les composants essentiels

Pour tout système multi-agent, il doit être possible d'exprimer le comportement des agents puis de disposer des outils et notations permettant leur manipulation notamment en ce qui concerne leur dynamique ainsi que celle du système dans sa totalité. Ayant construit le modèle à partir de ces considérations, les définitions et mécanismes élémentaires s'expriment de manière simple.

Un système multi-agent *SMA* se définit par $SMA = (I, \{A_i\}, R, \Theta, \Gamma)$ où I est l'ensemble d'identificateurs d'agents composant le système (excepté l'environnement noté Env qui est assimilé à l'identificateur 0), $\{A_i\}$ définit la famille d'automates associée, R le domaine des connaissances, Θ les axiomes ou règles de transitions et Γ un prédicat de cohérence.

Les éléments I, $\{A_i\}$, R et Θ ont été explicités dans les sections précédentes et permettent de spécifier les agents et leurs comportements.

Le prédicat Γ permet la définition d'un prédicat global afin de restreindre les configurations globales accessibles ou de modifier les contraintes d'applications des règles. L'intérêt de Γ est principalement de simplifier l'expression de certaines propriétés dont on veut munir a priori le système. Par exemple, il existe plusieurs manières d'aborder la modélisation de lois physiques telles que "deux entités solides ne peuvent partager un espace commun". Or, si l'on s'intéresse à un système composé d'agents situés comme des robots dans un environnement physique réel, le modèle doit pouvoir intégrer les lois permettant d'assurer que deux robots ne peuvent pas être au même endroit au même moment (détection de collisions etc.). La mise en place de règles physiques au niveau des agents eux-mêmes (des automates donc) est une approche possible mais n'est pas valable dans le cas où les règles s'appliquent de manière concurrente/asynchrone. En utilisant Γ, cela se réduit à la définition d'un prédicat restreignant les configurations globales satisfaisantes aux seules configurations dans lesquelles il n'y a pas deux (ou plus) robots au même endroit.

6.4.2 Les interactions

Les interactions sont simplement exprimables à l'aide de transitions dans lesquelles deux (ou plus) entités sont actives. Si l'on considère la modélisation d'un échange synchrone de messages entre deux agents i et j, cela s'écrit de manière générale sous la forme d'une règle de taille fixe

$$(q_i, r_i[], v_i), (q_j, r_j[], v_j) \xrightarrow{\text{(Send, Receive)}} (q'_i, r'_i[], v'_i), (q'_j, r'_j[], v'_j)$$

avec la (ou les) fonction booléenne spécifique à cet échange en particulier. Cette règle pourra s'appliquer à tout couple (i, j) respectant les préconditions et postconditions de la règle.

Les communications indirectes (dépôt de phéromones, *Blackboard* etc.) se font naturellement par le biais de règles où intervient l'environnement. Une fourmi représentée par un automate identifié par i effectuera un dépôt de phéromone dans l'environnement par une règle de taille fixe du type

$$(q_{Env}, r_{Env}[], v_{Env}), (q_i, r_i[], v_i) \xrightarrow{\text{(Deposit)}} (q'_{Env}, r'_{Env}[], v_{Env}), (q'_i, r'_i[], v_i)$$

L'utilisation de règles à taille variable permet d'exprimer les interactions dont le "champ d'application" est dépendant du contexte : par exemple, la diffusion d'un message au seul voisinage d'un

agent i donné (chuchotement) sera définie par une règle de taille variable avec une condition portant sur le voisinage v_i (les seuls agents actifs de la transition générée par cette règle seront l'agent i et tous les agents j tels que $j \in v_i$).

6.4.3 Application concurrente ou distribuée des règles

De nombreux systèmes multi-agents, peuvent nécessiter l'application de plusieurs règles lors d'une même transition. Cela permet de modéliser l'évolution asynchrone des parties d'un système. Il est dès lors intéressant de pouvoir étudier la compatibilité des transitions dans ce cadre.

Soit un système de transitions défini à partir d'une configuration initiale donnée :

$$ST = (\Omega, Act, \longrightarrow_{\Theta}, \overrightarrow{s}_{in})$$

Considérons deux règles θ et θ' de taille repective k et k', qui s'appliquent aux occurrences φ et φ' d'une configuration $(s_1, s_2 .... s_n) \in \Omega$: il existe deux injections $\varphi : [k] \longmapsto I$ avec $J = \varphi([k])$ et $\varphi' : [k'] \longmapsto I$ avec $J' = \varphi'([k'])$ telles que

$$Precond_{\theta}((s_i/J)_{i \in J}) = Precond_{\theta'}((s_i/J')_{i \in J'}) = True$$

Si J et J' sont disjoints, le système peut effectuer les transitions induites par θ et θ' de façon indépendante et concurrente. Par contre, si $J \cap J' \neq \emptyset$, les transitions peuvent être incompatibles sur les agents $i \in J \cap J'$.

6.4.4 Notion d'implémentation, de simulation

Un système de transition $ST = (\Omega, Act, \longrightarrow, \overrightarrow{s}_{in})$ implémente le système de transition $ST' = (\Omega', Act', \longrightarrow', \overrightarrow{s}'_{in})$ s'il existe un couple d'applications (σ, ρ) : $\sigma : \Omega \longrightarrow \Omega'$, $\rho : Act \longrightarrow Act'$ tel que

- $\sigma(\overrightarrow{s}_{in}) = \overrightarrow{s}'_{in}$ et
- Si $\rho(a)$ n'est pas l'action globale vide : $\overrightarrow{s}_0 \xrightarrow{a} \overrightarrow{s}_1 \Longrightarrow \sigma(\overrightarrow{s}_0) \xrightarrow{\rho(a)} \sigma(\overrightarrow{s}_1)$
  Si $\rho(a)$ est l'action globale vide : $\overrightarrow{s}_0 \xrightarrow{a} \overrightarrow{s}_1 \Longrightarrow \sigma(\overrightarrow{s}_0) = \sigma(\overrightarrow{s}_1)$.

On remarquera qu'il peut exister de nombreuses implémentations différentes d'un même système. L'intérêt est ici de fournir un moyen simple d'établir que deux systèmes de transition permettent la description d'un même système mais selon une "précision" et, parfois, des processus différents. Ce point peut être d'importance pour ce qui est de la vérification de l'adéquation d'un système multi-agent proposé pour résoudre un problème donné : on peut prouver l'existence d'un couple (σ, ρ) mettant en évidence que le modèle du système implémente bien celui du problème.

6.5 Concepts et notions exprimables dans le modèle

Une multitude de concepts et notions sont utilisés dans le domaine des systèmes multi-agents. Au stade d'avancement actuel de nos travaux, nous n'avons pas abordé tous ces aspects de manière exhaustives. Toutefois, nous pouvons avancer quelques éléments (parfois informels) quant à l'expression de certains d'entre eux.

Environnement. Comme nous l'avons dit plus haut, le choix a été fait de modéliser l'environnement par un automate particulier du système. Pour certaines applications, dans lesquelles l'environnement doit être distribué, rien ne s'oppose à traduire cette distribution par l'association de plusieurs automates aux différentes "parties" de l'environnement. L'accessibilité de ces "parties" pour les agents se traduisant par le biais des voisinages. La dynamicité d'un environnement peut être identifiée à l'existence de transitions dans lesquelles **seul** l'agent (ou les agents) correspondant à l'environnement est modifié (ou des règles pour lesquelles $Trans_\theta$ porte exclusivement sur les configurations locales de Env).

Autonomie. La question de l'autonomie est délicate à traiter dans notre modèle. En effet, juger qu'un système (ou un agent) est autonome ou non peut être fortement subjectif. L'autonomie de décision d'un agent pourrait se traduire par le fait que nulle transition élémentaire étiquetable *Decide* (après "repliage" pour revenir au modèle simple d'agent proposé en début de section) n'est mise en oeuvre par des règles dont les conditions portent sur d'autres configurations locales que celle de l'agent considéré.

Réorganisation, auto-organisation. Les processus de réorganisation se traduisent, dans notre modèle, par l'application de transitions modifiant les voisinages. L'analyse des processus de décision présidant au déclenchement des modification de voisinage devrait permettre de pouvoir distinguer les systèmes auto-organisateurs et auto-organisés.

Agent cognitif, agent réactif. Si ces deux types d'agent peuvent être décrits dans le modèle, pouvoir identifier que l'automate associé à un agent décrit une architecture réactive ou cognitive (délibérative) pourrait se faire par la définition d'opérations d'abstraction (ou "repliage") pour ramener le graphe d'états d'un automate à des graphes d'états caractéristiques d'architectures spécifiques.

Coopération. La coopération dans le cadre des systèmes multi-agents adaptatifs sera, à court terme, un centre d'intérêt majeur dans les travaux à venir. Cette étude nécessitera l'association d'une fonction d'évaluation de la coopération (et/ou de la non coopération) au niveau des configurations locales (en se basant sur les connaissances $r_i[]$ du triplet) pour la notion d'agents coopératifs et au niveau du système multi-agent (donc de l'ensemble des configurations locales associées

à des agents, i.e. environnement exclu) pour la notion de système à milieu intérieur coopératif. L'adéquation fonctionnelle serait mesurable grâce à l'évaluation des configurations globales (celles-ci contiennent l'environnement et le système multi-agent). La définition d'un système coopératif (et pas seulement d'un système à milieu intérieur coopératif) devra passer par une projection sur les seules configurations locales ayant l'environnement dans leur voisinage et une évaluation de la coopération de ces configurations locales uniquement relative à l'environnement (uniquement l'évaluation de "la frontière" SMA-Environnement donc).

Émergence. L'étude de l'émergence peut porter sur la détection de propriétés spécifiques des états du système (portant donc sur les configurations globales) et sur l'analyse de la dynamique d'apparition de ces phénomènes en s'intéressant aux comportements (suites de transitions) permettant de les atteindre. La définition de propriétés (relatives à la définition pratique de l'émergence donnée dans la partie 1.1.2.5) portant sur les règles régissant la dynamique d'un système pourrait aussi fournir un moyen de dire s'il est émergent ou non.

Fiabilité, robustesse, convergence. Ces notions sont définissables par des propriétés portant sur les configurations accessibles et les séquences de transitions y conduisant. L'utilisation de logiques modales, temporelles, etc. dans ce cadre pourraient permettre l'utilisation d'outils de preuves automatiques existants.

Interopérabilité, adaptativité. On peut définir de tels systèmes avec le modèle mais on ne peut probablement pas décrire ces propriétés.

Ouverture/fermeture. Le modèle ne permet pas, en l'état, d'exprimer l'ajout ou le retrait d'agents d'une manière reflétant fidèlement le processus. Cependant, en ajoutant un état *Out* au graphe des automates, il est possible de simuler ces mécanismes : lorsqu'un automate est inactif, on le considère comme n'appartenant pas au système, quitter cet état particulier correspondant à la création de l'agent et y rentrer à son retrait du système. La contrainte forte étant qu'il est nécessaire de maintenir ces automates "fantômes" dans le modèle même lorsqu'ils ne servent à rien (et il faut en prévoir assez "en réserve"). Cependant, du fait de l'utilisation de règles basées sur des conditions, la seule adjonction systématique de tests excluant les automates *Out* suffit.

7 Exemple d'application du modèle au *MSS*

LE modèle décrit dans le chapitre précédent permettant de modéliser des systèmes multi-agent, nous l'avons utilisé, à titre d'exemple, pour modéliser le *Mechanical Synthesis Solver*.

7.1 Représentation de l'environnement

L'environnement du *MSS* peut être modélisé de manière simple car, dans l'implémentation que nous en avons fait, peu d'éléments sont pris en compte. Du fait de sa faible dynamicité, le graphe d'état associé à son automate (identificateur 0 ou Env) peut se réduire à un seul état q. A cet unique état sont associées les variables correspondant aux objets ayant une importance du point de vue du mécanisme ($trajectories[n]$ qui est un tableau contenant les trajectoires) ou du point de vue des agents ($messages[]$ qui contient les messages de la boîte aux lettres utilisée par les agents pour communiquer). Chaque message contient le numéro de l'envoyeur (*sender*), celui du destinataire (*recipient*) et les différents champs contenant les données pertinentes à communiquer. Pour simplifier, nous ne décrirons pas ici tous les types de messages et tous les champs de données associés.

La seule connaissance d'importance pour l'environnement est la connaissance du nombre de messages contenu dans la boîte aux lettres (*nbMessages* à valeurs dans N^+).

7.2 Représentation des agents

7.2.1 Automate des agents mécaniques

A partir du graphe général d'état d'agent coopératif présenté dans le modèle, on peut en définir une instance pour les agents mécaniques (figure 7.1).

Ensemble d'états des agents :

q_{Out} L'agent ne participe pas à l'activité du système (considéré comme non créé).

q_{Wait} L'agent est en attente de réception d'un message.

$q_{DetectNcs}$ L'agent détermine si des situations non coopératives sont détectées.

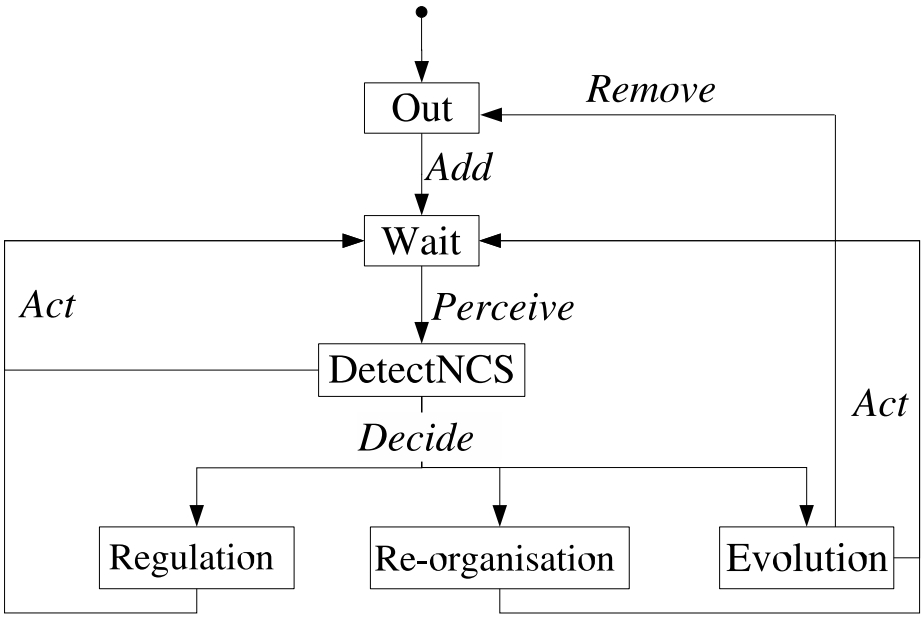

Figure 7.1 **—** Graphe d'états d'un agent mécanique

$q_{Adjustment}$ L'agent calcule l'action la plus adéquate en terme de modification de position, taille, etc.

$q_{Re-organisation}$ L'agent effectue des actions de réorganisation (*Link*, *Unlink*).

$q_{Evolution}$ L'agent introduit de nouveaux agents dans le système ou bien disparaît.

Ne mettant pas en oeuvre leurs compétences par eux-mêmes (la cinématique est calculée par MECANO), les agents mécaniques n'ont pas de comportement nominal : leur activité se réduit au traitement des situations non coopératives.

A chacun de ces états sont associés des variables représentant les caractéristiques de chaque agent. A cette fin, un ensemble $NOEUD = (R \times R) \cup \{null\}$ est préalablement défini pour modéliser les noeuds.

Agent barre Deux variables ($noeud_1$ et $noeud_2$) à valeurs dans $NOEUD$ représentant les points d'attache du composant à chaque extrémité de la barre.

Agent charnière Deux variables ($noeud_1$ et $noeud_2$) à valeurs dans $NOEUD$ représentant les deux points d'attache de l'agent.

Agent point trajectoire Une variable (*noeud*) à valeurs dans $NOEUD$ représentant le point.

Agent point d'attachement Une variable (*noeud*) à valeurs dans $NOEUD$ représentant le point d'attache de l'agent.

7.2.2 Description des connaissances des agents

Chaque agent possède des connaissances spécifiques relatives à son type et à son activité. Certains champs de connaissance sont communs à tous les agents : par exemple, la connaissance du type d'un agent ($r_i[].type$ à valeurs dans { *Bar*, *Hinge*, *TrajectoryPoint*, *AttachmentPoint* }).

Les connaissances spécifiques aux différents agents sont :

Agent barre Tout agent barre maintient la connaissance des agents avec lesquels il est attaché : $attach_1$ et $attach_2$ sont les deux champs référençant l'identificateurs des agents voisins pour chaque extrémité (valeurs dans $I \cup \{null\}$). $support$ contient l'identificateur de l'agent point trajectoire attaché sur la barre s'il y a lieu. Afin de traiter les situations non coopératives, il doit aussi enregistrer les erreurs qu'il reçoit de ses voisins, soient les facteurs de modification d'amplitude ($r_i[].amplitudeError_1$ et $r_i[].amplitudeError_2$ à valeurs dans R) et les angles de déviations (($r_i[].angleError_1$ et $r_i[].angleError_2$ à valeurs dans $[-\pi, \pi]$)). L'agent prend aussi en compte les déplacement effectués par chacune de ses extrémités lors du pas de simulation précédent ($r_i[].move_1$ et $r_i[].move_2$ à valeurs dans $R \times R$).

Agent charnière Tout comme les agents barre, ils ont la connaissances des agents auxquels ils sont attachés ($attach_1$ et $attach_2 \in I \cup \{null\}$). Pour la gestion des ajustements, ils disposent d'une mémoire coopérative qui stocke les erreurs reçues au cours du temps ($r_i[].memoryCoop \in 2^{R \times R}$).

Agent point trajectoire Le point trajectoire dispose de la connaissance de l'identité de la barre sur laquelle il se trouve ($r_i[].attach$ à valeurs dans $I \cup \{null\}$) et d'une mémoire coopérative pour le traitement des erreurs de position ($r_i[].memoryCoop \in 2^{R \times R}$). Il a aussi connaissance de la trajectoire qu'il doit suivre ($r_i[].trajectory$ à valeur dans N) et de la distance qui l'en sépare ($r_i[].distance$ à valeur dans $R \times \mathbb{R}$ i.e. un vecteur).

Agent point d'attachement Les points d'attachement n'ont que la connaissance de l'agent auquel ils sont liés ($r_i[].attach$ à valeurs dans $I \cup \{null\}$).

7.3 Règles de transition

Nous allons décrire dans cette section les différentes règles permettant de modéliser la dynamique du système.

La première règle est une règle globale (elle s'applique à tout le système) et concerne la mise en oeuvre du calcul de la cinématique du mécanisme par *MECANO*. Cette simulation est effectuée lorsque tous les agents sont dans l'états q_{Wait} et qu'il n'y a aucun message dans la boîte aux lettres.

$$(s_{Env}, s_1, \ldots, s_k, \ldots, s_n) \xrightarrow{MECANO} (s'_{Env}, s_1, \ldots, s_k, \ldots, s_n)$$

Précondition :

$$(r_{Env}[Env].NbMessage = 0) \wedge$$
$$(\forall k \neq Env,\ q_k = q_{Wait})$$

Postcondition :

$$(q_{Env}[messages[]] \neq q'_{Env}[messages[]]) \wedge$$
$$(r'_{Env}[Env].NbMessage > 0)$$

Après application de cette règle aucune configuration locale n'a été modifiée sauf celle de l'environnement puisque la boîte aux lettres ontient maintenant tous les messages de mise à jour pour les agents.

Il faut ensuite, pour chacune des transitions apparaissant sur le graphe d'état des agents mécaniques, expliciter les différentes actions mises en oeuvre par les agents. Dans chacune des règles décrite ci-dessous, l'agent *i* est considéré comme l'agent actif.

7.3.1 $q_{Wait} \rightarrow q_{DetectNcs}$

Cette transition correspond à la réception d'un message par l'agent.

. Réception d'un message *UpdateData* (après une simulation du mécanisme par MECANO) contenant les nouvelles coordonnées d'un noeud de l'agent le recevant. Cette règle s'applique aux seuls agents charnière.

$$s_{Env}, s_i = (q_{Wait}, r_i, v_i) \rightarrow s'_{Env}, s'_i = (q_{DetectNcs}, r'_i, v_i)$$

Précondition :

$$(r_i[i].type = Hinge) \wedge$$
$$(r_{Env}[Env].nbMessages \neq 0) \wedge$$
$$q_{Env}[messages[0].recipient = i]$$

Postcondition :

$$(r'_{Env}[Env].nbMessage = r_{Env}[Env].nbMessage - 1) \wedge$$
$$(q'_{Env}[messages[0]] \neq q_{Env}[messages[0]]) \wedge$$
$$(q_{Wait}[noeud_k] \neq q_{DetectNcs}[noeud_k]) \wedge$$
$$(r_i[i].memory \neq r'_i[i].memory)$$

Cette règle spécifie que si le premier message de la boîte aux lettres est destiné à un agent, cet agent le consomme et modifie ses caractéristiques (position du noeud $noeud_k$ dont les nouvelles coordonnées sont données par le message). La mémoire est modifiée pour intégrer l'"oubli" par une réduction de tous les vecteurs d'un certain pourcentage.

. Réception d'un message *UpdateData* (après une simulation) contenant les nouvelles coordonnées du noeud de l'agent le recevant. Cas des agents de type point trajectoire :

$$s_{Env}, s_i \;=\; (q_{Wait}, r_i, v_i) \;\rightarrow\; s'_{Env}, s'_i \;=\; (q_{DetectNcs}, r'_i, v_i)$$

Précondition :

$(r_i[i].type \;= TrajectoryPoint) \wedge$

$(r_{Env}[Env].nbMessages \;!= 0) \wedge$

$q_{Env}[messages[0].recipient \;=\; i]$

Postcondition :

$(r'_{Env}[Env].nbMessage \;=\; r_{Env}[Env].nbMessage - 1) \wedge$

$(q'_{Env}[messages[0]] \;\neq\; q_{Env}[messages[0]]) \wedge$

$(q_{Wait}[noeud] \;\neq\; q_{DetectNcs}[noeud]) \wedge$

$(r_i[i].memory \;\neq\; r'_i[i].memory)$

Si dans la boîte aux lettres, le prochain un message est destiné à l'agent *point trajectoire i*, ce dernier le consomme, modifie les valeurs de sa caractéristique *noeud* et met à jour sa mémoire coopérative pour intégrer la nouvelle distance entre l'agent et la trajectoire ainsi que l'"oubli".

. Réception d'un message *UpdateData* (après une simulation) contenant les nouvelles coordonnées du noeud de l'agent le recevant. Cas des agents de type point d'attachement :

$$s_{Env}, s_i \;=\; (q_{Wait}, r_i, v_i) \;\rightarrow\; s'_{Env}, s'_i \;=\; (q_{DetectNcs}, r'_i, v_i)$$

Précondition :

$(r_i[i].type \;= AttachmentPoint) \wedge$

$(r_{Env}[Env].nbMessages \;!= 0) \wedge$

$q_{Env}[messages[0].recipient \;=\; i]$

Postcondition :

$(r'_{Env}[Env].nbMessage \;=\; r_{Env}[Env].nbMessage - 1) \wedge$

$(q'_{Env}[messages[0]] \;\neq\; q_{Env}[messages[0]]) \wedge$

$(q_{Wait}[noeud] \;\neq\; q_{DetectNcs}[noeud])$

L'agent *point d'attachement* lit le message dans la boîte aux lettres et met à jour sa position.

. Réception d'un message *UpdateData* (après une simulation) contenant les nouvelles coordonnées du noeud de l'agent le recevant. Cas des agents de type barre :

$$s_{Env}, s_i \;=\; (q_{Wait}, r_i, v_i) \;\rightarrow\; s'_{Env}, s'_i \;=\; (q_{DetectNcs}, r'_i, v_i)$$

Précondition :

$(r_i[i].type = TrajectoryPoint) \wedge$

$(r_{Env}[Env].nbMessages\ != 0) \wedge$

$q_{Env}[messages[0].recipient = i]$

Postcondition :

$(r'_{Env}[Env].nbMessage = r_{Env}[Env].nbMessage - 1) \wedge$

$(q'_{Env}[messages[0]] \neq q_{Env}[messages[0]]) \wedge$

$(q_{Wait}[noeud_k] \neq q_{DetectNcs}[noeud_k]) \wedge$

$(r'_i.move_k = \overrightarrow{(q_{Wait}[noeud_k], q_{DetectNcs}[noeud_k])})$

Si le prochain message est destiné à l'agent *barre i*, ce dernier le lit et met à jour la position de l'extrémité concernée *k*. L'agent met aussi à jour la connaissance relative au déplacement de cette extrémité ($move_k$).

. Réception d'un message *Incompetence* par un agent barre.

$$s_i = (q_{Wait}, r_i, v_i) \rightarrow s'_i = (q_{DetectNcs}, r'_i, v_i)$$

Précondition :

$(r_i[i].type = Bar) \wedge$

$(r_{Env}[Env].nbMessages \neq 0) \wedge$

$q_{Env}[messages[0].recipient = i]$

Postcondition :

$(r'_{Env}[Env].nbMessage = r_{Env}[Env].nbMessage - 1) \wedge$

$(q'_{Env}[messages[0]] \neq q_{Env}[messages[0]]) \wedge$

$((r_i(i).amplitudeError_1 \neq r'_i(i).amplitudeError_1) \vee$

$(r_i(i).amplitudeError_2 \neq r'_i(i).amplitudeError_2)) \wedge$

$((r_i(i).angleError_1 \neq r'_i(i).angleError_1) \vee$

$(r_i(i).angleError_2 \neq r'_i(i).angleError_2))$

Lorsqu'un agent *barre i* reçoit un message *Incompetence*, il met à jour les champs relatifs aux erreurs contenues dans le message. Le message venant de l'une ou l'autre des deux charnières voisines.

. Réception d'un message *IncompetenceAdjustment* par un agent charnière. Le message contient le déplacement (considéré comme une erreur par la charnière) demandé par une barre voisine.

$$s_i = (q_{Wait}, r_i, v_i) \rightarrow s'_i = (q_{DetectNcs}, r'_i, v_i)$$

Précondition :

$(r_i[i].type = Hinge) \wedge$

$(r_{Env}[Env].nbMessages \neq 0) \wedge$

$q_{Env}[messages[0].recipient = i]$

Postcondition :

$(r'_{Env}[Env].nbMessage = r_{Env}[Env].nbMessage - 1) \wedge$

$(q'_{Env}[messages[0]] \neq q_{Env}[messages[0]]) \wedge$

$(r_i(i).memory \neq r'_i(i).memory)$

L'agent charnière modifie sa mémoire coopérative en ajoutant la nouvelle erreur transmise par le message.

. Réception d'un message *Incompetence* par un agent charnière. Le message est à destination de la barre voisine qui n'est pas à l'origine du message.

$$s_i = (q_{Wait}, r_i, v_i) \rightarrow s'_i = (q_{DetectNcs}, r_i, v_i)$$

Précondition :

$(r_i[i].type = Hinge) \wedge$

$(r_{Env}[Env].nbMessages \neq 0) \wedge$

$(q_{Env}[messages[0].recipient = i]) \wedge$

$(q_{Env}[messages[0].sender = r_i[i]].attach_k)$

Postcondition :

$(r'_{Env}[Env].nbMessage = r_{Env}[Env].nbMessage) \wedge$

$(q'_{Env}[messages[]] \neq q_{Env}[messages[]])$

Si l'agent ayant envoyé le message est l'agent référencé comme la barre attachée à la première extrémité ($k = 1$) le message sera déposé pour l'agent $r_i[i]].attach_2$ et inversement si ($k = 2$).

7.3.2 $q_{DetectNcs} \rightarrow q_{Wait}$

S'il n'y a aucune situation non coopérative à traiter, l'agent retourne dans l'état q_{Wait}.

. Cas d'un agent barre.

$$s_i = (q_{DetectNcs}, r_i, v_i) \rightarrow s'_i = (q_{Wait}, r_i, v_i)$$

Précondition :

$(r_i[i].type = Bar) \wedge$

$(r_i(i).amplitudeError_1 = 0.0) \wedge$

$(r_i(i).amplitudeError_2 = 0.0) \wedge$

$(r_i(i).angleError_1 = 0.0) \wedge$

$(r_i(i).angleError_2 = 0.0)$

Postcondition :

. Cas d'un agent *charnière* ou *point trajectoire*.

$$s_i = (q_{DetectNcs}, r_i, v_i) \rightarrow s'_i = (q_{Wait}, r_i, v_i)$$

Précondition :

$((r_i[i].type = Hinge) \vee$
$(r_i[i].type = TrajectoryPoint)) \wedge$
$(r_i(i).memory = \{\}) \wedge$

Postcondition :

7.3.3 $q_{DetectNcs} \rightarrow q_{Regulation}$

Si l'agent détecte une situation d'incompétence, il peut passer dans l'état $q_{Regulation}$ pour résoudre le problème.

. Cas d'un agent *point trajectoire* ou *charnière* : la mémoire coopérative n'est pas vide et elle n'est pas dans un état stable.

$$s_i = (q_{DetectNcs}, r_i, v_i) \rightarrow s'_i = (q_{Regulation}, r_i, v_i)$$

Précondition :

$((r_i[i].type = Hinge) \vee$
$(r_i[i].type = TrajectoryPoint)) \wedge$
$(r_i[i].memory \neq \{\}) \wedge$
$(\neg isStable(r_i[i].memory)$

Postcondition :

La fonction booléenne $isStable()$ correspond aux calculs effectués sur la mémoire coopérative pour déterminer si elle a atteint un état stable (les ajustements proposés sont négligeables par rapport aux ajustements effectués lors des derniers pas).

. Pour un agent *barre*.

$$s_i = (q_{DetectNcs}, r_i, v_i) \rightarrow s'_i = (q_{Adjustment}, r_i, v_i)$$

Précondition :

$(r_i[i].type = Bar) \wedge$
$((r_i(i).amplitudeError_1 \neq 0.0) \vee$
$(r_i(i).amplitudeError_2 \neq 0.0)) \vee$
$((r_i(i).angleError_1 \neq 0.0) \vee$
$(r_i(i).angleError_2 \neq 0.0))$

Postcondition :

7.3.4 $q_{DetectNcs} \rightarrow q_{Re-organisation}$

Si un agent détecte une situation non coopérative d'inutilité il passe dans l'état $q_{Re-organisation}$.

. Pour le *point trajectoire* : s'il n'est pas attaché à un agent *barre*.

$$s_i = (q_{DetectNcs}, r_i, v_i) \rightarrow s'_i = (q_{Re-organisation}, r_i, v_i)$$

Précondition :

$(r_i[i].type = TrajectoryPoint)) \wedge$

$(r_i[i].attach = null)$

Postcondition :

. Pour un agent *barre* : s'il n'est attaché à aucun agent.

$$s_i = (q_{DetectNcs}, r_i, v_i) \rightarrow s'_i = (q_{Re-organisation}, r_i, v_i)$$

Précondition :

$(r_i[i].type = Bar) \wedge$

$(r_i[i].attach_1 = null) \wedge$

$(r_i[i].attach_2 = null)$

Postcondition :

. Pour un agent *charnière* : s'il n'est pas attaché à deux agents.

$$s_i = (q_{DetectNcs}, r_i, v_i) \rightarrow s'_i = (q_{Re-organisation}, r_i, v_i)$$

Précondition :

$(r_i[i].type = Hinge) \wedge$

$((r_i[i].attach_1 \neq null) \vee$

$(r_i[i].attach_2 \neq null))$

Postcondition :

Les agents charnière et point trajectoire détectent une situation non coopérative nécessitant une réorganisation lorsque leur mémoire coopérative se considère dans un état stable.

7.3.5 $q_{Regulation} \rightarrow q_{Wait}$

Lorsqu'un agent est dans l'état $q_{Regulation}$ il met en oeuvre une action de correction et retourne dans l'état q_{Wait}.

. Cas du point trajectoire.

$$s_{Env}, s_i \;=\; (q_{Regulation}, r_i, V_i) \xrightarrow{ChangePosition} s'_{Env}, s'_i \;=\; (q_{Wait}, r'_i, V_i)$$

Précondition :

$r_i[i].type \;= TrajectoryPoint$

Postcondition :

$(q_{Regulation}[noeud] \;=\; q_{Regulation}[noeud] + computeMove(r_i[i].memory)) \wedge$
$(r_i[].memory \;\neq\; r'_i[i].memory) \wedge$
$(r_{Env}[Env].nbMessages++) \wedge$
$(q_{Env}[messages[]] \;\neq\; q'_{Env}[messages[]])$

L'agent modifie sa position et met à jour la mémoire coopérative en fonction de cette correction. De plus, il envoie une partie de l'erreur dans un message à destination de la barre sur laquelle il est attaché. La fonction $computeMove()$ correspond aux calculs effectués sur la mémoire coopérative pour déterminer le mouvement pouvant être mis en oeuvre afin de corriger le problème de manière coopérative.

. Pour un agent *charnière*.

$$s_i \;=\; (q_{Regulation}, r_i, V_i) \xrightarrow{ChangePosition} s'_i \;=\; (q_{Wait}, r'_i, V_i)$$

Précondition :

$r_i[i].type \;= Hinge$

Postcondition :

$(q_{Wait}[noeud_1] = q_{Regulation}[noeud_1] + computeMove(r_i[i].memory)) \wedge$
$(q_{Wait}[noeud_2] = q_{Regulation}[noeud_2] + computeMove(r_i[i].memory)) \wedge$
$(r_i[].memory \;\neq\; r'_i[i].memory)$

L'agent modifie la position des deux noeuds le caractérisant et met à jour la mémoire coopérative pour intégrer ce déplacement. La fonction $computeMove()$ correspond aux calculs effectués sur la mémoire coopérative pour déterminer le mouvement pouvant être mis en oeuvre afin de corriger le problème de manière coopérative.

. Les agents *barre* envoient aux charnières avec lesquelles ils sont liées les corrections qu'ils ont calculées à partir des erreurs reçues (*IncompetenceAdjustment*). Ils leur communiquent aussi les erreurs à transmettre aux barres suivantes (message *Incompetence*). Quatre messages sont donc déposés dans la boîte aux lettres (deux pour chacune des charnières voisines).

$$s_{Env}, s_i \;=\; (q_{Regulation}, r_i, V_i) \xrightarrow{ChangePosition} s'_{Env}, s'_i \;=\; (q_{Wait}, r'_i, V_i)$$

Précondition :

$r_i[i].type = Bar$

Postcondition :

$(r'_{Env}[Env].nbMessages = r_{Env}[Env].nbMessages + 4) \wedge$

$(q_{Env}[messages[]] \neq q'_{Env}[messages[]])$

7.3.6 $q_{Re-organisation} \rightarrow q_{Wait}$

Les agents dans l'état $q_{Re-organisation}$ mettent en oeuvre les actions d'attachement ou de détachement (*link*, *unLink*) en fonction de la situation. Les action de détachement sont les plus simples à exprimer.

. Détachement d'un point trajectoire i de la barre j sur laquelle il se trouve actuellement.

$$s_i, s_j \xrightarrow{unLink} s'_i, s'_j$$

Précondition :

$(r_i[i].type = TrajectoryPoint) \wedge$

$(r_i[i].attach = j) \wedge$

$isStable(r_i[i].memory) \wedge$

$(r_j[j].type = Bar)$

Postcondition :

$(r'_i[i].attach = null) \wedge$

$(r'_i[i].memory = \{\}) \wedge$

$(r'_j[j].support = null)$

Si la mémoire coopérative de l'agent est stable il doit se détacher, considérant que le mécanisme courant ne lui permet plus de s'améliorer.

. Détachement d'un agent *charnière* d'une des barres à laquelle il est attaché.

$$s_i, s_j \xrightarrow{unLink} s'_i, s'_j$$

Précondition :

$(r_i[i].type = Hinge) \wedge$

$(minDegCoop(r_i) = j) \wedge$

$(r_i[i].attach_k = j) \wedge$

$(r_j[j].type = Bar) \wedge$

$(r_j[j].attach_m = i) \wedge$

Postcondition :

$(r'_i[i].attach_k = null) \wedge$

$(r'_i[i].memory = \{\}) \wedge$

$(r'_j[j].attach_m = null)$

Dans le cas ou la mémoire coopérative est stable, l'agent *charnière* se détache de l'agent barre voisin ayant le degré de coopération le plus faible (la fonction $minDegCoop()$ retourne l'identifiant du voisin le moins coopératif).

La mise en oeuvre du protocole complet de recherche d'un nouveau voisin (en fonction des degrés de coopération) pour effectuer une action *link* est plus lourde à mettre en oeuvre à cause du nombre important d'agent pouvant être concerné. Ce protocole peut être exprimé de manière condensée de la manière suivante :

$$\overrightarrow{s} \xrightarrow{link} \overrightarrow{s'}$$

Précondition :

$(r_i[i].type = TrajectoryPoint) \wedge$

$(r_i[i].attach = null) \wedge$

$(r_j[j].type = Bar)$

Postcondition :

$(r'_i[i].attach = j) \wedge$

$(r'_j[j].support = i)$

Afin de préciser ce comportement, il serait nécessaire de définir de nouveaux états décrivant les diverses phases dans le recherche de nouveaux partenaires (envoi des requêtes, analyse des réponses, choix du nouveau partenaire) au niveau de l'état $q_{Re-organisation}$. De plus, de nouveaux champs de connaissances sont nécessaires comme, par exemple, les degrés de coopération associés aux agents qui entrent en compte lors du choix de l'agent avec lequel l'action *link* va s'effectuer.

7.4 Conclusion

Nous avons présenté ici une illustration de l'utilisation du modèle défini dans le chapitre précédent. On peut remarquer que les diverses phases mises en oeuvre correspondent, de manière naturelle, avec les phases de conception définies par la méthode *ADELFE*. Ceci est en grande partie dû au fait que le modèle a été lui-même développé selon les mêmes principes que ceux de la méthode, à savoir les principes édictés par la théorie de *AMAS*. De fait, dans les deux cas, on s'intéresse tout d'abord à

l'environnement puis à la définition des agents et enfin aux comportements de ces agents.

Un autre point notable est l'utilité de l'emploi de règles pour définir les comportements des agents et du système. Bien que la dynamique globale soit relativement complexe, le modèle permet son expression au moyen d'un nombre finalement relativement réduit de règles.

Conclusion

LES travaux présentés dans cette thèse portent sur trois axes majeurs : l'étude, au travers de l'expérimentation, de la théorie des systèmes multi-agents adaptatifs d'une part, la résolution innovante d'un problème difficile pour les concepteurs de mécanimes d'autre part, et l'élaboration d'un modèle basé sur les automates pour finir. A cette fin, le *Mechanical Synthesis Solver*, outil de conception automatique de mécanismes, a été développé et de nombreux algorithmes de traitement des situations non coopératives au niveau des agents mécaniques ont pu être testés. Cette même application a été utilisée afin d'illustrer l'utilisation du modèle basé sur les automates pour un système multi-agent particulier.

7.5 Le *Mechanical Synthesis Solver*

Perspectives Outre l'amélioration des actions de régulation, l'utilisation des degrés de non coopération et l'analyse des degrés de liberté que nous avons abordés à la fin du second chapitre, deux constats se sont imposés quant aux perspectives du *MSS*.

En premier lieu, afin de contourner le problème de la méconnaissance des compétences des agents, l'intégration du processus de calcul de la cinématique dans les agents est incontournable. Il est nécessaire que les agents réalisent eux-mêmes la simulation afin de pouvoir récupérer les données des mouvements effectués par le mécanisme. Cela veut dire que le *MSS* doit, en plus de sa qualité de solveur, être un simulateur distribué de mécanismes, ce qui permettra aux composants mécaniques de "savoir" quelles ont été les diverses influences qui ont participé à un mouvement donné.

Le deuxième changement qui pourrait grandement aider au développement du *MSS* est que les agents doivent être non pas les composants mécaniques mais les noeuds. Ainsi, un mécanisme serait non pas un agrégat de composants élémentaires mais un ensemble de noeuds dont les relations détermineraient les différents corps rigides et les joints du système. Cette idée est en partie issue de l'idée de simulateur distribué pour le calcul des mouvements : ne pouvant définir les comportements de composants (c'est justement le problème !) il faut décomposer à nouveau les agents du système en système multi-agent. On rejoint en ce sens certains algorithmes d'affichage de graphes qui fonctionnent selon le principe de liens élastiques entre les noeuds ou même certains simulateur jouets

que l'on peut trouver sur l'Internet. Même si ce type de simulation ne fournit pas de résultats très précis, il suffirait, a priori, que le comportement exhibé soit suffisamment proche de la réalité pour probablement fournir des données exploitables. De plus, un autre avantage avec l'"agentification" des noeuds est la possibilité de permettre l'apparition de corps rigides complexes (contenant de nombreux sommets) à l'aide des processus de réorganisation et d'évolution.

Des résultats du *MSS*. En tant que programmeur, les résultats obtenus à l'issue du développement de cette application sont décevants puisque, dans son état actuel, il se rapproche plus d'un prototype que d'un logiciel complet et opérationnel. Toutefois, l'évolution des résultats, au fil des algorithmes testés, reste encourageante pour ce qui est de l'applicabilité des principes de la théorie des *AMAS* à ce problème particulier. En effet, les améliorations obtenues par la mise au point des traitements basés sur la mémoire coopérative représentent une avancée appréciable dans la résolution de ce problème. De plus, le fait que cette approche purement locale, qui évite toute utilisation d'informations de niveau global, fournisse de meilleurs résultats que les précédentes ne peut que renforcer la poursuite des études menées sur la conception de systèmes artificiels à fonctionnalité émergente.

Le travail réalisé sur le *MSS* a tout de même permis de relever un point d'importance vis-à-vis de conception des systèmes basés sur la théorie des *AMAS* : la difficulté, sinon l'impossibilité, de parvenir à déterminer de manière précise les situations non coopératives et les traitements associés. Cette remarque est nouvelle dans l'experience accumulée, au gré des développements des diverses applications réalisées jusqu'à aujourd'hui, au sein de l'équipe *SMAC*. S'il n'existe probablement pas de véritable solution pour surmonter ce problème, ce devra être un point d'analyse important dont il faudra tenir compte à l'avenir afin d'identifier cet écueil, s'il se présente à nouveau dans les futures applications. A cette fin, la méthode *ADELFE* devra aborder cette question dans ses développements.

Pour finir, il ne faut pas perdre de vue que l'on doit pouvoir résoudre le problème de la connaissance des agents mécaniques en intégrant le calcul de la cinématique au sein du *MSS*. Cet axe de développemnt est probablement le seul qui puisse réellement permettre, à l'avenir, d'obtenir un outil complet de résolution automatique du problème de conception de mécanismes.

7.6 Modèle des systèmes multi-agents

Les résultats des travaux relatifs au modèle dédié aux systèmes multi-agents sont, pour leur part, beaucoup plus positifs. Nous avons pu, a priori, définir un modèle complet permettant de modéliser une plage très large de systèmes multi-agents. De plus, l'utilisation de règles permet de définir de manière simple les nombreuses transitions qui sont nécessaires à la modélisation du comportement

des systèmes multi-agents. On remarquera que ce modèle ayant été développé selon une approche similaire à celle qui préside la conception des systèmes multi-agents en général (en partant des agents pour "remonter" vers le système global), il se révèle très adapté et pratique à utiliser. L'exemple de son application pour la modélisation du *MSS* en est l'illustration.

Le modèle ayant atteint un niveau d'expressivité conséquent, les travaux relatifs à l'expression et à la preuve de propriétés vont être mis en oeuvre dans un futur proche. Notre principal centre d'intérêt en ce sens porte sur les thèmes de l'émergence, de la coopération et de la dynamique de convergence des systèmes multi-agents adaptatifs. Si ces travaux aboutissent, nous pouvons espérer de grandes avancées dans le domaine de l'étude de l'émergence et des systèmes complexes en général, et dans celui de la conception des systèmes artificiels permettant la résolution de problèmes par émergence en particulier.

Une dernière perspective, pour l'utilisation de ce modèle, est de fournir des outils pour supporter le processus de conception des systèmes multi-agents adaptatifs défini dans *ADELFE* en terme de vérification de modèle et de guide pour le développement des situations non coopératives. Pour cela, il faudra intégrer au processus de conception l'élaboration automatique des automates représentant les agents à partir des définitions qui en sont faites en *UML*, et la génération des règles en se basant sur les diagrammes de protocole réalisés en *AUML*. On remarquera à ce propos que *ADELFE* intègre déjà la génération automatique de machines à état à partir des diagrammes *AUML*. A partir de ce modèle, des outils de preuves automatiques pourront vérifier l'adéquation du système en cours de conception avec le modèle d'agents coopératifs, et vérifier que les comportements définis par les règles sont bien conformes à la théorie des *AMAS* du point de vue de l'autonomie de décision, de la coopération, de l'émergence des fonctionnalités, etc. Le support au développement des situations non coopératives pourra quant à lui être mis en oeuvre par un outil d'assistance présentant au concepteur toutes les transitions possibles pour les différents états des agents ou du système, ainsi que des évaluations de ces transitions du point de vue de la coopération.

7.7 Sur la théorie et la méthodologie

Dans la seconde partie de cette thèse, nous avons présenté les phases de conception du *MSS* ainsi que les développements des situations non coopératives pour les agents mécaniques. Les difficultés rencontrées nous permettent de tirer plusieurs enseignements de ce travail. Du point de vue méthodologique, la méthode *ADELFE* pêche sur le support apporté lors de l'analyse des situations non coopératives et des actions à mettre ne oeuvre pour les résoudre. Conformément à l'objectif premier de l'expérimentation dans le cadre de l'étude de la théorie des *AMAS*, nous avons pu souligner de

nouveaux éléments permettant d'ouvrir de nouveaux axes de recherche.

Conséquences méthodologiques. La méthode *ADELFE* offre une couverture plus que correcte lors des premières phases de conception. Comme le reflète la partie de cette thèse consacrée à la conception du *MSS*, les phases d'expression des besoins, d'analyse et de conception, associées aux notations et aux divers outils spécifiquement développés pour supporter la méthode, permettent de concevoir relativement aisément un système multi-adaptatif. Toutefois, de par le fait que la méthode ne couvre pas les phases plus tardives de conception, la méthode fournit une aide plus que limitée lors de la mise au point des processus de réorganisation et, plus généralement, lors du travail portant sur l'identification et le traitement des situations non coopératives.

Cette faiblesse a été identifiée lors du développement d'autres applications comme le souligne G. Picard dans la conclusion de sa thèse (Picard, 2004). Parmi les solutions proposées, celle du *Living Design* (Georgé *et al.*, 2003a) semble être celle qui aurait été la plus adaptée à la vue de notre expérience sur le développement du *MSS*. En effet, un tel procédé aurait permis de fournir un cadre adapté aux nombreuses expérimentations que nous avons menées et qui se sont révélées être un bon facteur d'apprentissage dans notre connaissance du domaine mécanique qui nous était méconnu.

Conséquences théoriques. Du point de vue théorique, le principal apport qu'a fourni le développement du *MSS* est que l'on ne peut raisonnablement pas réussir à développer des agents coopératifs dont on ne peut définir les compétences. Ce cas ne s'était, jusqu'à présent, jamais présenté dans les problèmes traités à l'aide de la théorie des *AMAS*. Généralement, soit les compétences étaient déjà dans les agents - et il n'y avait aucun problème à ce niveau -, soit il était possible d'obtenir une connaissance précise de ces compétences et de s'appuyer dessus lors de la conception pour déterminer les situation non coopératives et les traitements associés. Le problème rencontré dans le *MSS* est que l'activité des agents est prise entièrement en charge par un outil extérieur dont le principe de fonctionnement ne nous permet pas, qui plus est, de récupérer des données exploitables. De plus, le domaine relatif à l'application - ici la mécanique -, s'appuie lui aussi sur des principes incompatibles avec les systèmes multi-agents et rend toute expertise non constructive de ce point de vue.

Les premiers algorithmes d'ajustement des dimensions apportent quand à eux la confirmation que l'exploitation de données globales, et, a fortiori, de données relatives à la fonction que doit réaliser le système (la raideur), n'est pas pertinente dans le cadre de l'utilisation de la théorie des *AMAS*. Bien que généralement ces informations semblent fournir des critères permettant de guider rapidement le processus de résolution vers des solutions, il s'avère qu'ils restent relativement inefficaces pour ce qui est l'évitement des minima locaux et, dans le *MSS*, pour le passage de singularités dans l'espace des solutions.

Bibliographie

E.H.L. AARTS et P.J.M. van LAARHOVEN : Statistical cooling : A general approach to combinatorial optimization problems. volume 40, pages 193–226, 1985.

P. ALA-SIURU, J. PLOMP et J. HUHTINEN : Designing agent systems : Some critical issues. *In* Heikki HYÖTYNIEMI, éditeur : *Proceedings of the 9th Finnish Artificial Intelligence Conference*, volume 2, pages 61–67, Helsinki University of Technology, August 2000. Finnish Artificial Intelligence Society FAIS.

S.M. ALI et R.M. ZIMMER : The question concerning emergence. *In Abstract Book of First International Conference on Computing Anticipatory Systems*, 1997.

A. ARNOLD : *Systèmes de transitions finis et sémantique des processus communicants*. Masson, 1992.

J. BATALI : The negociation and acquisition of recursive grammars as a result of competition among grammars. *In Linguistic Evolution through Langage acquisition : Formal and Computational Models*. Cambridge University Press, 2002.

Carole BERNON, Valérie CAMPS, Marie-Pierre GLEIZES, Pierre GLIZE, Sylvain PEYRUQUEOU et Gauthier PICARD : Ingénierie des AMAS pour l'emploi du temps. ETTO - Emergent TimeTable Organization . Rapport de recherche ASA-PRC IA, IRIT, Université Paul Sabatier, Toulouse, 2002.

R.A. BROOKS : Elephants don't play chess. *In Designing Autonomous Agents*, pages 3–15. MIT Press, 1990.

N. BUSI, P. CIANCARINI, R. GORRIERI et G. ZAVATTARO : Models for coordinating agents : a guided tour. *In* A. OMICINI, R. TOLKSDORF, G. WEISS et F. Zambonelli (EDS.), éditeurs : *Coordination Models and applications for agents*. pringer-Verlag, 2001.

M. CAMPBELL : *The A-Design invention machine : A means of automating and investigating conceptual design*. Thèse de doctorat, Carnegie Mellon University, 2000.

M. CAMPBELL, J. CAGAN et K. KOTOVSKY : A-design : An agent-based approach to conceptual design in a dynamic environment. *In Research in Engineering Design*, volume 11, 1999.

V. CAMPS : *Vers une théorie de l'auto-organisation dans les systèmes multi-agents basée sur la coopération : application à la recherche d'information dans un système d'information répatie*. Thèse de doctorat, Université Paul Sabatier, 1996.

V. CAMPS, M.P. GLEIZES et P. GLIZE : Une théorie des phénomènes globaux fondée sur des interactions locales. *In Actes des Sixième journées francophones IAD&SMA Pont-á-Mousson - Editions Hermès*, 1998.

P. CANNIZZO-PUYET, B. CARPUAT, N. CEVALLOS RIVAS, P. COHEN, M.P. GLEIZES, P. GLIZE, A. MACHONIN, C. PIQUEMAL-BALUARD, C. REGIS et S. TROUILHET : Rapport d'activité de l'équipe Système Multi-Inférentiel. Projet SYNERGIC. Rapport interne RIT/91-100-R, IRIT, Université Paul Sabatier, Toulouse, Décembre 1991.

D. CAPERA, J.P. GEORGÉ, M.P. GLEIZES et P. GLIZE : The amas theoy for complex problem solveing based on self-organizing cooperative agents. *In TAPOCS*, Linz, Autriche, 2003.

C. CARABELEA, O. BOISSIER et A. FLOREA : Autonomie dans les systèmes multi-agents - essai de classification. *In JFIADSMA*, 2003.

RC. CHELOUAH et P. SIARRY : A continuous genetic algorithm designed for the global optimisation. *In Journal of Heuristics*, volume 6, 2000.

Y. COLLETTE et P. SIARRY : *Optimisation Multiobjectif*. Eyrolle, 2002.

R. DAVIS et R. SMITH : Negotiation as a metaphor for distributed problem solving. *In Artificial Intelligence*, volume 20, 1983.

Y. DEMAZEAU : From interactions to collective behavior in agent-based systems. *In Proceedings of the 1st european conference on cognitive sciences*, 1995.

Y. DEMAZEAU : *VOYELLES*. Habilitation à diriger des recherches, INP Grenoble, Grenoble, Avril 2001.

A. DROGOUL et D. SERVAT : Informatique diffuse : vers un monde fait d'agents ? *In Actes des JFIADSMA'01. Fondements des systèmes multi-agents : modèles, spécifications formelles et vérifications*, Paris, 2001. Hermes.

J. DRÉO, A. PÉTROWSKI, P. SIARRY et E. TAILLARD : *Métaheuristiques pour l'optimisation difficile*. Editions Eyrolles, Paris, 2003.

C.L. DYM : *Engineering Design : A Synthesis of Views*. Cambridge University Press, 1994.

L.D. ERMAN, F. HAYES-ROTH, V.R. LESSER et R. REDDY : The hearsay-ii speech-understanding system : Integrating knowledge to resolve uncertainty. *In ACM Computing Surveys*, volume 12, 1980.

H. ESCHENAUER, J. KOSKI et A. OSYCZKA : *Multicriteria Design Optimization*. Springer-Verlag, Berlin, Germany, 2000.

J. FERBER : *Les systèmes multi-agents : Vers une intelligence collective*. InterEditions, Paris, 1995.

K. FISCHER : Holonic multiagent systems - theory and applications. *In Proceedings of the 9th Portuguese Conference on Progress in Artificial Intelligence (EPIA-99)*, volume 1695. LNAI, 1999.

S. FORREST : Emergent computation : Self-organizing, collective, and cooperative phenomena in natural and artificial computing networks. *In Special issue of Physica D*. MIT Press / North-Holland, 1991.

L. GASSER, C. BRAGANZA et N. HERMAN : *MACE : A flexible testbed for distributed AI*. Pitman, 1987.

J.-P. GEORGÉ, G. PICARD, M.-P. GLEIZES et P. GLIZE : Living Design for Open Computational Systems. *In* M. FREDRIKSSON, A. RICCI, R. GUSTAVSSON et A. OMICINI, éditeurs : *International Workshop on Theory And Practice of Open Computational Systems (TAPOCS) at 12th IEEE International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE'03)*, pages 389–394, Linz, Austria, June 2003a. IEEE Computer Society.

Jean-Pierre GEORGÉ : L'émergence. Rapport de recherche 2003-12-R, IRIT, Université Paul Sabatier, Toulouse, juillet 2003.

Jean-Pierre GEORGÉ : *Résolution de problèmes par émergence, Etude d'un Environnement de Programmation émergente*. Thèse de doctorat, Université Paul Sabatier, Toulouse, juillet 2004.

J.P. GEORGÉ, M.P. GLEIZES et P. GLIZE : Conception de systèmes adaptatifs à fonctionnalité émergente : la théorie amas. *In Revue RIA*, volume 4, 2003b.

Pierre GLIZE : *L'Adaptation des Systèmes à Fonctionnalité Emergente par Auto-Organisation*. Habilitation à diriger les recherches, Université Paul Sabatier, juillet 2001.

F. GLOVER et M. LAGUNA : *Tabou Search*. Kluwer Academic ublishers, 1997.

D.E. GOLDBERG : *Genetic Algorithms in Search, Optimisation and Machine Learning*. Addison-Weyley, 1989.

J. GOLDSTEIN : Emergence as a construct : Hisotry ans issues. *In Emergence*, volume 1, pages 49–71, 1999.

B. HAJEK et G. SASAKI : Simulated annealing - to cool or not. volume 12, pages 443–447, 1989.

C. HEWITT : Viewing control structures as patterns of passing messages. *In Artificial Intelligence*, volume 8, 1977.

J.H HOLLAND : *Emergence - from Order to Chaos*. Addison-Wesley, 1997.

P. HORN : Autonomic computing : Ibm's perspective on the state of information technology. *In Proceedings of the 9th Finnish Artificial Intelligence Conference*, 2001.

D. KASAKOV et M. BARTLETT : Social learning through evolution of langage. *In Proceedings of The First European Workshop on Multi-Agent Systems*, 2003.

J. KENNEDY et R.C. EBERHART : Particle swarm optimization. *In Journal of Mechanical Design*, volume 4, 1995.

J. KENNEDY, R.C. EBERHART et Y. SHI : *Swarm intelligence*. Morgan Kaufmann Publishers, San Francisco, US, 2001.

S. KIRBY : Learning, bottlenecks and the evolution of recursive syntax. *In Linguistic Evolution through Langage acquisition : Formal and Computational Models*. Oxford University Press, 2002.

S. KIRKPATRICK, C. GELATT et M. VECCHI : Optimisation by simulated annealing. *In Science*, volume 220, 1983.

D.B. LENAT : Beings : Knowledge as interacting experts. *In Proceedings of IJCAI'75*, 1975.

M.J. MATARIĆ : *Interaction and Intelligent Behavior*. Thèse de doctorat, MIT, 1994.

J. MCCARTHY, M. MINSKY, N. ROCHESTER et C. SHANNON : A proposal for the dartmouth summer research project on artificial intelligence. 1955.

R. MILNER : *Communication and concurrency*. Prentice Hall, 1989.

G. PAHL et W. BEITZ : *Engineering Design : A Systematic Approach*. Springer-Verlag, London, UK, 1996.

G. PICARD : *Méthodologie de développement de systèmes multi-agents adapatatifs et conception de logiciels à fonctionnalité émergente*. Thèse de doctorat, Université Paul Sabatier de Toulouse III, 2004.

G. PICARD et M.-P. GLEIZES : The ADELFE Methodology – Designing Adaptive Cooperative Multi-Agent Systems (Chapter 8). *In* F. BERGENTI, M-P. GLEIZES et F. ZAMBONELLI, éditeurs : *Methodologies and Software Engineering for Agent Systems*. Kluwer Publishing, 2004.

M.A. PUCHETA et A. CARDONA : *Síntesis de tipo y dimensional de mecanismos utilizando algoritmos genéticos y ecuaciones algebraicas exastas*, volume 13. 2003.

S. SAÏT et H. YOUSSEF : *Iterative computer algorithms with applications in engineering*. IEEE Computer Society Press, 1999.

L.C. SCHMIDT et J. CAGAN : Optimal configuration design : An integrated approach using grammars. *In Proceedings of the IEEE conference on Neural Networks*, volume 120, 1998.

O. SELFRIDGE et U. NEISSER : Pattern recognition by machine. *In Scientific American*, volume 203, 1960.

W. SEN et J.B. YANG : *Multiple Criteria Decision Support in Engineering Design*. Springer-Verlag, London, UK, 1998.

T. SONTHEIMER et P. GLIZE : Informatique diffuse : vers un monde fait d'agents ? *In Actes des JFIADSMA'00. Systèmes multi-agents : Méthodologie, technologie et expériences*, Paris, 2001. Hermes.

L. STEELS : The spotaneous self-organisation of an adaptive langage. *In Machine intelligence*, volume 15. Oxford University Press, 1996.

L.W. TSAI : *Mechanism Design : Enumeration of Kinematic Strucutres According to Function*. CRC Press, 2001.

M. WEISER et J.P. BROWN : Designing calm technology. *In PowerGrid Journal*, 1996.

R.V. WELCH : *Conceptual Design of Mechanical Systems : A Representation and Computational Model*. Thèse de doctorat, University of Massachusetts, 1992.

M.J. WOOLDRIDGE et N.R. JENNINGS : Intelligent agents. *In Workshop on agent theories, Architectures, and languages ECAI'94*, volume 890, Amsterdam, 1995. Springer Verlag.

Liste des figures

1.1 Principe de l'auto-organisation 19

2.1 Méthodologie systématique pour la conception de mécanismes 30

2.2 De la chaîne cinématique aux permutations de la matrice adjacente 34

2.3 *Degree Code* 34

2.4 Exemple de problème et solution 4-barres (RRRR) 35

2.5 Solution 4-barres (PRRR) (RPRR) 35

2.6 Processus global du A-Design 38

2.7 Exemple d'introduction d'un ressort dans un mécanisme incomplet 42

2.8 Préférences des différents types d'agents 43

2.9 Classification générale des méthodes d'optimisation mono-objectif 47

2.10 Algorithme tabou simple 52

2.11 Structure générale de l'algorithme tabou continu ECTS 53

2.12 Algorithme du recuit simulé 54

2.13 Principe des algorithmes évolutionnaires 57

3.1 Processus d'ADELFE 64

3.2 Cas d'utilisation 66

3.3 Diagramme de classes préliminaire du MSS 67

3.4 Trace des interactions non coopératives issues d'un point trajectoire 69

3.5 Description d'un cycle de résolution 70

3.6 Agents coopératifs mécaniques 72

3.7 Modèle d'agent coopératif 73

4.1 Interdépendances des types d'action 82
4.2 Description d'un mécanisme 4-barres 86
4.3 Résultats d'un algorithme réactif pour un mécanisme 4-barres 87
4.4 Description d'un mécanisme 4-barres 87
4.5 Résultats d'un algorithme réactif pour un mécanisme 4-barres 88
4.6 Graphique d'illustration des limites d'un algorithme réactif 89
4.7 Graphique d'illustration des limites d'un algorithme apprenant 91
4.8 Exemple de calcul coopératif (partie basse) et non-coopératif (partie haute) 93
4.9 Apprentissage du point trajectoire avec un algorithme basé sur une mémoire coopérative 97
4.10 Résultats de l'ajustement d'un 4-barres avec un algorithme basé sur une mémoire coopérative 97

5.1 Etat initial du système 104
5.2 Evolution vers un 4-barres 105
5.3 Phase d'ajustement 106
5.4 L'echec de l'ajustement entraine une réorganisation 107
5.5 Deuxième phase d'ajustement 108
5.6 Solution et mesure de l'erreur au cours de la résolution 109

6.1 Graphe d'états d'un agent 114
6.2 Graphe d'états d'un agent coopératif 115

7.1 Graphe d'états d'un agent mécanique 128

Liste des tableaux

2.1 Correspondance entre graphes et mécanismes 30

2.2 Variables par domaine 40

2.3 Métaheuristiques et domaines d'inspiration 50