RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

Université Mira Abderrahmene de Béjaia

CONTRÔLE DISTRIBUÉ D'UNE ORGANISATION DE ROBOTS

PAR

Boufaghes Wahid

DIRECTEUR DU MÉMOIRE

M. AMIRAT Yacine, Professeur, Université Paris 12, France

PRÉSENTÉ À LA FACULTÉ DES SCIENCES ET SCIENCES DE L'INGÉNIEUR
DÉPARTEMENT D'INFORMATIQUE, ÉCOLE DOCTORALE D'INFORMATIQUE
ReSyD
(Réseaux et Systèmes Distribués)
POUR L'OBTENTION DU GRADE DE MAGISTÈRE EN INFORMATIQUE
À
L'Université Mira Abderrahmene de Béjaïa

Devant le jury composé de :

Président :	Mme. TAS Saadia, Maître de Conférences, Université de Béjaïa, Algérie.
Rapporteur :	M. AMIRAT Yacine, Professeur à l'Université Paris 12 - Val de Marne, France.
Examinateurs :	M. MELLIT Ali, Maître de Conférences, Université de Jijel, Algérie.
	M. BOUKERRAM Abdallah, Maître de Conférences, Université de Sétif, Algérie.

Promotion 2006 - 2007

UNIVERSITÉ MIRA ABDERRAHMENE

École doctorale d'informatique, ReSyD.

Date: **Avril 2008**

Nom : **Boufaghes Wahid.**

Titre : **Contrôle distribué d'une organisation de robots.**

Département : **Informatique.**

Grade : **Magistère en informatique.**

À ma mère

À mon père

À mes frères

À mes sœurs

À toute ma famille

Table des Matières

Table des figures

Remerciements

Si ce mémoire a pu voir le jour, c'est certainement grâce au soutien et l'aide de plusieurs personnes qui m'ont permis d'accomplir ce travail dans des conditions idéales. Je profite de cet espace pour les remercier tous.

Je voudrais tout d'abord exprimer mes profonds remerciements à Monsieur AMIRAT Yacine, Professeur à l'Université Paris - Val de Marne, pour son encadrement, sa disponibilité, sa confiance, sa patience et pour le soutien qu'il a su m'accorder durant toute cette période. Son expérience, ses qualités scientifiques et humaines ont toujours été sources d'enrichissement me permettant de mener à bien ce travail de recherche.

Je tiens à remercier chaleureusement Madame TAS Saadia, Maître de Conférence à l'Université de Béjaïa, Monsieur BOUKERRAM Abdallah, Maître de Conférence à l'Université de Sétif et Monsieur MELLIT Ali, Maître de Conférence à l'Université de Jijel, pour avoir accepté de faire partie de jury de ce mémoire.

Je désire sincèrement remercier Mr TARI Kamel, le chef de département d'informatique de l'université de Bejaia pour son aide, ses conseils et sa disponibilité. Un merci bien distingué à mes enseignants de l'école doctorale ReSyD. J'aimerai remercier également tous les étudiants de l'école doctorale pour l'environnement de travail très agréable durant ces deux dernières années.

J'aimerai adresser mes remerciements à Said et Mohammed pour leur collaboration et ses conseils avisés sur l'ensemble de ce travail de recherche.

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

Résumé

CE mémoire concerne le contrôle distribué d'une organisation de robots mobiles autonomes avec pour objectif l'exécution robuste et auto-organisée de tâches complexes dans un environnement dynamique. Plus précisément, la problématique que nous traitons concerne la coordination et l'adaptation de systèmes multi-robots évoluant dans des environnements structurés mais dynamiques. Des plans de tâches sont assignés aux robots pour leurs permettre d'atteindre des buts locaux et globaux. Ces plans sont soumis à des contraintes d'exécution et à l'occurrence de perturbations pouvant remettre en cause les plans de tâches en cours d'exécution. Les robots se trouve donc obligés de réordonnancer (ou réadapter) leurs plans locaux de manière dynamique tout en prenant en compte les objectifs locaux et globaux des robots. Ainsi, nous proposons une approche algorithmique et méthodologique permettant aux robots d'une organisation multi-robots de réordonnancer dynamiquement et de manière distribuée leurs plans locaux tout en essayant d'atteindre les objectifs locaux et globaux de l'ensemble de robots. Pour ce faire, nous avons modélisé le système multi-robots sous la forme d'un système multi-agents. À l'occurrence d'une perturbation, les agents effectuent d'abord un traitement local et en cas d'échec, ils négocient une solution de réordonnancement avec les agents auxquels ils sont liés.

Mots clés : contrôle multi-robots, système multi-agents, systèmes distribués, interaction, auto-organisation, ordonnancement dynamique, allocation de tâches, propagation de contraintes.

Abstract

THIS report concerns the distributed control for an organization of autonomous mobile robots with for objective the robust and self-organized execution of complex tasks in a dynamic environment. More exactly, the problematic which we treat concerns the coordination and the adaptation of multi-robots systems evolving in structured but dynamic environments. Plans of tasks are assigned to robots to allow for their to reach local and global goals. These plans are subjected to constraints of execution and to occurrence of disruptions which can question the plans of tasks in the execution. Robots is thus grateful to reschedule (or to readapt) their local plans in a dynamic way while taking into account the local and global objectives of robots. So, we propose an algorithmic and methodological approach allowing the robots of an organization multi-robots to dynamically reschedule and of a distributed way their local plans while trying to reach the local and global objectives of the group of robots. To do it, we modelled the multi-robots system under the form of a multi-agents system. In the occurrence of a disturbance, the agents make at first a local treatment and in case of failure, they negotiate a solution of rescheduling with the agents to whom they are bound.

Keywords : multi-robots control, multi-agents systems, distributed systems, interaction, self-organisation, dynamic scheduling, tasks allocation, constraints propagation.

خلاصة

هده المذكرة تعالج المراقبة الموزعة لمنظمة من الروبوتات المتنقلة ذات الحكم الذاتي، هده المراقبة تهدف إلى تنفيذ فعال وتنظيم ذاتي لمهام معقدة في بيئات ديناميكية. بدقه أكبر، المشكلة التي نعالجها هي تنسيق وتكييف نظم متعددة الروبوتات و التي تعمل في بيئات مهيكلة ولكن ديناميكيه. مخططات المهام المسندة إلى الروبوتات تسمح لها بتحقيق أهدافها المحلية والكلية. هذه المخططات تخضع لقيود التنفيذ و إلى وقوع الاضطرابات التي يمكن آن تشكك في مخططات العمل التي هي في طور التنفيذ. و بالتالي فان الروبوتات تكون مجبرة على إعادة ترتيب (أو إعادة تأهيل) في المخططات المحلية بطريقة ديناميكيه مع الأخذ بعين الاعتبار الأهداف المحلية و الكلية للروبوتات. وهكذا ، فإننا نقترح طريقة منهجية و خوارزمية لتمكين منظمة الروبوتات من إعادة الترتيب بشكل ديناميكي و موزع للمخططات المحلية مع محاولة الوصول إلى أهداف شخصية وكلية لجميع الروبوتات. لذلك ، فقد مثلنا النظام متعدد الروبوتات في شكل لنظام متعدد الأعوان. في حالة وجود اضطراب، يقوم الأعوان بمحاولة العلاج محليا و في حال فشل ذلك، يقومون بالتفاوض على حل لإعادة ترتيب المخططات مع الوكلاء المرتبطين بهم .

الكلمات الرئيسية : المراقبة متعددة الروبوتات ، النظم متعددة الأعوان، النظم الموزعة ،التفاعل، التنظيم الذاتي، الترتيب الديناميكي، توزيع المهام ، انتشار القيود..

Introduction générale

L'ÉTUDE et la mise en œuvre de plusieurs robots mobiles constituant un groupe autonome de robots (généralement coopératifs) pour l'accomplissement d'une ou de plusieurs tâches (généralement communes), apparaissent comme une tendance récente et importante dans le domaine de la robotique. Ils sont connus sous l'appellation : "Systèmes Robotiques Distribués" ou "Systèmes Multi-Robots" (MRS : Multi-Robot System, en anglais). Cet intérêt est dû à une large variété des domaines d'applications exigeant des solutions multi-robots. Ainsi, pour beaucoup d'applications, une équipe de robots simples peut être employée plus efficacement qu'un seul robot complexe.

Les systèmes multi-robots présentent beaucoup plus d'avantages par rapport aux systèmes mono-robot (SRS) classiques, comme la réduction du coût total du system en utilisant plusieurs robots simples moins chers par rapport à un robot complexe. Aussi, les MRS peuvent augmenter la flexibilité et la robustesse du système en tirant profit du parallélisme et de la redondance inhérentes. La complexité d'une tâche dans certains environnements peut exiger l'utilisation des robots multiples, car les besoins nécessaires en ressources sont trop difficiles à satisfaire par un seul robot. Par conséquent, il y a souvent nécessité de distribuer les activités et l'intelligence [29] entre un ensemble d'entités physiques (robots) ou virtuelles (agents).

Cependant, l'utilisation de ces systèmes pose des défis additionnels, par exemple un MRS mal conçu peut être moins efficace qu'un SRS bien conçu. Un défi primordial dans la conception des MRS efficaces est le contrôle de la complexité présentée par un groupe de robots interagissant entre eux dans un environnement dynamique. Dans la plupart des cas, il est exigé non seulement la mise en place d'un contrôle individuel pour chaque robot, mais impose aussi l'utilisation des mécanismes de contrôle appropriés afin que l'assemblage de toutes les entités robotiques montre des configuration cohérentes et efficaces pour la réalisation coopérative et robuste de leurs tâches.

La thématique générale de recherche abordée ici, concerne le contrôle d'un groupe de robots mobiles autonomes et coopératifs, l'objectif étant de réaliser des tâches complexes dans un environnement dynamique. Les avantages liés à la coopération de robots mobiles sont multiples. Parmi eux, nous pouvons citer la fiabilité, la flexibilité, la robustesse et la rapidité accrue des systèmes contrôlés induites principalement par la redondance et

les mécanismes de contrôle des robots.

Les difficultés de réaliser la coopération dans un système multi-robots sont principalement liées à la complexité accrue présentée dans le système par un plus grands nombres de robots. Dans un système mono-robot, le robot fonctionnant dans son environnement en contrôlant seulement ses propres capacités (physiques et cognitives). Par contre, en plus du contrôle individuel, chaque robot dans un MRS doit, contrôler et gérer ses interactions avec son environnement et avec les autres robots membres du groupe. Un bon système de contrôle pour un système multi-robot doit traiter ce dernier facteur pour augmente les performances avec les robots additionnels. Ce n'est pas évident en considérant chaque robot fonctionnant dans un environnement dynamique et peut le changer, et les autres robots doivent pouvoir absorber ce changement, ainsi leurs tâches ne doivent pas être affectées.

Dans le cadre de ce projet, nous nous focaliserons sur la coordination et l'adaptation des systèmes multi-robots évoluant dans des environnements dynamiques. Les robots possèdent des plans de tâches pour atteindre des buts locaux et globaux communs et qui sont soumis à des contraintes d'exécution et à l'occurrence des évènements perturbateurs. Ces derniers peuvent remettre en cause les plans en cours d'exécution. Les robots doivent donc réordonnancer (réadapter) leurs plans de manière dynamique tout en prenant en compte les objectifs locaux et globaux des robots.

En raison de ses aspects fortement combinatoires, de sa nature dynamique et de son intérêt pratique, le problème d'ordonnancement a été largement étudié dans la littérature par diverses méthodes, on peut les classer en deux grandes familles : exactes et approchées. Les premières sont capables de fournir des solutions optimales mais nécessitent des coûts calculatoires importants à cause de la taille de l'arbre des solutions générée. L'avantage des méthodes approchées par rapport aux premières méthodes, consiste en la réduction de temps du calcul et permettent d'obtenir un bon compromis entre la qualité des solutions et le temps de calcul. Leur inconvénient major est la sous-utilisation de la coopération puisque les solution proposées sont centralisées.

Afin de répondre à ces nouveaux défis, de nouvelles méthodes ont vu le jour, on parle des méthodes utilisant le paradigme multi-agents (SMA), issus des travaux en Intelligence Artificielle Distribuée (IAD). Les SMA fournissent des outils appropriés pour la modélisation des problèmes industriels. Ainsi, Les modèles qui en résultent sont composés d'un ensemble d'entités dynamiques, appelées agents, dont les comportements dépendent de leurs interactions et de l'environnement au sein duquel ils s'évoluent. Les relations entre les différents agents du système sont dynamiques et aucun contrôle central ne supervise l'exécution globale. La fonctionnalité globale est distribuée et dépend essentiellement de l'organisation des agents au sein du système établi par les relations entre les agents. Les propriétés qui définissent les agents : l'autonomie, la pro-activité, la

réactivité et la sociabilité [30], rendent l'approche de modélisation multi-agents particulièrement bien adaptée à la problématique de modélisation des systèmes multi-robots. Ce paradigme garanti la réactivité et l'autonomie décisionnelle, et la distribution des interactions entre les entités du système.

Bref, cette recherche a pour but de proposer une solution méthodologique et algorithmique au problème de l'ordonnancement des tâches dans le contexte de la robotique distribuée. Plus précisément, le problème de l'ordonnancement multi-robots que nous cherchons à résoudre se situe dans le contexte suivant : étant donnée une mission multi-robots (tâche globale du système) décomposable en sous-missions individuelles, partiellement et globalement ordonnées (plans de tâches), soumises à des contraintes de type précédence entre tâches et de ressources, et une équipe de robots ayant des capacités pour effectuer ces sous-missions dans des environnements dynamiques. L'organisation de robots doit assurer l'accomplissement robuste et auto-organisé de tâches. Pour ce faire, le système multi-robots peut être modélisé sous la forme d'un système multi-agents, nous associons à chaque robot un agent (i.e. une unité logicielle de résolution de problèmes), chargé de son contrôle tout en étant prêt à coopérer avec les autres agents si nécessaire. Ce SMA doit pouvoir résoudre les problèmes suivants :

1. L'allocation des nouvelles tâches arrivant après le démarrage de la mission, aux différents agent-robots tout en essayant de garantir les objectifs locaux et globaux.
2. Chaque agent-robot doit être capable de réordonnancer (réadapter) son plan de tâches en fonction d'une évolution au niveau individuel ou global en cas de réorganisation du système (événement perturbateur, ajout de nouvelles tâches, défaillance d'un robot, etc).
3. La garantie de la cohérence globale de l'exécution de la mission (respecter les contraintes de précédence).

Nous supposons que l'élaboration et l'attribution des plans (missions individuelles) initiaux sont réalisées hors-ligne par un planificateur spécialisé (manuel ou automatique) non traité dans ce mémoire.

Pendant la réalisation d'une mission, l'ordonnancement doit :

1. Minimiser le retard engendré par une perturbation (réalisation la plus rapide de l'application).
2. Minimiser le nombre des contraintes non respectées. Par conséquent, minimiser le nombre d'agents affectés par la perturbation.

La méthode d'ordonnancement proposée doit :

1. Garantir l'obtention d'une solution dans toutes les situations d'exceptions possibles.

2. Minimiser le temps de l'élaboration d'un nouveau ordonnancement (complexité de la méthode).

En fonction du type d'événement perturbateur engendrant un réordonnancement d'un ou de plusieurs plans, le réordonnancement de plan pour un agent peut s'effectuer de différentes manières, comme :

- La permutation interne des tâches (d'un même plan).
- Le décalage à gauche ou à droite des tâches.
- L'insertion d'une nouvelle tâche dans son plan courant.
- L'abandon d'une tâche au profit d'un autre agent jugé plus prioritaire.
- La remise en cause des contraintes temporelles.

Structure du mémoire

Le présent mémoire est décomposé en cinq chapitres. Le premier expose une introduction au contrôle robotique, ainsi, nous présentons les composantes fondamentales de l'architecture d'un robot mobile autonome et faisons une analyse des résultats des travaux menés sur les architectures de contrôle et de décision des systèmes mono et multi-robots. Dans le deuxième chapitre, nous présentons les notions utilisées pour traiter des problèmes d'ordonnancement général, notamment celles qui sont le plus souvent référencées pour traiter des problèmes d'ordonnancement dans les systèmes multi-robots, nous exposons les différents types de problèmes d'ordonnancement et analysons les principales approches de résolution existantes dans la littérature. Dans le troisième chapitre, nous proposons un modèle multi-agents pour la mise en œuvre d'applications robotiques complexes exigeant un réordonnancement de tâches sous de fortes contraintes de réactivité et de couplage entre les plans de tâches des robots de l'organisation. Le quatrième chapitre est consacré à la présentation détaillée du modèle de coopération proposé pour le réordonnancement et l'allocation dynamique de tâches, ainsi, nous exposons en détaille les stratégies de résolutions proposées. Le dernier chapitre expose l'implémentation et la validation des modèles de coopération et de l'algorithmique associée proposés dans ce mémoire

Chapitre 1

Introduction aux systèmes robotiques distribués

1.1 Introduction

Le développement d'un système multi-robots doit obligatoirement s'interroger avec l'élaboration de l'architecture de contrôle et de décision de chaque robot membre, pour cela, nous présentons dans un premier temps, les principales composantes de l'architecture d'un robot mobile autonome et son fonctionnement général, puis nous faisons un bref tour d'horizon des architectures existantes de contrôle et de décision d'un seul robot mobile et discutons les critères de choix d'une architecture de contrôle. Dans un deuxième temps, nous présentons les systèmes robotiques distribués, les principes de base des approches proposées pour le contrôle des organisations de robots, ainsi que quelques architectures de références et des exemples d'applications.

1.2 Architecture d'un robot mobile

Le contrôle d'un robot est la capacité de le guider à atteindre son but de façon autonome, c'est-à-dire sans interagir avec un opérateur humain. On appelle robots intelligents ceux qui intègrent un contrôleur intelligent [38]. La plupart de ces robots intelligents sont utilisés dans des tâches d'intervention et de service qui sont caractérisées par une méconnaissance a priori de l'environnement et de leurs changements, tels que : la supervision, le surveillance, la maintenance, le transport et l'exploration dans des environnements dynamiques et hasardeux ou hostiles pour l'homme, tels que les centrales nucléaires, l'espace, les mines et le fond des mers.

1.2.1 Composantes d'un robot mobile autonome

Pour pouvoir traiter les problèmes de complexité et de changements dynamiques et imprévus inhérents aux environnements réels (déplacement, apparition, disparition des objets de différents types, etc) dans lesquels un robot mobile s'évolue continuellement en interagissant avec les autres entités (robots, objets, opérateur humain, etc), un robot devra être constitué des composantes matérielles et logicielles avec une complexité fortement liée à celle de l'environnement.

1.2.1.1 Composantes matérielles

Un robot mobile embarqué [36] doit avoir un corps physique (hardware) dans lequel on trouve une plate-forme mobile à laquelle on peut rattacher un ensemble de :

- **Capteurs :** lui permettant d'acquérir les informations provenant de l'environnement (sonars à ultrasons, odomètre, cameras vidéo, etc).
- **Actionneurs (effecteurs) :** sont utilisés pour se déplacer à l'intérieur de l'environnement et d'y interagir (roues contrôlées, bras manipulateurs, etc).

1.2.1.2 Composantes logicielles

Elles constituent l'ensemble de modules responsable du processus de récupération et d'interprétation des données brutes en provenance de capteurs ou le traitement des commandes de haut niveau selon les besoins de l'application (liés aux buts pour lesquels le robot est conçu) afin de prendre des décisions pour générer des commandes de contrôle adéquates aux actionneurs, qui vont les exécuter en tant qu'actions dans l'environnement. Le regroupement de ces composantes logicielles est appelé `Architecture de Contrôle Robotique`.

Qu'est-ce que l'autonomie d'un robot ?

Les environnements inconnus structurés ou non demandent au robot d'avoir plus ou moins d'autonomie tant au niveau décisionnel qu'au niveau physique (déplacement).

Une définition commune de l'autonomie d'un robot est :

> Un robot doit être capable d'accomplir les tâches pour lesquelles il a été conçu de manière cohérente sans intervention de l'opérateur humain ou d'autres systèmes.

Selon [1], l'autonomie d'un robot est décomposée en trois sous composantes : Énergétique, Décisionnelle et Sensorielle.

Donc, un robot mobile autonome est un système mécanique, électronique et informatique complexe.

1.2.2 Boucles sensori-motrices d'un robot mobile

Un robot mobile est commandé par une boucle de contrôle exécutée périodiquement.

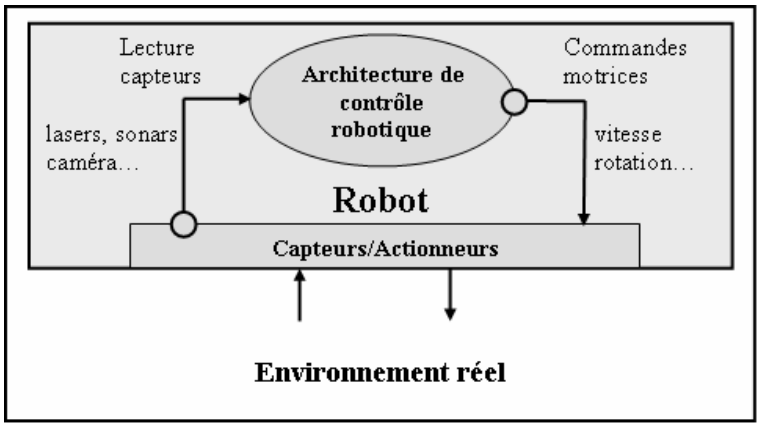


FIG. 1.1 – Boucle de contrôle d'un robot mobile

Cette boucle fait une lecture des données reçues par les capteurs, l'interprète selon les besoins (types d'applications), génère les commandes et les envoie aux actionneurs. La fréquence d'exécution de cette boucle varie selon les types de capteurs et d'actionneurs utilisés, ainsi que la dynamique de l'environnement du robot. Selon l'architecture utilisée, la boucle de contrôle peut être décomposée en plusieurs sous boucles de contrôle organisées selon plusieurs manières.

1.2.3 Architecture de contrôle robotique

La littérature est pleine d'architectures de contrôle possibles pour un robot mobile, ces architectures constituent un petit nombre de classes fondamentalement différentes des méthodologies de contrôle d'un robot. Les trois classes principales sont : les architectures de contrôle comportementales, les architectures de contrôle délibératives et les architectures issues de leur combinaison, c'est à dire les architectures de contrôle hybrides.

1.2.3.1 Architectures de contrôle comportementales

L'architecture de subsumption présentée dans [12] (cf. Fig.1.2) a été parmi les premières architectures comportementales proposées. Elle est composée d'une collection de comportements. Chaque comportement est implémenté en tant que module logiciel ou matériel recevant des entrées à partir des capteurs du robot et/ou d'autres comportements dans le système, et envoyant des sorties aux actionneurs du robot et/ou

aux autres comportements. Ce qui donne une architecture constituée d'un réseau de comportements agissants l'un sur l'autre, leurs activations dépend d'un ensemble de règles (arbitrage de comportement), comme les priorités entre les comportements ou un mécanisme de vote. La réactivité est assurée dans ces systèmes en raison de leurs réponses en temps réel dues à leurs constitutions d'une collection de règles de la forme Condition-Action préprogrammée et concurrentes. Un système à base de comportements est réalisé de bas en haut (bottom-up) et une représentation interne distribuée de l'état de robot peut être employée pour émerger des comportements de plus en plus abstraits en les étendant l'un sur l'autre.

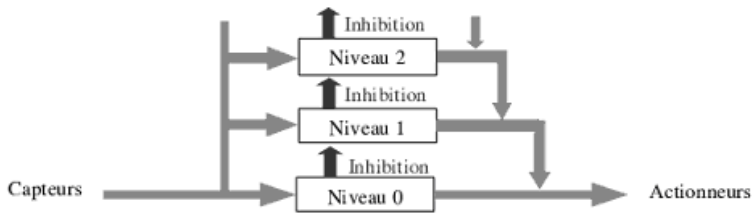


FIG. 1.2 – Architecture de subsumption

Plusieurs architectures comportementales ont été proposées, Alliance [45], BLE [59], [32], [41], etc.

Limitations

- L'absence de représentations abstraites des connaissances fait qu'elles ne se prêtent pas très bien à traiter des situations plus complexes. Difficulté à réaliser des tâches structurées qui ont besoin d'être planifiées.
- Quand la complexité et le nombre de comportement augmentent, les interactions entre les comportements augmentent aussi jusqu'au point où il devient difficile de prédire le comportement général du système.
- Tout nouveau composant doit prendre en compte l'organisation interne des niveaux inférieurs pour pouvoir subsumer ses entrées et sorties, ce qui fait qu'un changement dans le niveau le plus bas risque d'avoir des conséquences sur tout le système

1.2.3.2 Architectures de contrôle délibératives

Dans ce type d'architectures, le robot utilise toutes les informations provenant des capteurs et toutes les connaissances stockées localement, pour sélectionner la prochaine action à entreprendre. Ce raisonnement se traduit typiquement sous forme de planification, qui exige une recherche des séquences possibles *état-action* et de leurs résultats. Le robot doit construire et évaluer potentiellement tous les plans possibles jusqu'à ce

qu'il trouve un qui lui indiquera comment atteindre son but, résoudre un problème ou décider une trajectoire pour l'exécuter. La planification exige l'existence d'une représentation interne du monde, ce qui permet au robot d'anticiper l'avenir, pour prévoir les résultats des actions possibles dans divers états, afin de produire des plans optimaux d'actions. Le modèle interne, doit ainsi être maintenu précis et à jour. Les architectures proposées dans [42], [7] et [49] en sont de bons exemples.

Limitations

- La génération de plans précis demande beaucoup de ressources (mémoire et temps de calcul).
- Puisque l'environnement est dynamique et partiellement observable, et que les capteurs sont imprécis, il est très difficile de tout prévoir à l'avance.
- la délibération permet de prédire et penser à l'avenir (planification) pour éviter des mauvaises actions. Mais penser trop longtemps peut être dangereuse (par exemple, tomber à un obstacle).

1.2.3.3 Architectures de contrôle hybrides

Plusieurs architectures hybride ont été proposées, CLARATy [40], IDEA [39], LAAS [46], DAMN [47], MBA [10], etc. Ces architectures essayent de combiner les avantages des architectures comportementales et celles délibératives, c'est à dire, elles sont réalisées par l'intégration dans la même architecture : Des comportements qui contrôlent les fonctions de bas niveau et qui sont responsables de la réactivité du système, ainsi que des modules décisionnels (planification, ordonnancement, etc) responsables de la prise de décision de haut niveau. L'interaction des deux parties exige un composant intermédiaire. Pour cela, les architectures hybrides s'appellent souvent les "architectures à trois couches" [12], elles sont constituées de trois couches : réactives, intermédiaires et délibératives.

L'architecture à trois-couche ne représente pas la seule combinaison possible des modules réactifs avec les modules de délibération (planification, ordonnancement et gestion de ressources, allocation de tâches, etc). Ces modules appartiennent à une intégration hiérarchique de délibération et de réaction avec différentes activités et différents niveaux d'abstraction. La délibération et la réaction peuvent être couplées en tant qu'activités concurrentes, comme la délibération peut guider la réaction en (re)configurant les paramètres des modules de contrôle réactif (comme dans **CLARATy**).

1.2.3.4 Meilleure architecture

Il n'y a pas à proprement dit une meilleure architecture, mais ça dépend de l'application et du contexte pour lesquels un robot est conçu.

Chacune des architectures présentées ci-dessus a ses forces et faiblesses, et toutes jouent des rôles importants et réussissent dans certaines configurations (problèmes et applications). Les différentes architectures sont souhaitables pour différentes situations, nature des tâches, et les capacités matérielles et logiciel du robot.

Si on peut obtenir un modèle complet et précis du monde et un temps suffisant pour planifier, une architecture délibérative va permettre au robot d'agir stratégiquement (prédiction et raisonnement à l'avenir : planifier) et sélectionner la prochaine action à entreprendre pour éviter des mauvaises actions dans une situation donnée. Cependant, étant situé dans un monde bruyant et dynamique rend habituellement ceci impossible, alors penser long que réagir peut être dangereux (par exemple, tomber à un obstacle). Ainsi, peu de robots situés sont purement délibératifs. Quant, les architectures comportementales sont plus adaptées pour les environnements dynamiques dans lesquels des réponses plus rapides et adaptatives sont nécessaires, mais le manque des capacités délibératives (planification et ordonnancement, etc) rend ces approches inadéquates pour des tâches plus complexes. En revanche, les architectures hybrides s'avèrent bien adaptées pour des environnements et des tâches où des modèles internes et la planification peuvent être déployés et qui n'exigent pas un très fort temps réel.

Ainsi, pour choisir une architecture de robot mobile autonome temps réel, il faut satisfaire quelques spécifications de comportement, citons par exemple [17] : *la réactivité, la robustesse, le comportement intelligent, l'intégration d'informations multiples, la résolution de buts multiples, la sûreté.* Ainsi que quelques exigences au niveau de la conception du système : la programmabilité, la modularité, la flexibilité, l'évolutivité, l'autonomie et l'adaptabilité.

L'autonomie décisionnelle d'un robot doit inclure l'implémentation d'une ou de plusieurs fonctionnalités de l'Intelligence Artificielle, telles que [35] : *la planification, le contrôle d'exécution, la reconnaissance de situation, le diagnostic* et peut également incorporer des mécanismes *d'apprentissage.* La complexité du processus de prise de décision est liée à un certains nombre de propriétés de l'environnement : *déterministe ou non déterministe, régulier ou non régulier, statique ou dynamique, discret ou continu.*

1.2.4 Conclusion

Dans cette première partie, nous avons présenté les grandes classes des architectures de contrôle des robots mobiles autonomes. Les architectures purement délibératives présentent l'intérêt de doter le robot mobile de capacités de raisonnement lui permettant d'accomplir une tâche par décomposition en sous tâches. Ces architectures ont cependant plusieurs inconvénients. L'inconvénient majeur même pour un robot doté de

moyens de perceptions assez importants est celui du manque probable de caractère réactif qui confère au robot des capacités de suivre l'évolution de son environnement par des réactions rapides à des stimulus de l'environnement. Quant aux architectures purement réactives, malgré tout l'intérêt qu'elles présentent au niveau robustesse, elles souffrent cependant d'un manque de flexibilité et présentent des limites dans l'accomplissement des tâches complexes nécessitant une planification. En effet, les architectures hybrides sont caractérisées par l'interaction qui peut émerger entre la planification et la réactivité. Dans ces architectures, on cherche à concilier une approche purement réactive et une approche plus délibérative orientée objectif, qui donnera au robot la capacité de s'adapter à un large éventail de tâches. Ainsi, le niveau de réactivité dépend de la décomposition hiérarchique du modèle de l'environnement, du niveau d'abstraction de l'information perceptuelle traitée et aussi du degré d'utilisation de la notion de concurrence dans la gestion de l'activité du robot. Cette concurrence repose sur des modules comportementaux non fonctionnels qui sont généralement des modules particuliers Perception/Décision/Action indépendants et concurrents directement liés au capteurs et qui permettent au robot de présenter un comportement donné.

1.3 Les systèmes Multi-robots

Un système Multi-robot est composé de plusieurs robots constituant des équipes pour l'accomplissement d'un ou de plusieurs buts généralement communs (coopératifs ou non). Les systèmes multi-robots présentent beaucoup d'avantages par rapport aux systèmes mono-robot classiques, ils peuvent augmenter la flexibilité et la robustesse du système en tirant profit du parallélisme et de la redondance inhérentes. La complexité d'une tâche dans certains environnements peut exiger l'utilisation des robots multiples, car les besoins nécessaires en ressources sont trop difficiles à satisfaire par un seul robot. Par conséquent, il y a souvent nécessité de distribuer les activités et l'intelligence [29] entre un ensemble d'entités physiques ou virtuelles.

1.3.1 motivations des Systèmes Multi-robots

Il y a plusieurs applications dans lesquelles les équipes de robots peuvent présenter des meilleures solutions qu'un seul robot. plusieurs raisons pour lesquelles les robots multiples pourraient être préférables, citons par exemples :

- Les systèmes multi-robots peuvent augmenter l'efficacité en distribuant ou en parallélisant la charge de travail exigée. Pour les missions qui peuvent être décomposées en sous missions indépendantes, différents robots ou sous-groupes de robots peuvent travailler simultanément sur les différentes sous-missions.

- Quelques types de tâches exigent souvent plusieurs robots quand un seul robot est incapable de l'exécuter tout seul. La manipulation d'un objet lourd, la surveillance simultanée de différents endroits et la formation de coalition ne peuvent pas être effectués par un seul robot.
- Les robots hétérogènes peuvent fonctionner ensemble pour produire des solutions plus flexibles. Les robots spécialisés peuvent souvent trouver des meilleures solutions aux problèmes pour lesquels ils sont mieux équipés, contrairement aux robots généralistes qui sont plus souples mais pas efficaces à accomplir chaque type de tâche.
- Les équipes de robots peuvent présenter plus de diversité et d'innovation dans les types de solutions qu'elles peuvent réaliser, il y a plusieurs façons de distribuer les tâches sur une équipe. D'ailleurs, les robots avec différents états ou connaissances peuvent trouver ou préférer différentes manières de résoudre des parties de la mission.
- La robustesse indisponible dans une solution mono-robot, si un robot tombe en panne, le reste du système peut compenser la perte.

Dans un système multi-robots, plusieurs entités coexistent dans un environnement commun. Cette coexistence s'effectue selon une approche *coopérative* (délégation de tâches, atteinte d'un but commun, partage de ressources, etc.) ou selon une approche *concurrentielle*. La majorité des projets de recherche traitent les systèmes multi-robots coopératifs. La problématique qui porte sur la coopération tente de répondre aux interrogations suivantes : qui fait quoi, quand, où, comment, avec quelles ressources, sous quelles conditions, et avec qui.

1.3.2 La coopération

La coopération est définie par :

> *la situation dans laquelle plusieurs robots travaillent ensemble en vue d'accomplir des buts globaux qui ne peuvent pas être réalisées par un seul robot ou dont l'exécution peut être améliorée en employant plusieurs robots, de ce fait obtenant des meilleures performances.*

1.3.2.1 Les protocoles de coopération

Les protocoles de coopération ou d'interaction sont des mécanismes basés sur la communication entre les agents. Un protocole de coopération est un mécanisme d'interactions automatisé que les agent-robot ont adopté pour prendre des décisions. En fonction de ces protocoles, les agents s'envoient des requêtes, des informations ou bien encore se délèguent des tâches. De manière générale, un protocole décompose une tâche

en un ensemble de sous-tàches qu'il distribue aux agents du système suivant une stratégie donnée et permet aux agents de coopérer. Dans les SMA, le CNP (Contract Net Protocol) [54] peut être vu comme un processus de délégation. Il est composé d'un ensemble de sous-tâches : envoi de l'offre, réponse à l'offre, etc. D'après [20], les protocoles doivent respecter les critères suivants :

- Éviter le recouvrement en ressources critiques.
- Allouer les tâches aux agents en fonction de leurs compétences.
- Informer les agents des différentes allocations.
- Allouer les responsabilités aux agents de manière à assurer une cohérence globale du système.
- Réallouer les tâches aux agents s'il est nécessaire de les effectuer de manière urgente.
- Allouer les tâches aux agents en cherchant à optimiser certains critères comme les coûts de communication.

Cependant, de nombreuses méthodes ont été proposées pour améliorer le CNP. En particulier, le CNP modifié permet de mener plusieurs négociations en parallèle avec des pré-allocations et des pré-rejets, ceci permettant d'éviter des réallocations trop coûteuses [5].

D'autres protocoles ont été proposés dans le cadre des langages d'interaction entre agents tel que FIPA-ACL (Agent Communication Language) [2] ; [3]. Les protocoles d'interactions ont bien entendus indispensables au sein des SMA lorsque les agents doivent prendre des décisions collectives.

Dans un système multi-robots, les problématiques de la robotique mono-robot sont considérablement augmentées par les interactions entre robots. Ces interactions peuvent représenter une menace pour l'intégrité du système ou tout au moins risquent-elles de dégrader ses performances. Cependant, les interactions peuvent également être mises à profit pour réaliser plus efficacement des tâches mono-robot (grâce aux redondance, partage de tâches et de ressources, etc) ou bien réaliser des tâches qui, par nature ne pourraient être réalisées par un unique robot(opérations distantes simultanées, etc). Pour mettre à profit les interactions dans un système multi-robots, il faut y introduire des mécanismes de *coordination*, afin de rendre cohérentes entre elles les actions des différents robots.

1.3.3 La coordination

En raison des tendances vers des systèmes multi-robots, la coordination multi-robots a excité une attention significative. La coordination est la synchronisation des actions en réponse à un événement de déclenchement (de la coordination). L'événement envoie un

signal aux entités coordonnées afin d'exécuter les actions qui doivent être coordonnées. cet événement de déclenchement peut être le résultat d'une occurrence d'événement dans l'environnement ou basé sur le temps. La coordination peut être implémentée de manière délibérative (comme résultats des actions synchronisées des entités multiples) ou réactive (comme dans les actions d'évitement de collisions entre deux entités mobiles). Dans le premier cas, la coordination peut exister comme effort partagé des entités ou comme séquence d'actions d'une seule entité (plan) où chaque action déclenche la prochaine. Le deuxième cas se produit quand les entités appellent le signal de synchronisation qui stimule alors une action dans chaque entité.

Dans les systèmes multi-robots, la coordination peut aider les robots à accomplir les tâches dont ils sont incapables de l'exécuter individuellement (comme le transport multi-robots d'objets [51]). Il peut également s'appliquer à une tâche de groupe telle que la formation de coalition [58]. Ces dernières exigent la coordination de la position de chaque robot avec les autres. Les signaux de synchronisation dans ces cas sont provoqués par la proximité à d'autres entités dans l'environnement. La coordination peut également être employée pour déterminer des rôles pour les robots en coopération. Dans une tâche commune, les rôles peuvent être utiles pour désambiguïser les actions des robots aux événements partagés.

1.3.3.1 Besoins de la coordination

Il y a trois raisons principales pour que les actions des robots multiples doivent être coordonnées, parce que :

- il y a des dépendances entre les actions des agents, l'interdépendance se produit quand les buts entrepris par différents agents sont liés l'un à l'autre, parce que les décisions locales prises par un agent ont un impact sur les décisions d'autre membre de la communauté ou en raison de la possibilité d'interactions nuisibles parmi les agents (par exemple, deux robots mobiles tentent de sortir d'une sortie étroite simultanément, ayant pour résultat une collision, des dommages aux robots et le blocage de la sortie).
- il y a un besoin d'atteindre des contraintes globales, les contraintes globales existent quand la solution développée par un groupe d'agents doit satisfaire certaines conditions pour qu'elle soit considérée réussie.
- Un seul individu n'a la compétence, les ressources ou l'information suffisantes à résoudre le problème entier, beaucoup de problèmes ne peuvent pas être résolus par des individus travaillant en isolation parce qu'ils ne possèdent pas l'expertise nécessaire, les ressources ou l'information nécessaires. Les exemples appropriés incluent les tâches de manipulation des objets lourds, surveillance simultanée de différents endroits. Il peut être impraticable ou indésirable de synthétiser de

manière permanente les composants nécessaires dans une seule entité, donc les alliances provisoires à travers la résolution coopérative des problèmes peuvent être la seule manière de procéder. L'expertise différente peut devoir être combinée pour produire un résultat en dehors de la portée de n'importe lequel des différents constituants (par exemple, dans le diagnostic médical, la connaissance de la maladie du cœur, des désordres de sang et les problèmes respiratoires peuvent devoir être combinés pour diagnostiquer la maladie d'un patient). Les différents agents peuvent avoir différentes ressources (par exemples capacité de traitement, mémoire et communications) qui sont nécessaires pour résoudre un problème complexe. Finalement, les différents agents peuvent avoir différentes informations ou points de vue d'un problème (par exemple dans les systèmes concurrents technologiques, le même produit peut être regardé d'une perspective de conception, de fabrication et de vente).

De nombreuses approches ont été proposées pour coordonner les actions ou les buts des agents. Une revue détaillée de ces approches et des applications des systèmes multi-robots est proposée dans [21] et [15]. Les travaux majeurs reposent sur les mécanismes de planification/ordonnancement [16], allocations de tâches [18] et de négociation [50].

1.3.3.2 Modes de coordination

On peut identifier deux modes fondamentaux de coordination :

– Coordination par ajustements mutuels : c'est la forme de coordination où, les entités s'accordent pour partager des ressources et des tâches pour atteindre un but commun : aucune entité n'a de contrôle sur les autres et le processus de décision est conjoint.

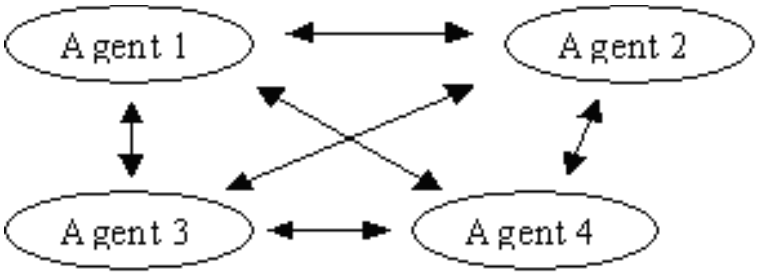


FIG. 1.3 – Exemple de coordination par ajustement mutuel

– Coordination dirigée (ou par leadership) : il y a une relation hiérarchique entre les agents, certains agents exercent alors un contrôle sur d'autres.

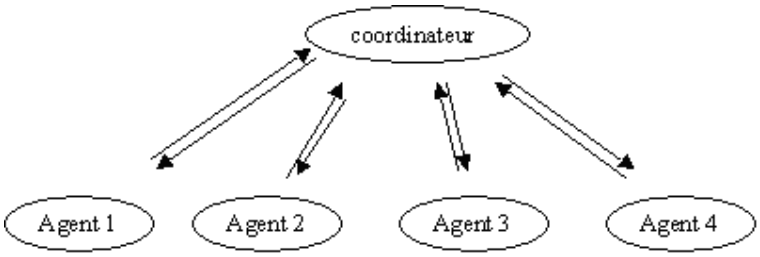


FIG. 1.4 – Exemple de coordination dirigée

Les différents modes de coordination implémentés dans les systèmes multi-robots sont tous des dérivés d'un ou des deux modes.

1.3.4 La communication

La communication est un aspect important pour atteindre la coordination des robots. La communication est la livraison d'information prévue pour provoquer une mise à jour des connaissances et/ou une réponse du destinataire. La communication par définition implique plus d'une entité qui rend les applications multi-robots plus performantes.

Il y a plusieurs aspects à considérer en établissant la communication dans un système multi-robots. Ceux-ci incluent le rôle, la forme, le protocole, et les problèmes d'inclure la communication dans la conception des robots.

1.3.4.1 Rôle de la communication

La communication peut être appliquée de plusieurs manières ou pour plusieurs raisons :

- ***Partage de la connaissance :*** la communication peut être employée pour compléter les tâches et les informations d'état perçues. L'information des capteurs peut être communiquée pour améliorer la vue d'un robot de son environnement ou pour améliorer l'exactitude de localisation de robot. Par exemple, dans ALLIANCE [45], la communication est employée pour compléter les informations perçues sur les tâches entre les robots coopérants. Elle fournit des informations concernant les actions des autres robots qui ne peuvent pas être senties dans l'équipe, ainsi un robot peut prendre les décisions d'effectuer la tâche qui ne sera pas en conflit avec d'autres.
- ***Résolution de conflit :*** la communication peut être employée pour prévenir ou résoudre les conflits (par exemple de ressources) entre les robots. Les mécanismes de prévention incluent la négociation pour l'allocation de tâches ou de rôles [51].

Pour ce faire, les robots envoient des offres basées sur l'information interne au système qui les utilise pour assigner les tâches ou les rôles. La résolution de conflit implique également un mécanisme d'arbitrage qui est appelé quand les conflits de ressources se produisent entre les robots.

– ***Améliorer la coordination :*** dans certaines tâches coordonnées, la communication peut être utile. Le succès de la coordination dépend de la synchronisation des actions, les entités impliquées peuvent employer la communication en tant que déclenchement à effectuer leurs actions coordonnées. Ceci peut être réalisé par les différentes formes de communication décrites dans la section ci-après. Explicitement, la communication peut être employée par les robots pour partager l'information concernant l'activité coordonnée comme le signal de synchronisation. Implicitement, la communication peut être fournie par les effets observables d'actions appliquées par les robots coordonnés dans un environnement partagé (comme les objets partagés). Les robots peuvent développer des modèles de chacun d'autre pour augmenter leurs efforts de coordination.

1.3.4.2 Forme de communication

La communication entre les robots peut être classifiée en tant qu'explicite ou implicite.

– *La communication explicite* est la transmission délibérée d'information. Le type le plus fondamental est la signalisation lorsque le robot fait une action discrète, telle que l'allumage de la lumière, faisant un son ou effectuant un mouvement délibéré, en prenant en compte la portée des signaux utilisés dans l'environnement. Bien que simple, ils peuvent avoir un impact puissant, particulièrement sur la synchronisation de la coordination. Des formes plus complexes de communication explicite incluent l'envoie des données par l'intermédiaire des transmissions radio ou des liens filaires. Les données peuvent être de petit nombre de bits aux paquets d'information et d'images. La communication explicite peut être directe (robot à robot) ou indirecte (diffusion) selon son application.

– *La communication implicite* implique de communiquer l'information volontairement ou involontairement en laissant l'évidence de l'activité, par exemple les traces des pas des animaux sont des messages qui peuvent être utilisées par un animal pour retrouver ses congénères, comme elles peuvent servir à les pister par un chasseur. Le fauchage d'une pelouse, si la pelouse est à moitié fauchée, il sera évident à n'importe quel robot le rencontrant que des travaux ont été menés. Des décisions peuvent être basées sur cette compréhension

L'observation est souvent abordée dans plusieurs systèmes en tant que communication implicite [9], mais également peut être vue comme forme de communication [26],

où le robot doit observer et raisonner sur les actions des autres comme l'observation qui a lieu dans une formation de coalition ou le mouvement ensemble. Chaque robot dans une formation doit observer sa position relative aux autres pour maintenir sa position. L'observation peut également être employée pour gagner des habiletés d'un similaire ou d'un maître. L'action du robot peut ou peut ne pas être mise au profit d'un robot observant.

La communication peut être active ou passive. Dans le premier cas, la communication est déclenchée périodiquement (basée sur le temps) et dans le deuxième cas, elle est basée sur les événements, c'est à dire à l'occurrence d'un événement (par exemple, le changement d'état d'un robot tel que la fin d'exécution d'une tâche, etc)

1.3.4.3 Protocole de communication

Un protocole est nécessaire pour déterminer l'efficacité et l'opportunité de communication en contrôlant et en interprétant l'information communiquée. Il est important d'établir une structure de données commune entre les entités communicantes, ainsi ils peuvent interpréter l'information reçue correctement, même si les robots se diffèrent dans d'autres aspects de leur architecture.

Pour la communication explicite, le protocole doit valider l'utilisation de chaque robot de la méthode de communication pour qu'il n'ait aucun conflit. Il y a plusieurs manières pour implémenter le protocole avec des exemples de la gestion des réseaux d'ordinateurs fournissant un guide car les problèmes sont analogues. Le besoin de protocole est évident pour la communication explicite mais pas pour la communication implicite et l'observation. Les entités employant ces formes doivent pouvoir interpréter des informations de plus en plus complexes des capteurs de l'environnement. Ceci exige le protocole de déterminer le lien de communication par lequel les messages doivent passer.

Pour que la communication augmente les performances des systèmes multi-robots que des systèmes distribués, les problèmes leur inhérents doivent être traités. Parmi les problèmes communs, nous citons :

- ***Mise en échelle (Scalability) :*** le protocole de communication doit pouvoir être efficacement utilisé par plusieurs robots. La bande passante et le protocole devraient pouvoir supporter des robots supplémentaires sans diminution des performances.
- ***Goulot d'étranglement :*** ce problème se produit quand plusieurs entités tentent de communiquer en même temps. Le protocole devra être capable de gérer le support de communication. Sinon, l'information peut être mise en attente (et par conséquent périmée ou perdue).

- ***Bruit :*** la communication explicite peut être soumise au bruit comme n'importe quel type de communication par un moyen de communication. Pour la communication implicite et l'observation, le bruit peut être des distractions de contexte, ou des différent modèles de communication utilisés. Par conséquent, l'information sera non interprétable et donc perdue ou bien mal interprétée.
- ***Mystification :*** la communication entre les entités négociant peut être trompeuse. Par exemple, un robot indique qu'il puisse effectuer une tâche qu'un autre robot est dépendant, alors qu'il ne puisse pas. L'autre robot peut alors être bloqué en attendant l'accomplissement de la tâche.
- ***Latence :*** il y a un temps pris entre l'instant de l'envoi d'information et la fin de son interprétation par le mécanisme de réception. En conséquence, les données peuvent être périmées, ce qui va être nuisible s'il porte des informations sur des objets dynamiques.
- ***Échec :*** les systèmes dépendants d'un lien de communication (par exemple le contrôle centralisé) souffrent si ce dernier échoue. Les robots peuvent perdre partiellement ou complètement la connaissance et devenir déclinés et incapables d'effectuer leurs tâches.

1.3.5 Architectures de contrôle pour système multi-robots

Il existe diverses architectures développées récemment pour contrôler une équipe de robots. Les architectures de "Subsumption" de Brooks [12] ou encore la robotique dite "Behavior-Based Robotics" [8] ; [37], les architectures hybrides [40] ; [27] en sont des bon exemples. L'architecture ALLIANCE/L-ALLIANCE est également un très bon exemple de modèle couramment référencé encore aujourd'hui. Nous présentons quelques architectures de références récentes, bien adaptées aux systèmes modernes.

1.3.5.1 ALLIANCE

ALLIANCE [45] est une architecture pour la coopération tolérante aux fautes au sein d'un système Multi-robots hétérogènes. C'est une architecture distribuée qui impose certaines règles d'implantation de façon à assurer une certaine tolérance aux fautes. La Figure suivante résume schématiquement le fonctionnement de cette architecture.

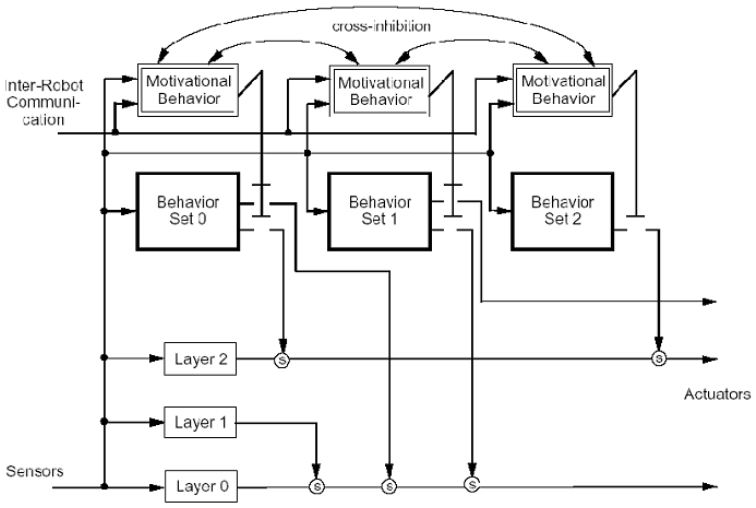


Fig. 1.5 – Architecture ALLIANCE

Elle implémente à un premier niveau les principes de l'architecture de Subsumption, par l'utilisation de comportements réactifs (fonctions reliant directement les capteurs aux actionneurs). Ensuite, à un deuxième niveau, le principe d'"Ensemble de Comportement" (Behavior Set) est utilisé dont le but est de contrôler l'activation ou l'hibernation de groupes de comportements situés dans le premier niveau. Ce deuxième niveau permet donc à l'architecture de se reconfigurer dynamiquement, ce qui permet d'adresser la problématique de la coopération de systèmes multi-robots hétérogènes. Pour arriver à établir une telle coopération, l'architecture utilise un troisième niveau, basé sur le principe de "Motivational Behavior". Ce troisième niveau permet tout simplement d'activer/désactiver le deuxième niveau, celui des "Behavior Sets", mais en plus de considérer les informations sensorielles comme le font les autres niveaux, il considère des informations de motivation interne permettant entre autre l'inhibition latérale des motivations (cross-inhibition), et finalement des informations provenant de la communication inter-robots.

Une extension de l'architecture ALLIANCE, nommée L-ALLIANCE, a été développée de façon à permettre l'implantation de mécanismes d'apprentissage par renforcement [33] (reinforcement learning) au sein de l'architecture ALLIANCE. Donc, on remarque la distinction avec l'architecture originale, qui correspond à l'ajout du principe de Monitor. Brièvement, le Monitor est en mesure d'influencer les comportements de la troisième couche, celle des "Motivational Behavior".

1.3.5.2 CAMPOUT

CAMPOUT (Control Architecture for Multirobot Planetary OUTposts)[28] est une architecture développée par la *NASA* (National Aeronautics and Space Administration) au "Jet Propulsion Laboratory du California Institute of Technology". c'est une architecture hybride (réactive et délibérative). CAMPOUT est à priori destinée pour les systèmes multi-robots à perception et cognition distribuées. Elle part du principe que chacun des robots est une entité indépendante et que la prise de décision se fait de façon entièrement et strictement distribuée.

Cette architecture est composée de trois couches. La couche supérieure permet la planification, l'allocation et le monitorage hiérarchiques de tâches. La seconde couche permet la composition de différentes classes de comportements. La dernière couche de cette architecture permet de faire le lien entre la couche des comportements et le matériel du système robotisé (capteurs et actionneurs). La Figure (Fig.1.6) schématise le fonctionnement de l'architecture de CAMPOUT. On peut y retrouver les différentes classes de comportements ainsi que les différents outils disponibles pour chacune des tâches. Ces outils semblent intéressants puisqu'ils offrent une interface évoluée pour définir la composition des éléments des différentes couches, soit par des langages de haut niveau ou par des outils graphiques.

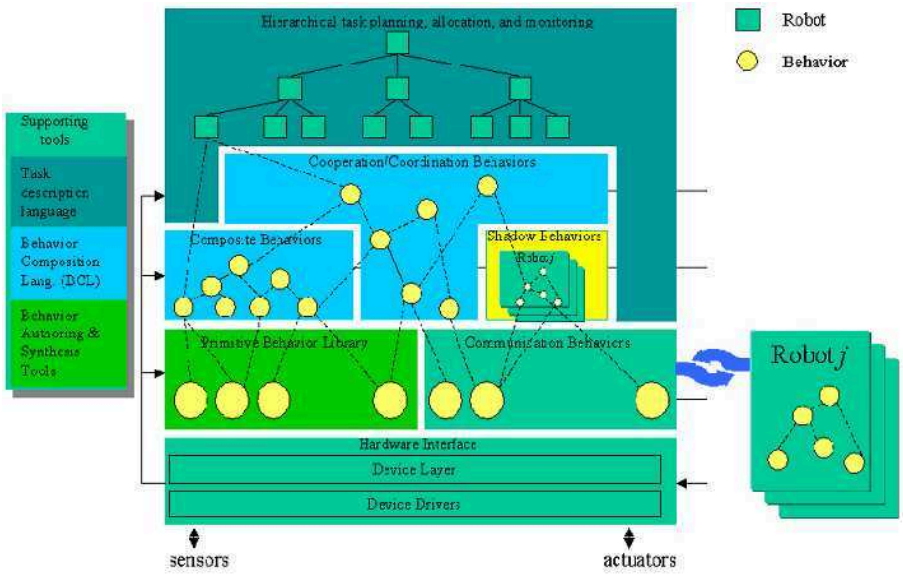


FIG. 1.6 – Architecture CAMPOUT

L'architecture étant à la base conçue pour les systèmes multi-robots coopératifs, elle intègre certains éléments pertinents à ces systèmes, comme une infrastructure de communication basée sur des sockets TCP/IP. Au niveau des comportements, trois classes

sont importantes pour ces systèmes, soient les comportements de communication (Communication Behaviors), les comportements images (Shadow Behaviors) qui permettent la représentation des comportements des coéquipiers et finalement les comportements de coopération/coordination (Cooperation/ Coordination Behaviors). Les robots d'un système peuvent utiliser des comportements primitifs et composés (Primitive Behaviors and Composite Behaviors) pour accomplir leur travail. La gestion de l'utilisation des divers comportements est assurée par la couche supérieure de l'architecture (Hierarchical task planning, allocation and monitoring).

1.3.5.3 CLARAty

CLARAty [40] est une évolution de l'architecture à trois-couches, qui fournit une large gamme de fonctionnalités robotiques et simplifie l'intégration de nouvelles technologies sur les plates-formes robotiques. CLARAty a été conçue spécifiquement pour des applications de contrôle robotiques en espace. Elle comporte une couche fonctionnelle de primitifs robotiques, couplée avec une couche décisionnelle de planification et d'exécution ; chacune de ces couches contient une hiérarchie des composants s'étendant du plus élémentaire au plus intelligent.

- La couche décisionnelle décompose les tâches de haut niveau en sous tâches plus simples ordonnancées selon les contraintes temporelles et de l'état du système. Elle accède aux fonctionnalités appropriées de la couche fonctionnelle pour les réaliser. La Figure (Fig.1.7) montre une représentation très simplifiée de la couche de décision.

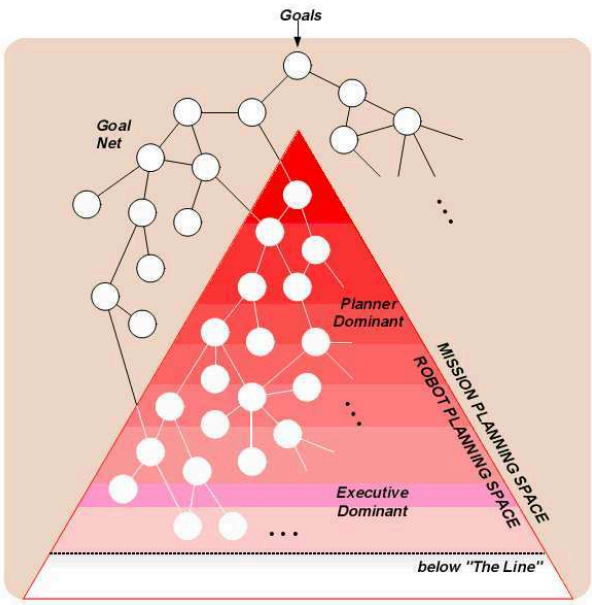


Fig. 1.7 – Couche Décisionnelle de l'architecture CLARAty

– La couche fonctionnelle (Functional Layer) fournit un ensemble de fonctionnalités standards et génériques de robot qui sont connectées aux capteurs et actionneurs du système. Ces fonctionnalités sont organisées en hiérarchie de classes logicielles des composantes robotiques comme dans les systèmes orientés objets, La Figure (Fig.1.8) donne une description simplifiée de la hiérarchie d'objet trouvée dans la couche fonctionnelle.

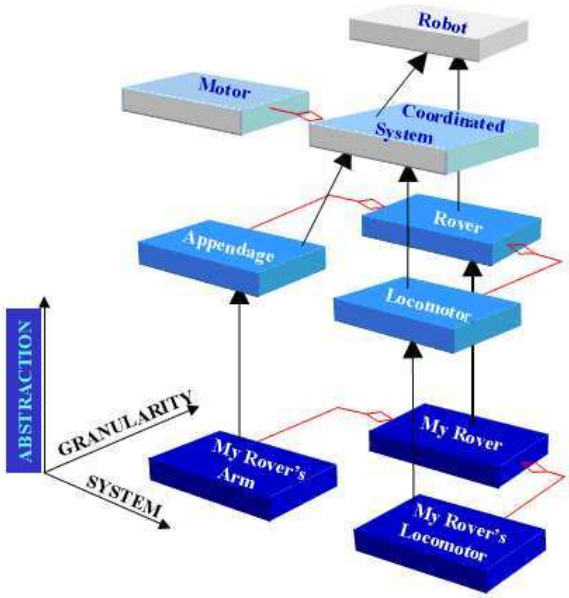


FIG. 1.8 – Exemple illustrant des concepts d'hiérarchie d'objet et d'héritage de classe dans l'architecture CLARAty

Dans ce diagramme, une quatrième dimension d'abstraction a été ajoutée pour illustrer la structure d'héritage des classes dans la couche fonctionnelle. En bas, l'objet Rover agrège les objets bras et locomoteur, Tandis que ces objets comportent un système spécifique MyRover, chacun est dérivé des classes parentes qui sont beaucoup plus générales.

L'interaction entre les deux couches, peut être comprise en considérant la création et l'exécution des activités sur une ligne-temporelle. La Figure (Fig.1.9) montre les deux couches avec la séquence d'activation en vert.

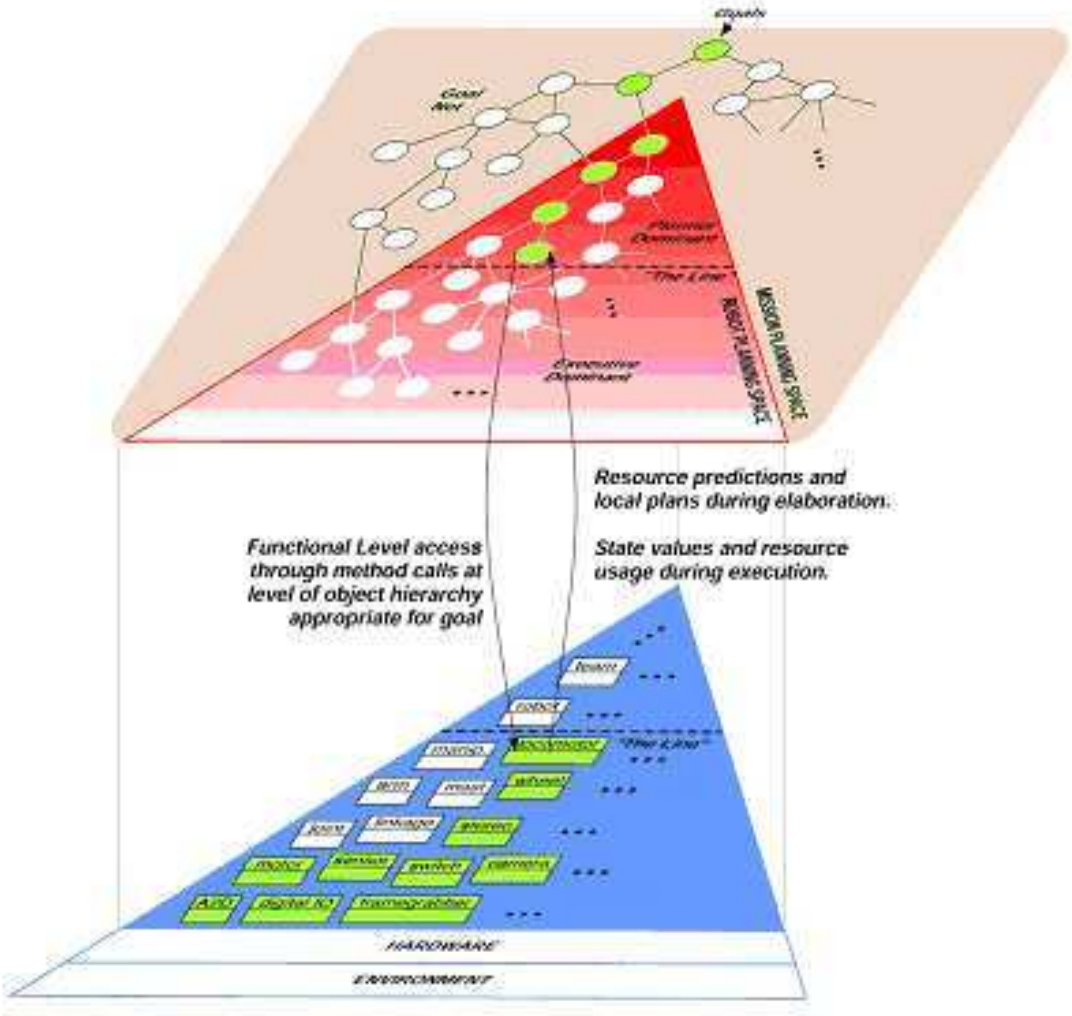


FIG. 1.9 – Relation entre la couche fonctionnelles et la couche décisionnelle dans l'architecture CLARAty

Dans la couche décisionnelle, les buts de haut niveau sont décomposés en buts subordonnés jusqu'à ce qu'il y ait un certain but de niveau inférieur qui peut accéder directement à la couche fonctionnelle. Pendant la planification et l'ordonnancement, ce processus se produit avec les requêtes d'utilisation de ressources et de plans locaux. Si une information de haute fidélité est demandée par la couche fonctionnelle, par exemple, si les marges de ressource sont petites, alors l'objet de la couche fonctionnelle peut également devoir accéder à ses subordonnés pour améliorer les prévisions.

1.3.6 Applications des systèmes multi-robots coopératifs

Il existe de nombreux projets de recherche portant sur la mise en pratique de systèmes multi-robots coopératifs. La surveillance, le transport, la construction de sites habitables, l'extraction de ressources, l'exploration et le sauvetage, en sont de bons exemples. Une attention particulière est portée à l'exploration extra-planétaire.

1.3.6.1 Robots pour exploration extra-planétaire

L'exploration extra-planétaire représente une avenue de développement très intéressante pour la robotique mobile autonome. Ainsi, des ressources considérables déployées par les agences spatiales internationales pour développer plusieurs technologies robotiques.

Le plan d'exploration du gouvernement américain vise dans un premier temps le développement de missions habitées pour la Lune vers 2015 dans le but de développer des technologies pour des missions habitées pour Mars vers 2030. Avec un temps de transmission de près de 40 minutes aller-retour Terre-Mars, la télé-opération est indésirable pour un système robotisé déployé sur Mars. L'autonomie des systèmes développés constitue donc un élément-clé pour le succès des missions robotiques martiennes. Ainsi, plusieurs travaux de recherche se font sur les architectures de contrôle à capacités délibératives, CAMPOUT et CLARAty en sont de bons exemples, mais il y a également plusieurs systèmes expérimentaux qui simulent des systèmes robotisés entièrement autonomes en mission sur Mars. Le projet "Robot Work Crew" [53] développé par la NASA, adresse la problématique de la coopération multi-robots pour l'exécution de tâches en couplage rigide (manipulation coopérative) et la coopération de groupes homogènes et hétérogènes de robots en considérant un environnement très difficile tel que le sol martien. Leur principale réalisation expérimentale est la collaboration de deux robots mobiles pour le déploiement d'une tente photovoltaïque en utilisant l'architecture CAMPOUT [28].

1.3.6.2 Robots au service de l'homme

Les systèmes multi-robots déployés dans des environnement habité par l'être humain (par exemple en milieu industriel) pour l'aider à exécuter ses tâches. plusieurs projets de recherche traitent des applications bien concrètes et présentent des résultats expérimentaux prometteurs. Par exemples, les autoroutes autonomes qui sont en développement depuis quelques années [52].

Le projet Martha [6] est un bon exemple de système multi-robots coopératif au service de l'homme. L'objectif de ce projet est de développer des techniques d'opération

et de contrôle d'une flotte de robots mobiles autonomes pour le transport de conteneurs dans les ports maritimes, les aéroports et les cours de triage. Dans l'architecture de contrôle proposée dans ce projet, on retrouve trois niveaux, ce qui constitue donc une architecture à trois couches conventionnelle.

Le niveau décisionnel est organisé en trois sous couches : la première en bas pour la gestion de la coordination avec les autres robots, appelée "Coordination Layer", une seconde pour la gestion de l'exécution de tâches, c'est la "Task Layer", la troisième en haut pour la gestion de l'exécution des missions, c'est une "Mission Layer". Chaque sous couche est composée d'une relation superviseur-planificateur afin de gérer et de raffiner l'exécution. Au-dessus de cette architecture se trouve une "Central Station" qui se charge seulement d'établir la mission à accomplir par chaque robot, donc, on laisse l'autonomie décisionnelle au système distribué. La mission reçue du station centrale est ensuite décomposée sous forme de tâches par la couche de Mission. Chaque tâche est ensuite décomposée en plans par la couche Tâche, puis ces plans seront coordonnés par la couche de Coordination. Pour assurer une bonne coordination des plans (la coordination de l'équipe de travail), la couche de coordination utilise le concept de "Plan-Merging Protocol". Avec ce protocole qui utilise une communication inter-robot, un robot est en mesure de fusionner son plan avec ceux des autres robots et donc de proposer un nouveau plan coordonné. Les plans sont finalement exécutés grâce au niveau de contrôle et au niveau fonctionnel.

En général, les stratégies de coordination multi-robots adoptent soit une approche centralisée, où un seul agent-robot planifie et/ou coordonne tout le groupe de robots, soit une approche distribuée, où chaque robot est responsable de sa propre planification. Les approches centralisées souffrent de quelques problèmes, tel qu'elles ne sont pas valables pour les grands groupes de robots, répondre lentement aux changements de l'environnement, besoins de communication lourdes, système fragile avec un seul point d'échec. L'avantage principal de ces approches est qu'elles peuvent produire des plans globalement optimaux et cohérents. Tandis que la plupart des approches distribuées peuvent écarter les problèmes inhérents aux approches centralisées, leur inconvénient majore est qu'elles puissent produire des plans globaux sous optimaux.

1.3.7 conclusion

Nous nous sommes focalisé dans cette deuxième partie sur l'intégration d'un robot mobile dans un groupe de robots coopératifs. Ceci implique entre autres, que le contrôle du robot en question ne dépendra plus uniquement de ses propres perceptions et buts, mais devra tenir compte aussi d'un certain nombre de considérations liées à l'évolution globale du système. Ce qui va ajouter certainement un niveau de complexité au contrôle

du robot mobile.

1.4 Conclusion

Dans ce premier chapitre, nous avons présenté et analysé les résultats des travaux menés sur les architectures de contrôle et de décision des systèmes multi-robots mobiles autonomes. Le développement de ces architectures passe par la conception de ses composantes et leur organisation. Dans la première partie du chapitre, nous avons identifié trois types d'architectures de contrôle : les architectures comportementales, les architectures délibératives et les architectures hybrides. La première permet de doter le robot des capacités à réagir en temps réel aux changement de l'environnement, ce qui lui confère toute sa robustesse. La deuxième confère au robot des capacités de planification et d'ordonnancement permettant la décomposition hiérarchique d'une mission à travers des niveaux d'abstraction qui lui permettent de s'adapter à un large éventail de tâches. L'approche hybride cherche à concilier une approche comportementale et une approche délibérative orientée but et caractérisée par l'interaction entre les comportements (planification et réaction) de ces deux approches. La flexibilité dans cette approche dépend du niveau de réactivité utilisée qui est lié à la décomposition hiérarchique du modèle de l'environnement et surtout à la notion de concurrence dans la gestion des comportements du robot. La deuxième partie, propose une vision d'ensemble des caractéristiques et techniques associées à la mise en pratique des systèmes multi-robots.

L'approche d'ordonnancement proposée dans ce mémoire s'inspire de techniques de coordination de robots, en considérant la nature des tâches jointes et des interactions envisageables dans un système de robots mobiles. Le chapitre suivant pressente l'état de l'art sur l'ordonnancement et l'allocation distribués de tâches qui ont relation avec notre approche.

Chapitre 2

État de l'art sur l'ordonnancement et l'allocation de tâches

2.1 Introduction

Les problèmes d'ordonnancement sont très étudiés par le domaine des recherches opérationnelles et utilisé dans plusieurs domaines, tel que l'économie, les mathématiques, l'IA, la production (ateliers), l'informatique (processus). l'automatique (robotique), etc.

Les problèmes d'ordonnancement dans les systèmes robotiques distribués consistent à ordonnancer un ensemble de tâches (ou opérations) devant être exécutées par une équipe de robots. Les tâches peuvent ou peuvent ne pas être liées entre elles par des contraintes conjonctives (contraintes de précédences) et des contraintes disjonctives (contraintes de ressources). Le but du problème est généralement de déterminer un ordonnancement de durée totale minimale. De ce fait, l'ordonnancement est un problème d'optimisation, c'est à dire il doit produire une solution (séquence) optimale.

Dans un environnement dynamique, de nombreux aléas peuvent perturber les prévisions d'ordonnancement préalable. Entre la construction de l'ordonnancement et sa réalisation (ou exécution) effective, de nombreux événements non contrôlés (nouvelles tâches) et indésirable (aléas) peuvent intervenir à n'importe quel instant pendant l'exécution. Dans une mission multi-robots, ils peuvent s'exprimer par la terminaison en avant ou en retard d'une tâche, panne d'un robot, une ressource manquante, une nouvelle tâche, annulation d'une tâche, etc. Pour faire face à ces aléas, le système d'ordonnancement doit introduire des corrections (ou des réparations), c'est à dire réordonnancer les tâches des robots. Cette activité de réordonnancement est en fait, au cœur de la pratique et doit être prise en compte tout particulièrement.

2.2 Définitions

Dans ce qui suit, nous présentons les notions qui sont le plus souvent référencées pour traiter des problèmes d'ordonnancement dans les systèmes multi-robots.

2.2.1 Ordonnancement et allocation de tâches

Nous avons remarqué dans la littérature que, les termes ordonnancement et allocation de tâches concernent des domaines fortement liés, l'allocation de tâches est souvent confondue avec l'ordonnancement de tâches dans les applications multi-robots, on parle alors d'ordonnancement local (ou interne) et distribué (ou global). Nous estimons que l'appellation ordonnancement local exprime bien le partage des ressources d'un robot par un ensemble de tâches. Cependant, l'ordonnancement distribué, étant donné qu'il s'agit d'allouer les tâches à un groupe de robots. Nous avons donc adopté les définitions suivantes :

- Ordonnancement de tâches : détermination d'un ordre d'exécution des tâches selon des critères bien spécifiés, ou bien, construction d'un plan de tâches cohérent pour un robot.
- Réordonnancement de tâches : la séquence déterminée au préalable est mise à jour (réordonnancée) en fonction des nouvelles contraintes ou conditions de la mission.
- Allocation de tâches : attribution d'un ensemble de tâches à un groupe de robots en respectant et optimisant certains critères.

2.2.2 Ordonnancement et planification

Nous avons remarqué aussi que, dans le sens commun, les deux termes : planification et ordonnancement, sont souvent employés pour désigner la même chose : construire des "plannings" d'exécution des tâches en utilisant un ensemble de moyens (robots, machines, ressources, etc). Néanmoins, souvent une différence est à faire.

Pour enlever les confusions de vocabulaire, nous admettrons que la frontière entre la planification et l'ordonnancement est liée au nombre de niveaux d'abstractions :

- En ordonnancement, il y a un seul niveau d'abstraction, les tâches sont considérées comme atomiques et les données sont fixées.
- En planification, il existe plusieurs niveaux, à chaque niveau pouvant être associées différentes échelles de temps. Ceci implique une idée de décomposition, à la fois temporelle et hiérarchique. La planification complète d'une mission multi-robots peut ainsi la décomposer en plusieurs sous missions. Mais à l'intérieur de chaque sous mission on retrouvera un ou plusieurs problème d'ordonnancement

de tâches. Cette idée de tâche atomique (au moins dans l'énoncé du problème) est une hypothèse implicite mais fondamentale.

En Intelligence Artificielle, la planification se focalise sur la sélection d'actions parmi différentes alternatives, il s'agit de décider la nature des tâches à entreprendre et de les ordonner pour satisfaire un ou plusieurs objectifs. On parle donc de génération de plans. L'ordonnancement s'attache, lui, au placement temporel (date de début, date de fin, etc) et spatial (allocation de ressources) des tâches qui doit permettre la réalisation de ces actions. De manière générale, la planification et l'ordonnancement sont des moyens par lesquels une organisation ou bien une entité, peut choisir une suite d'actions pour atteindre un objectif donné.

2.2.3 Ordonnancement et réordonnancement

Réordonnancer un ensembles de tâches, se rapporte à réparer un ordonnancement établi a priori de tâches, mais à cause de la dynamique du système, des modifications des conditions et des événements indésirables (perturbations) se produisent pendant l'exécution de cet ordonnancement.

Á partir de ces définitions, on peut noter que, le raisonnement sur le temps et les ressources est au cœur des problèmes d'ordonnancement. Les problèmes d'ordonnancement sont principalement des problèmes d'optimisation, tel qu'il est souvent facile de trouver un ordonnancement réalisable (acceptable) en séparant les tâches dans le temps avec un écart suffisant. Mais en trouver un bon n'est pas évident.

2.3 Éléments d'un problème d'ordonnancement

Les principaux éléments d'un problème d'ordonnancement sont :
- Un ensemble de tâches.
- Un ensemble de ressources pour effectuer les tâches.
- Des contraintes sur les tâches et les ressources.
- Un ou plusieurs critères d'optimisation.

2.3.1 Les tâches

Une tâche est une entité globale (décomposable en sous tâches, telle que Goto(x, y)) ou élémentaire (non décomposable, telle que TurnLeft)).

Chaque tâche est définie par un nom T_i et par des caractéristiques temporelles :
- Une date de début d'exécution s_i (Start Time).

- Une date de disponibilité ou date de début au plus tôt r_i (Release Time).
- Un temps d'exécution ou une durée opératoire p_i (Processing Time).
- Une date de fin d'exécution e_i (End Time).
- Une date de fin au plus tard d_i (Deadline).

Deux types de tâches sont distingués :

- Les tâches préemptives (ou morcelables) qui peuvent être exécutées en plusieurs fois.
- Les tâches non préemptives qui sont exécutées en une seule fois et ne peuvent pas être interrompues avant qu'elles soient complètement terminées.

2.3.2 Les ressources

Une ressource est un moyen technique ou humain utilisé pour réaliser une tâche. Ce moyen technique est donc nécessaire et indispensable pour le bon fonctionnement d'une mission multi-robots

Dans une application multi-robots, plusieurs types de ressources sont distingués :

- Les ressources renouvelables, qui, après avoir été allouées à une tâche, elle redevient disponible à la fin de cette tâche pour les autres tâches et qui peuvent être réutilisées (robot, couloir, etc).
- Les ressources consommables, qui, après avoir été allouées à une tâche, ne sont plus disponibles, et sont donc épuisées (énergie, etc).

Ces ressources peuvent être classées d'un autre point de vue :

- Les ressources disjonctives (ou non partageables) qui ne peuvent exécuter qu'une seule tâche à la fois (robot mono-tâche),
- Les ressources cumulatives (ou partageables) qui peuvent exécuter plusieurs tâches simultanément (robot multi-tâches).

2.3.3 Les contraintes

Les contraintes représentent les conditions à respecter lors de la construction de l'ordonnancement pour qu'il soit réalisable. Elles rendent les problèmes d'ordonnancement plus difficiles car il faut les respecter durant leur résolution.

Généralement les tâches sont liées par des contraintes temporelles ou/et de ressources.

1. **Les contraintes temporelles :** elles représentent des restrictions sur les valeurs que peuvent prendre certaines variables temporelles d'ordonnancement. Á titre d'exemple, nous pressentons une liste non exhaustive de contraintes souvent rencontrées :

- Les contraintes de précédence : lorsque la tâche T_i doit précéder la tâche T_{i+1}, c'est à dire que la tâche T_{i+1} ne peut commencer qu'après la fin de la tâche T_i. Elle est notée "$T_i > T_{i+1}$", on notera G = (X, E) le graphe orienté associé, X étant l'ensemble des tâches et E l'ensemble des précédences. G est appelé le graphe de tâches.
- Les contraintes d'échéance : lorsque certaines tâches doivent être achevées avant une date préalablement fixée ;
- Les contraintes des dates au plus tôt (ou de localisation temporelle) : qui sont liées à l'indisponibilité de certains facteurs nécessaires pour commencer l'exécution des tâches. Ce type de contrainte peut être exprimé pour une tâche T_i par : $s_i = r_i$ où s_i est la date de début d'exécution de T_i et r_i la date de début au plus tôt de T_i.

2. **Les contraintes de ressources :** elles représentent des restrictions sur l'utilisation de certaines ressources limitées. Il y a deux types :
   - Contraintes disjonctives : deux tâches T_i et T_j sont liées par une contrainte disjonctive si T_i doit être exécutée soit avant T_j, soit après : $s_j - s_i \geq p_i$, ou $s_i - s_j \geq p_j$, par exemple, deux robots ne peuvent pas traverser simultanément le même couloir).
   - Contraintes cumulatives : dans ce type de contraintes, on doit respecter le nombre maximal des tâches exécutables simultanément sur la ressource, par exemple, au plus deux robots peuvent traverser simultanément un couloir.

2.3.4 Critères d'optimisation

Un critère correspond à des exigences qualitatives et quantitatives à satisfaire permettant d'évaluer la qualité de l'ordonnancement établi. Les critères usuels sont :

- Minimisation de la durée totale de l'ordonnancement (makespan) C_{max}.
- Minimisation du nombre de tâches qui ne respectent pas leurs contraintes temporelles.
- Minimisation de la charge de communication.
- L'équilibrage de la charge d'exécution de tâches.
- La minimisation du retard global.
- Minimisation d'un coût.
- Etc.

2.4 Classification des problèmes d'ordonnancement

Un problème d'ordonnancement est généralement décrit par un ensemble T de n tâches $T_1, T_2, .., T_n$, qui seront exécutées par un ensemble A de m agents (robots, machines, opérateur, etc) $A_1, A_2, .., Am$, chacun d'eux peut exécuter un sous ensemble de tâches, un graphe de tâches G(E, T), tel que les tâches de T représentent les états (ou les sommets) et E est l'ensemble d'arcs représentant les contraintes entre les tâches. $E(T_i, A_i)$ représente le temps d'exécution de la tâche T_i par l'agent A_i, et $C(T_i, A_i, A_j)$ représente le délai de communication pour l'envoi des résultats de la tâche T_i de l'agent A_i à l'agent A_j.

Un problème d'ordonnancement peut être modélisé formellement de différentes manières, par exemple une modélisation de quatre variables $\alpha|\beta|\gamma|\delta$ spécifiant les différents aspects inhérents aux problèmes d'ordonnancement. Tel que :

α représente l'environnement des agents, cet environnement peut être constitué d'un seul ou de plusieurs agents fonctionnant en parallèle. Alors, $\alpha \in \{P, Q, R\}$, d'où $\alpha = \{P, Q, R\}.m$, c'est à dire qu'on a m agents.

- P : des agents identiques parallèles : les agents fonctionnent en parallèle à une même vitesse, et par conséquent le temps d'exécution d'une tâche est le même dans tous les agents, donc il est indépendant de n'import quel agent, $E(T_i, A_i) = f(T_i)$.
- Q : des agents uniformes parallèles : les agents fonctionnent en parallèle à des vitesses distinctes, alors le temps d'exécution d'une tâche n'est pas le même pour tous les agents, donc il est uniformément dépendant de la vitesse de chaque agent, $E(T_i, A_i) = f(A_i)$.
- R : des agents indépendants parallèles : les agents fonctionnent en parallèle, et le temps d'exécution d'une tâche est une fonction arbitraire de la tâche choisie ainsi que de l'agent choisi pour l'exécuter, $E(T_i, A_i) = f(T_i, A_i)$.

β représente les caractéristiques des tâches, telles que les contraints de précédence et les coûts d'exécution. $\beta \in \beta_1, \beta_2, \beta_3$, $\beta_1 \in \{\phi, PREC, TREE\}$, $\beta_2 \in \{\phi, E(T) = 1\}$, $\beta_3 \in \{\phi, ri\}$.

- $\beta_1 = \phi$: il n'y a pas de contraintes de précédences entre les tâches, et elles peuvent s'exécuter dans n'import quel ordre.
- $\beta_1 = TREE$: les contraintes de précédences entre les tâches constituent une structure d'arbre enraciné.
- $\beta_1 = PREC$: les contraintes de précédences entre les tâches constituent une structure d'arbre de différentes formes (graphe acyclique).
- $\beta_2 = \phi$: les tâches ont des coûts d'exécution distincts et arbitraire.
- $\beta_2 = E(T_i) = 1$: toutes les tâches ont un coût d'exécution d'une unité de temps.
- $\beta_3 = \phi$: il n'y a pas de temps de début au plutôt d'exécution, et l'exécution d'une tâche peut débuter immédiatement après la fin d'exécution de la tâche précédente.

- $\beta_3 = r_i$: il y a un temps de début au plutôt d'exécution, avant cet instant l'exécution d'une tâche ne peut pas déclencher même si l'exécution de la tâche précédente a été terminée.

γ représente le critère d'optimisation, ou bien la fonction à optimiser ou à maximiser d'un problème d'ordonnancement particulier. $\gamma \in C_{max}, \Sigma C_i$.

- $\gamma = C_{max}$: l'objectif est de minimiser le coût maximal d'ordonnancement (le temps total de réalisation)
- $\gamma = \Sigma C_i$: l'objectif est de minimiser la somme d'un ensemble de paramètres à travers toutes les tâches, c'est à dire la somme des poids des temps de réalisation pour chaque tâche.

δ représente le délai de communication entre les agents. $\delta \in K, P, J, JP$.

- $\delta = K$: le coût de communication est constant (ou 0 s'il n'y a pas de coût de communication) entre toutes les paires d'agents pour tous les tâches.
- $\delta = P$: le coût de communication dépend seulement des paires d'agents choisies, mais indépendamment des tâches (la quantité des données échangées est constante pour tous les tâches, mais d'autre facteurs, comme la bande passant disponible entre deux agents peut ètre instable). $C(T_i, A_i, A_j) = f(A_i, A_j)$.
- $\delta = J$: le coût de communication dépend seulement des tâches. $C(T_i, A_i, A_j) = f(T_i)$.
- $\delta = JP$: le coût de communication ne dépend pas seulement des agents communicants les résultats des tâches, mais aussi à la tâche lui-même. $C(T_i, A_i, A_j) = f(T_i, A_i, A_j)$.

2.5 Techniques d'ordonnancement

Deux grandes techniques sont distinguées : ordonnancement statique et ordonnancement dynamique.

Dans les techniques d'ordonnancement statique (off-line scheduling), toutes les informations nécessaires pour l'ordonnancement sont connues a priori, et ne se changeront jamais pendant l'exécution de la mission. Une solution du problème est donc calculée avant que l'exécution effective commence. Ces techniques sont utiles dans le cas où les robots sont dotés des dispositifs puissants pour le calcul d'ordonnancement, ou si la solution sera exécutée plusieurs fois dans les mêmes conditions. Cependant, le système est tout à fait inflexible face aux changements rapide des environnements multi-robots.

D'autre part, l'ordonnancement dynamique (on-line scheduling) est beaucoup plus flexible que l'ordonnancement statique, il réévalue à chaque nouvelle demande la tâche

qui doit être exécutée, c'est à dire l'ordonnancement est effectué en temps réel et les tâches sont ordonnancées au fur et à mesure qu'elles arrivent.

En effet, l'ordonnancement dynamique doit :

- Effectuer l'ordonnancement dans des environnements fortement dynamiques et incertains, où l'information est incomplètes et des changements souvent imprévisibles.
- Modifier l'ordonnancement formé précédemment en temps réel (le réordonnancement), conformément aux informations récentes, ainsi minimiser la rupture d'ordonnancement, visant toujours l'utilisation la plus efficace possible des ressources et l'accomplissement la plus rapide des objectifs.
- Fournir une flexibilité pour réagir de manière robuste, efficace et opportune à n'importe quelle perturbation.

Les événements dynamiques (provoquant des perturbations) généralement considérés dans les recherches, peuvent être divisés en deux catégories principales :

- Perturbations liées aux ressources : panne des machines, maladie d'opérateur, indisponibilité des outils spécifiques et des matériaux, etc.
- Perturbations liées aux tâches : les tâches qui arrivent en-avance/en-retard, nouvelles tâches, priorités dynamiques, mise à jour des dates-limites, les tâches de précipitations, etc.

Néanmoins, lorsqu'il s'agit de faire face aux aléas (ou perturbations), les règles de réparation ne sont pas les seules possibilités. Il existe deux grandes catégories de méthodes de réordonnancement [57] :

- La génération d'ordonnancement robuste qui consiste à insérer, pendant la génération de l'ordonnancement des marges de manœuvre pour pouvoir absorber les effets des aléas qui peuvent intervenir. Il s'agit de terminer les tâches en avance par rapport à leur date prévue.
- La réparation de l'ordonnancement qui consiste à réadapter l'ordonnancement initial par une modification plus ou moins importante. Ainsi, on parle de *réordonnancement complet* (ou régénération) pour désigner le calcul d'un nouvel ordonnancement sans prendre en compte celui existant. Dans le cas contraire, on l'appel *réordonnancement partiel*, dans ce cas, on se limitant à une modification restreinte de l'ordonnancement existant. Il s'agit par exemple d'opérer un Right-Shift Rescheduling, consistant à décaler les tâches vers la droite de l'axe temporel jusqu'à l'absorption des retards induits par la perturbation.

Jusqu'à ce jour, il n'existe pas de solution connue pour résoudre le problème d'ordonnancement dynamique avec garantie de toutes les contraintes de temps dans un système répartis.

En effet, les deux types de problèmes d'ordonnancement (statique et dynamique), ont été prouvés qu'ils soient des problèmes NP-Complets.

Á cause de la forte dynamique qui caractérise les applications multi-robots, nous adopterons dans ce mémoire une technique de réordonnancement dynamique pour développer notre approche d'ordonnancement.

2.6 Ordonnancement dans un contexte multi-robots

Le problème d'ordonnancement dans le contexte multi-robots est un cas particulier du problème général d'ordonnancement auquel une série d'hypothèses restrictives doivent être rajoutée, ainsi qu'une précision du vocabulaire (terminologie).

Dans ce qui suit, nous allons fixer tout d'abord, le vocabulaire qui est généralement utilisé dans le contexte d'applications multi-robots puis, analysons la complexité de ces applications en présentant un exemple typique de ces dernières. Cette complexité a pour origines la nécessité de réordonnancer les tâches d'un ou de plusieurs robots en tenant compte des couplages qui peuvent exister entre leurs plans de tâches, ainsi que le caractère dynamique de l'environnement qui impose une haute réactivité contre ses changements rapides. L'analyse qui en est faite nous permet de dégager une approche méthodologique et algorithmique qui apporte une solution générique aux problèmes de mise en œuvre de ce type d'applications.

2.6.1 Définition du vocabulaire

Les paragraphes suivantes proposent les définitions des termes utilisés dans le reste de ce mémoire.

Tâche d'un robot

La tâche d'un robot mobile peut être définie sous la forme d'une suite d'actions allant d'actions de bas niveau (déplacement d'un point à un autre, suivi de contours, suivi d'une cible, etc) jusqu'aux actions de haut niveau (inspection d'un site, nettoyage d'une pièce, etc). Des paramètres peuvent être associés à l'exécution d'une tâche comme par exemple : les coordonnées à atteindre, le temps d'exécution, l'échéance d'exécution, les conditions de déclenchement de la tâche, etc.

Prédécesseur et successeur

Deux tâches de deux robots distincts ou d'un même robot sont respectivement appelées prédécesseur et successeur si la deuxième ne peut pas commencer son exécution qu'après la fin du traitement de la première. Ce couplage définit une contrainte de type

précédence entre tâches et peut constituer un paramètre associé à l'exécution d'une tâche.

Plan de tâches

Le plan de tâches constitue un ordonnancement logique des tâches assignées à un robot. Ces tâches sont généralement liées entre elles par des contraintes logiques ou temporelles. En général, la fin du traitement de la tâche T_i constitue une pré-condition pour le déclenchement du traitement de la tâche T_{i+1}. De même, le début du traitement de la tâche T_{i+1} constitue la post-condition pour la tâche T_i.

Ressource

On appelle ressource toute entité matérielle ou logicielle à laquelle est associé un ensemble de règles d'utilisation et de partage qui indique les types d'accès autorisés. Par exemple, un couloir à explorer peut être considéré comme une ressource commune à plusieurs robots. De même, au sein d'un même robot, une base de connaissances constitue une ressource partageable par plusieurs modules intelligents. Par ailleurs, dans un système multi-robots, un robot peut utiliser simultanément plusieurs ressources en vue de réaliser une tâche complexe. Il s'agit dans ce cas de ressources assurant des fonctions complémentaires de cette tâche. Á titre d'exemple, un robot d'inspection de minerais utilise un système de radar pour communiquer avec ses congénères, une boussole pour se repérer dans l'espace et un dispositif infrarouge pour se situer par rapport aux autres robots.

Contrainte

On appelle contrainte toute condition imposée par l'architecture fonctionnelle de l'application à un robot mobile pour l'accomplissement d'une tâche donnée. Dans le présent projet nous abordons les deux types de contraintes suivantes :

- *Contrainte temporelle :* les contraintes de précédence entre les tâches d'un même plan ou de plans différents.
- *Contrainte de ressource :* en les représentant par des contraintes de précédence.

Temps d'exécution d'une tâche

Le temps d'exécution d'une tâche est le temps estimé pour le traitement total de toutes les actions élémentaires qui constituent la tâche. Ce temps est borné par les deux variables temporelles : les instants de début et de fin d'une tâche. Il représente aussi l'intervalle temporel de la durée possible d'exécution complète de la tâche. Par exemple, si T_d et T_f sont respectivement les instants de début et de fin de traitement de la tâche T, nous pouvons les définir par deux manières différntes :

- Une contrainte d'ordre de type symbolique ($T_d < T_f$).

– Une contrainte d'ordre de type numérique ($Dure_{min} < T_f - T_d < Dure_{max}$).

Échéance et échéance de précédence
Elle détermine l'intervalle de temps total autorisé pour la réalisation d'une tâche d'un plan. Cet intervalle a pour origine temporelle l'instant de démarrage de l'application multi-robots. Les tâches soumises à ce type de contraintes sont souvent des tâches de type prédécesseur, dans ce cas, on l'appelle *échéance de précédence*.

Perturbation
On appelle perturbation tout événement se produisant dans ou concernant un robot et aboutissant à une modification du plan de tâches d'un robot. Au sens de l'automatique, elle est définie comme étant un événement lié à des changements d'état logiques ou traduisant des dérives de paramètres du système de contrôle/commande ou de l'environnement. Dans le contexte d'une application robotique, il peut s'agir d'un retard ou d'un état anormal du robot pendant l'exécution d'une tâche. Ainsi, une perturbation qui se produit pendant l'exécution d'une tâche du robot peut entraîner un retard et par conséquent peut introduire des contraintes sur les autres robots.

Tâche perturbatrice et tâche perturbée
Les effets d'une perturbation peuvent, par le jeu des contraintes de précédence entre les tâches, se propager d'un plan à un autre. Ainsi, une tâche T_1 affectée par une perturbation, peut affecter indirectement le bon déroulement d'une autre tâche T_2. La tâche T_1 est alors appelée perturbatrice puisqu'elle était la cible initiale de la perturbation, elle participe ensuite à sa propagation vers les autres tâches ou plans de tâches. La tâche T_2 est appelée tâche perturbée.

2.6.2 Caractéristiques du problème d'ordonnancement dans un contexte multi-robots

Dans ce qui suit, nous illustrons la difficulté de réordonnancer le plan de tâches d'un robot mobile dans un contexte multi-robots soumis à des contraintes temporelles et fonctionnelles. Pour ce faire, nous nous basons sur un exemple d'application typique.

2.6.3 Robots mobiles explorateurs

Cet exemple présente une application qui consiste à explorer un environnement dynamique (cf. Fig.2.1). À chaque robot est affectée une parcelle à explorer. La mise en œuvre d'une telle application nécessite une phase préparatoire hors ligne par une

station centrale, qui consiste à diviser une mission spécifique en un ensemble de plans et d 'assigner à chaque robot son plan de tâches correspondant. L'objectif est de réaliser un contrôle optimal qui respecte les contraintes liées aux couplages entre les tâches des robots.

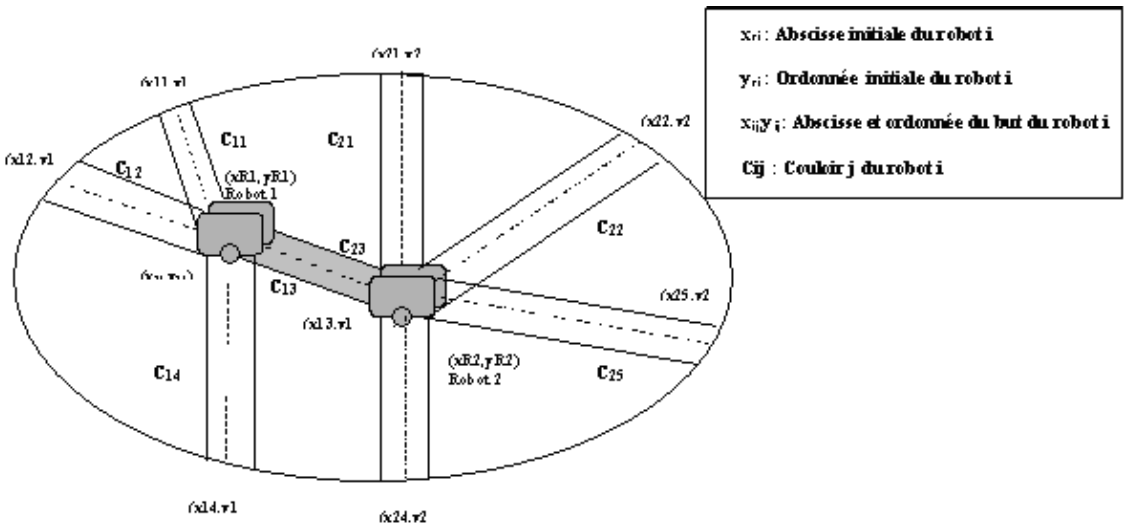


FIG. 2.1 – Application de type exploration de couloirs

L'analyse d'une telle application montre que des situations conflictuelles peuvent survenir dans certains cas. Les plans de tâches assignés aux robots peuvent être établis selon un critère donné et à partir d'une analyse globale de l'application. Il peut s'agir par exemple d'optimiser un critère de type temporel (réalisation la plus rapide de la mission) sous des contraintes globales (temporelles, fonctionnelles, spatiales). En effet, l'exploration simultanée d'un même couloir par deux robots implique un problème de partage d'une ressource commune dont l'accès est exclusif (limité à un seul robot). Ce problème se traduit dans le plan de tâches de chaque robot, par l'introduction d'une contrainte locale de type précédence d'une tâche par rapport à celle d'un autre robot utilisant la même ressource. Ainsi, la Figure (Fig.2.1) montre que le troisième couloir du robot_1 est identique au troisième couloir du robot_2, ce qui constitue une ressource commune à partager. Par conséquent, chaque robot doit avoir un plan qui tient compte du plan de l'autre robot. Comme l'exploration du couloir 3 est impérative pour les deux robots, la solution consiste à trouver la meilleure organisation temporelle des plans de tâches des deux robots. La Figure (Fig.2.2) illustre les plans initiaux assignés aux robots et la Figure (Fig.2.3). les plans optimaux assignés aux robots.

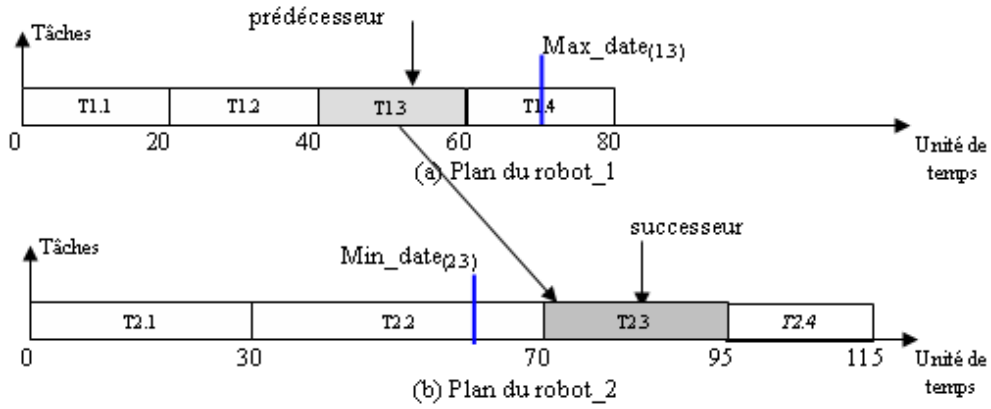


FIG. 2.2 – Plans de tâches initiaux des robots

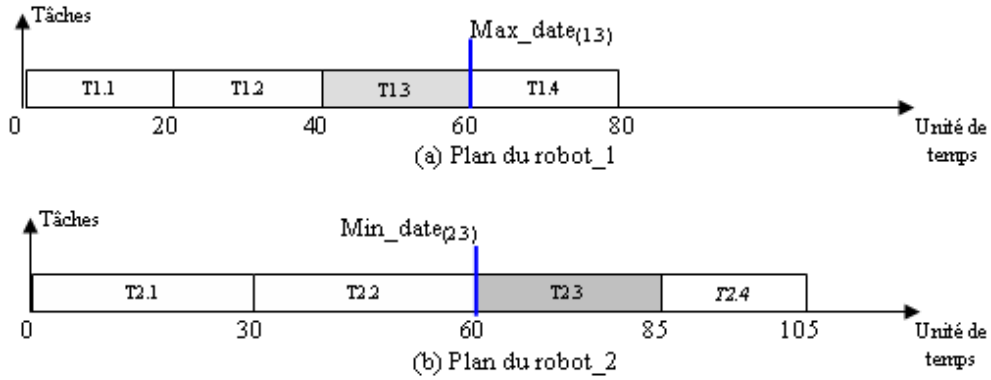


FIG. 2.3 – Plans de tâches optimaux

La deuxième phase consiste en l'exécution en ligne des plans. Pendant cette phase, un robot peut se trouver perturbé lors de l'exécution d'une tâche. Compte tenu du couplage existant entre les tâches des robots, cette perturbation peut se propager aux autres. Une réorganisation dans le temps des tâches restantes du robot perturbé s'avère donc nécessaire. Cette réorganisation vise à sortir le robot d'un état d'exception et à le remettre dans un état normal avec pour objectif de satisfaire son propre but et le but global de l'organisation. Par ailleurs, comme il s'agit à la fois d'un système multi-robots soumis à de fortes contraintes de réactivité et d'un ensemble de robots liés indirectement les uns aux autres par leurs plans de tâches, la réorganisation des tâches d'un robot est un problème réputé être NP-complet.

Parmi les caractéristiques usuels des environnements dynamiques,, la possibilité d'introduire dynamiquement des nouvelles tâches au système, ces tâches se présentent au

système pendant l'exécution d'une mission multi-robots. Cela signifie que la complexité du problème va augmenter considérablement, dans ce cas, il faut aussi ajouter la prise en compte de l'optimisation de certains nouveaux critères, comme l'équilibrage de charge, etc.

Les Figures (Fig.1.4.a) et (Fig.1.4.b) représentent respectivement le plan initial d'un robot_1 dans lequel apparaît une perturbation sous la forme d'un retard et son nouveau plan, obtenu après réorganisation dans le temps de ses tâches.

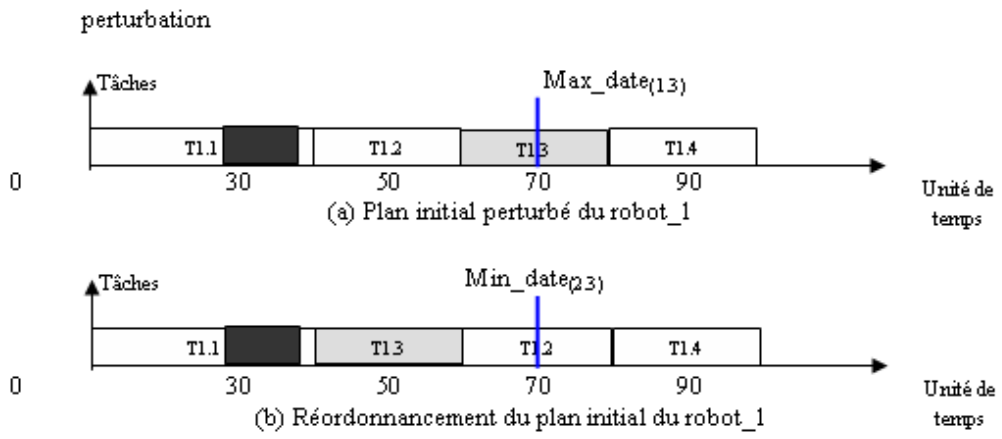


FIG. 2.4 – Figure II.3. Plans de tâches optimaux

En plus des problèmes liés au contrôle d'un seul robot, d'autres problèmes peuvent apparaître dans tels systèmes, ils ont pour conséquence l'accroissement de la complexité de la conduite de l'organisation multi-robots. En effet, dans l'exemple décrit précédemment, les plans de tâches assignés aux robots sont élaborés de manière à ce que chaque robot atteigne son propre but tout en gérant ses interactions avec autrui. Compte tenu du paramétrage particulier introduit dans la spécification du plan de tâches (contrainte de précédence), l'évolution de chaque robot doit garantir à la fois les contraintes associées à ce paramétrage et la cohérence du plan global de l'organisation. Le paramétrage consiste ici à définir en plus de la nature de la tâche (déplacement d'un point vers un autre, inspection d'une zone, etc), la contrainte qui lui est associée. À titre d'exemple, une tâche peut être soit un prédécesseur, soit un successeur, soit à la fois successeur et prédécesseur, soit enfin à la fois non-prédécesseur et non-successeur.

2.7 Approches d'ordonnancement

Les problèmes d'ordonnancements sont des problèmes complexes du point de vue mathématique, parce que le nombre de solutions possibles augmente exponentiellement

avec le nombre de tâches. Par exemple, dans un système multi-robots où 3 tâches doivent être réalisées par 2 robots dans un ordre quelconque, il existe 36 permutations possibles ($(3!)^2$). Mais pour 5 tâches et 5 robots, il y a presque 25 milliards de permutations possibles [34]. Du point de vue de la théorie de la complexité, on parle de problèmes NP-complets pour désigner les problèmes les plus difficiles pour lesquels il n'existe pas d'algorithme de résolution pouvant résoudre le problème en un temps polynomial. Il n'existe donc pas de méthode unique de résolution de tous les problèmes d'ordonnancement, mais plutôt des méthodes génériques qui permettent de résoudre des classes de problèmes spécifiques

Pour les systèmes d'ordonnancement classiques, on peut trouver deux paradigmes différents. D'une part, il y a des systèmes qui essayent de calculer une solution optimale, mais assumer un environnement stable, on parle de méthodes exactes. D'autre part, il y a des systèmes qui assument un environnement dynamique et complexe et essaient alors de trouver une solution réalisable (acceptable) qu'optimale, ce sont les méthodes approchés, dans lesquelles l'optimisation est seulement faite implicitement.

Récemment, Les systèmes d'ordonnancement sont souvent basés sur un nouveau paradigme, celui des systèmes multi-agents issus de l'IAD. Des composants logiciels actifs (appelés agents) perçoivent l'environnement d'un point de vue individuel et/ou par l'intermédiaire de communication pour essayer de trouver une solution. Il y a des systèmes qui emploient la communication indirecte (en utilisant par exemple un tableaux noirs pour placer l'information), dans d'autres systèmes, il y a une négociation directe entre les entités impliquées.

2.7.1 Les méthodes exactes

Dans ce type d'approches, les algorithmes suivent un (ou plusieurs) critère d'optimisation tout en essayant de garantir l'optimalité des solutions fournies. Elles se basent sur des calculs mathématiques complexes et coûteux rendant difficiles leur mise en œuvre et leur utilisation en dehors des cas très spécifiques et nécessitant des ressources calculatoires importantes. Plusieurs algorithmes ont été proposés dans ce contexte. On peut citer par exemple : Branch and Bound, Grid Search, Programmation dynamique, Programmation linéaire, backtrack, Divide-and-Conquer, etc.

La méthode "Branch and Bound" est un exemple typique des méthodes exactes qui illustre bien leur principe de fonctionnement. La méthode de branch and bound (procédure par évaluation et séparation progressive) consiste à énumérer les solutions d'une manière intelligente en ce sens que, en utilisant certaines propriétés du problème en question, cette technique arrive à éliminer des solutions partielles qui ne mènent jamais à la solution optimale. De ce fait, on arrive souvent à obtenir la solution recherchée en

un temps un peu raisonnable. Bien entendu, dans le pire cas, on retombe toujours sur l'élimination explicite de toutes les solutions du problème. Pour ce faire, cette méthode se dote d'une fonction qui permet de mettre une borne sur certaines solutions pour, soit les exclure soit les maintenir comme des solutions potentielles. La performance d'une méthode de branch and bound dépend, entre autres, de la qualité de cette fonction (de sa capacité d'exclure des solutions partielles tôt).

Ces techniques de résolution nécessitent des coûts calculatoires importants à cause de la taille de l'arbre des solutions ainsi générée. Dans une mission multi-robots, où l'environnement est instable, des aléas peuvent interrompre l'ordonnancement en cour d'exécution, il est donc peu adéquat de calculer une solution optimale extrêmement coûteuse en temps qui sera invalidée rapidement. Donc, ces méthodes sont incapables de fournir des solutions optimales dans un temps polynomial si on prend en compte toutes les contraintes qu'on peut rencontrer dans les applications robotiques qui sont des applications temps réel et distribuées.

2.7.2 Méthodes approchées

Les méthodes approximatives incluent les méthodes dites heuristiques et celles dites méta-heuristiques, qui permettent d'atteindre une solution sous optimale avec des temps de calcul réduits, elles sont donc plus avantageuses pour les problèmes de large envergure.

2.7.2.1 Les heuristiques

Elles se basent sur des règles simplifiées pour optimiser un ou plusieurs critères. Le principe général de ces méthodes est d'intégrer des stratégies de décision pour construire une solution proche de l'optimum. Les règles de priorité sont des méthodes d'ordonnancement populaires qui ont fait l'objet d'un grand nombre de travaux parce qu'elles sont relativement simples à mettre en œuvre. Les méthodes les plus connues sont :

- **FIFO (First In First Out)) :** la première tâche qui vient, sera la première tâche servie
- **SPT (Shortest Processing Time) :** La tâche ayant le temps d 'exécution le plus court est ordonnancée en premier lieu.
- **LPT (Longest Processing Time) :** La tâche ayant le temps d'exécution le plus important est traitée en premier lieu.
- **EDF (Earliest Deadline First) :** La tâche ayant la date d'échéance la plus petite est la plus prioritaire.
- **SRPT (Shortest Remaining Processing Time) :** La tâche ayant la plus courte durée d'exécution restant est la plus prioritaire. Elle est très utilisée dans

le cas des problèmes d'ordonnancement préemptifs.

- **ST (Slack Time) :** A chaque point de décision, la tâche ayant la plus petite marge temporelle est prioritaire. S'il y a une faute de disponibilité des ressources, cette marge peut devenir négative.

2.7.2.2 Les méta-heuristiques

Les méta-heuristiques sont des heuristiques adaptables à un grand nombre de problèmes différents sans changements majeurs dans l'algorithme. La plupart des heuristiques et des méta-heuristiques utilisent des processus aléatoires comme moyens de récolter de l'information et de faire face à des problèmes comme l'explosion combinatoire. Elles tirent en particulier leur intérêt de leur capacité à éviter les optima locaux, soit en acceptant une dégradation de la fonction objectif au cours de leur progression, soit en utilisant une population de points comme méthode de recherche. Les méta-heuristiques sont composées de deux grandes étapes : l'exploration, qui permet de visiter des régions différentes dans l'espace des solutions, et l'exploitation, qui permet de savoir où se trouvent les meilleures solutions.

Nous trouvons, dans cette catégorie, les algorithmes génétiques, les réseaux de neurones, la méthode tabou, le recuit simulé, les colonies de fourmis, la logique floue, etc., qui forment une sorte de lien entre plusieurs disciplines différentes et l'ordonnancement.

- **Le recuit simulé :** l'algorithme choisit aléatoirement une solution initiale S. Un voisin Sc de cette solution est généré. On calcule l'écart des fonctions coût Delta = f(Sc)-f(S). Si on obtient une réduction de la fonction coût, la solution courante est remplacée par le voisin. Sinon, si on a une augmentation de la fonction coût, le voisin remplace la solution courante avec une probabilité d'acceptation donnée par : EXP(-Delta/T) où T est un paramètre, appelé température, qui contrôle le processus de recuit.
- **La méthode Tabou :** À partir d'une solution initiale réalisable, le processus consiste, à chaque itération, à choisir la meilleure solution dans le voisinage de la solution courante. Afin d'éviter que l'algorithme soit piégé dans un minimum local ou un comportement cyclique, la recherche tabou utilise une structure de mémorisation temporaire dans laquelle elle sauvegarde les dernières solutions visitées : la liste tabou. Une solution reste interdite pendant un nombre d'itérations égal à la taille de cette liste. Par la suite, le meilleur voisin non tabou sera choisi pour la prochaine itération.
- **Les algorithmes génétiques :** l'exploration est réalisée par les opérateurs de mutation qui assure la diversification du voisinage ; l'exploitation est assurée par

les opérateurs de croisement et recherche les meilleurs enfants possibles (intensification).

– **Les colonies de fourmis :** c'est une nouvelle méthode de résolution des problèmes d'ordonnancement. Les colonies de fourmis sont basées sur le comportement réel de communication chez les fourmis qui consiste en "la trace" et "l'attrait".

L'avantage des méthodes approchées consiste en la réduction de temps du calcul. Elles permettent d'obtenir un bon compromis entre la qualité des solutions et le temps de calcul.

Cependant, elles présentent quelques difficultés quand elles sont appliquées dans les systèmes multi-robots. On peut noter par exemple, la sous-utilisation de la coopération. Dans la plupart de ces approches, le problème d'ordonnancement est traité en tant que problème d'allocation de ressources à un ensemble d'entités. Ces processus ont été implémentés dans des systèmes où la notion d'interaction entre entités est peu utilisée. Les solutions proposées sont de type centralisées qui implique tout le système alors que dans beaucoup de cas, la résolution n'implique qu'une partie du système, néanmoins, ces processus se rapprochent d'avantage d'une négociation entre entités

2.7.3 Méthodes à base d'agents

Une alternative qui a été proposée dans le domaine d'informatique est celle des systèmes multi-agents (SMA). Un tel système se compose des unités indépendantes et intelligentes de contrôle liées aux entités physiques ou fonctionnelles (robots, processus, tâches, ressource, etc). Ils représentent aussi une solution prometteuse pour la modélisation des systèmes multi-robots complexes, tel qu'ils offrent plus de flexibilité, de fiabilité, d'adaptabilité et de reconfigurabilité. Les agents agissent de façon autonome en poursuivant leur propre intérêt et s'interagissent, par exemple en utilisant des mécanismes d'échange d'information et de négociation.

2.7.3.1 Définition d'un système multi-agents

Bien qu'il existe un consensus général concernant les limitations des systèmes classiques et le besoin d'évolution vers des systèmes à base d'agents, la question de ce qui caractérise un système multi-agents n'a pas encore été tranchée. La définition d'un agent la plus employée est celle de Ferber [29] :

> *"Un agent est une entité physique ou logique capable d'agir sur elle-même et sur son environnement, qui dispose d'une représentation partielle de cet environnement et qui, dans un univers multi-agents, peut communiquer avec*

d'autres agents et dont le comportement est la conséquence de ses observations, de sa connaissance et des interactions avec les autres agents"

D'autres définitions incluent d'autres caractéristiques, comme par exemple la capacité d'apprentissage ou la continuité de l'existence de l'agent (par opposition à des objets ou composants invoqués seulement sur besoin).

Dès le début des années 1970 et jusqu'à aujourd'hui, il existe une école de pensée des systèmes multi-agents qui met en avant l'aspect délibératif, communicatif ou bien "social" de ces systèmes. Par contre, depuis le début des années 1990, on a vu un autre "courant", de développement plus pragmatique, ayant effectué un glissement du paradigme du "raisonnement" vers celui de l'action. Cela a eu pour conséquence l'émergence ces dernières années d'un très grand nombre d'applications multi-agents.

Tout en admettant qu'il n'est pas possible d'établir une classification qui convient à tous les systèmes multi-agents existants, Nwana [43] propose de distinguer entre sept différents types d'agents :

- Agents collaborants - Ils sont autonomes et coopèrent, afin d'atteindre un but. Souvent, ils doivent négocier pour trouver des accords mutuellement acceptables.
- Agents interfaces - C'est le cas de l'assistant personnel, capable d'apprendre les objectifs d'un utilisateur humain et d'agir de façon autonome pour l'aider ou pour exécuter des tâches à sa place.
- Agents mobiles - Il s'agit de processus logiciels capables de se déplacer à travers des réseaux informatiques, afin d'obtenir des informations localement disponibles ou d'exécuter des tâches distantes.
- Agents d'information/Internet - Ils gèrent, collectionnent et résument des informations dont la taille est trop grande pour être traitées manuellement, comme par exemple les agents des moteurs de recherche sur Internet.
- Agents réactifs - Ce type d'agents ne possède pas de modèle interne de son environnement, mais répond selon le principe "stimulus - réponse" aux événements détectés. L'interaction entre agents relativement simples peut mener à l'émergence d'un comportement synergétique de groupe..
- Agents hybrides - Ils possèdent plusieurs caractéristiques parmi celles énumérées ci-dessus.
- Agents Intelligents - Cette dernière classe d'agents est définie pour répondre au mieux à un problème donné ; chaque application pouvant avoir des besoins différents. BT Labs a par exemple identifié comme exigence minimale pour un agent intelligent qu'il doit être autonome, capable d'apprendre et de collaborer.

On peut ajouter la définition suivante d'un agent robotique :

- Agent robotique - On peut considérer l'architecture logicielle de contrôle d'un

robot comme un agent. Cette architecture peut d'ailleurs être testée et entraînée en simulation avant d'être mise en œuvre dans un vrai robot. Cependant la réalité physique d'un robot apporte des problèmes spécifiques (imprécision de la perception et de l'action, aspects temps réel, évolution du monde, etc) qui rendent particulièrement difficile, mais aussi particulièrement riche, la conception de tels agents robotiques. On peut aussi envisager des coopérations entre robots, par exemple dans le cadre de la compétition de robots footballeurs (RoboCup), présentée par ses organisateurs comme un nouveau problème étalon de l'intelligence artificielle.

Les approches multi-agents semblent être bien convenues aux problèmes d'ordonnancement complexes, particulièrement ceux qui demandent beaucoup d'interaction entre les composants du système et des réactions en temps réel, pour lesquelles les approches statiques ne peuvent pas fournir des résultats efficaces. Une approche multi-agents permet de résoudre les sous-problèmes localement par un agent, la solution globale est obtenue par les interactions entre les différents agents qui maintient les ordonnancements locaux. Les capacités inhérentes à ces approches incluent l'émergence de comportements, la réduction de complexité et des coûts, l'auto-configuration, la flexibilité, etc.

Les SMAs se composent des agents autonomes qui collaborent et coopèrent dynamiquement pour satisfaire les deux types objectifs locaux et globaux. Ces agents, avec l'émergence de comportements non préprogrammés, peuvent produire un ordonnancement flexible et dynamique plutôt qu'un simple ordonnancement optimal moins significatif dans des environnements dynamiques.

En résumé, les motivations derrière l'utilisation d'une approche SMA pour l'ordonnancement dynamique distribué sont les suivantes :

- Autonomie locale : un agent a la responsabilité d'effectuer l'ordonnancement local pour un ou plusieurs composants (fonctionnels ou physiques : robots, tâches, etc) pour l'accomplissement d'une mission multi-robots. Les agents sont capables d'observer leur environnement, de communiquer et coopérer avec d'autres agents afin de s'assurer que les ordonnancements locaux émergent un ordonnancement globale souhaitable. L'autonomie permet aux agents de répondre aux variations locales pour augmenter la flexibilité du système.
- Concurrence : la prise de décision basée sur la négociation pour l'ordonnancement ou le réordonnancement en ligne au lieu d'établir un ordonnancement totale pré-planifié. L'ordonnancement n'est pas planifié globalement (aucun ordonnancement séparé), mais émergé par la coopération et les décisions locales dynamiques et concourantes des agents. Le réordonnancement est exécuté localement en renégociant les tâches en échec. L'émergence de comportements peut s'adapter aux

circonstances changeantes beaucoup plus facilement que dans les systèmes centralisés.
- Robustesse : détection et rétablissement rapide des échecs.
- Structures ouvertes et dynamiques d'ordonnancement : il est possible d'intégrer dynamiquement des nouveaux agents (logiciel, matériel, ressources, tâches, etc), enlever des agents existants, ou encore améliorer les agents indépendamment, par exemple si des nouvelles fonctions sont nécessaires, sans perturber les liens précédemment établis et/ou réinitialiser l'environnement de travail.

2.8 Approches à base d'agent pour l'ordonnancement distribués des systèmes multi-robots

Á l'heure actuelle, il existe plusieurs systèmes d'ordonnancement basés sur le paradigme multi-agents dans plusieurs domaines : entreprises (production, logistique), réseaux informatiques (routage), réseaux de transport, robotique distribuées, etc. Ce développement est dû essentiellement à la nature distribuée des applications développées et du volume d'informations traitées. De ce fait, on peut dire que les systèmes distribués à base d'agents sont vus comme des outils pour l'aide à la décision.

Dans cette recherche, nous nous focaliserons sur l'ordonnancement dans les systèmes multi-robots.

Dans les MRS, le problème d'ordonnancement est traité en tant que problème de coordination entre les agents constituant le SMA qui contrôle l'équipe de robots. Cette coordination peut être réalisée en utilisant une méthode d'ordonnancement/réordonnancement par coopération d'agents.

En général, les stratégies de coordination multi-robots adoptent : soit une approche centralisée, où un seul agent planifie pour le groupe, soit une approche distribuée, où chaque agent est responsable de sa propre planification.

Les approches centralisées souffrent de quelques problèmes, elles ne sont pas valables pour les grands groupes de robots (problème de scalability, en anglais), répondre lentement aux changements de l'environnement, besoins de communication lourdes, système fragile avec un seul point d'échec. L'avantage principal de ces approches est qu'elles puissent produire des plans globalement optimaux et cohérents.

Tandis que la plupart des approches distribuées puissent écarter les problèmes inhérents aux approches centralisées, leur inconvénient est qu'elles puissent produire des plans réalisable plutôt qu'optimaux.

Récemment plusieurs plates-formes ont été proposées pour le contrôle des MRS, la majorité abordent la coopération et la coordination des robots, notamment par la mise en place d'un mécanisme d'allocation dynamique (ordonnancement distribué) de tâche, qui constitue généralement le cœur de ces plates-formes.

Nous présentons dans les paragraphes suivantes certaines des approches d'ordonnancement distribuées à base d'agent pour contrôler des systèmes multi-robots, qui ont relation avec le système proposé dans ce mémoire. Nous pouvons les classer en deux classes fondamentales : (ré)allocation coopérative et réordonnancement coopératif

2.8.1 Allocation et/ou réallocation de tâches

L'allocation de tâche est un concept fonctionnel principal exigé par n'importe quelle MRS. L'allocation dynamique de tâche est une classe d'allocation de tâche dans laquelle l'allocation des robots aux sous-tâches est un processus dynamique et doit être ajusté continuellement pour répondre aux changements d'environnement et/ou du groupe de robots. Le problème d'allocation distribuée de tâche dans les MRS est encore compliqué par le fait que l'allocation de tâche doit se produire dans un processus distribué dans lequel il n'y a aucun coordinateur central pour assigner les tâches. Ce qui augmente la complexité du problème parce que, à cause de la perception locale de chaque robot, aucun robot n'a une vue complète (globale) de l'état du système (informations sur le système). Ce manque d'informations ainsi que par fois de bruits de formation (informations fournies par les capteurs), chaque robot doit prendre des décisions locales de contrôle pour les actions à exécuter et quand, sans connaissance complète de ce que d'autres robots ont fait dans le passé, être faire maintenant ou feront à l'avenir.

La littérature est riche d'approches proposées comme solutions au problème de MRTA (Multi-Robot Task Allocation) [59], [18], [48], [31], [45]. On peut distinguer trois grandes classes d'approches d'allocation dynamique de tâche : à base d'inhibition mutuelle (ex : BLE), à base de motivation (ex : ALLIANCE) et à base d'enchère (ex : MURDOCH),

La majorité s'appuient sur la méthode d'enchère (ou appel d'offres, Auction-based en anglais) inspiré du marché public pour trouver le robot le plus adapté à l'exécution d'une tâche. Elles sont des enchères à premier prix (un seul tour) ou à prix négociable (plusieurs tours). Elles utilisent des variants du protocole de réseau contractuel(CNP : Contract-Net Protocol) pour la négociation entre les robots participant dans une enchère. Cette méthode a montrée son efficacité pour coordonner les organisation de robots [19]

Nous allons nous intéresser à cette méthode, qui fait partie de l'approche proposée dans ce manuscrit.

2.8.1.1 Principe général de la méthode d'enchère

L'idée générale de ce mécanisme est :

- L'administrateur (auctioneer, en anglais) envoie un appel d'offre contient une description de la tâche qu'il voudrait voir effectuée à tous ceux qu'il estime pouvoir répondre ou à tous les agents du système.
- Á partir de cette description, chaque offrant (bidder, en anglais) élabore une proposition ou un refus sous la forme d'un message, qu'il envoie ensuite vers l'administrateur.
- L'administrateur reçoit et évalue les propositions, puis attribue le marché au meilleur offrant.
- Enfin, l'offrant qui à reçu le marché et qui devient ainsi le contractant (contractor, en anglais), envoie un message à l'administrateur lui indiquant qu'il est toujours d'accord pour accomplir la tâche requise et qu'il s'engage de ce fait à réaliser ou bien qu'il ne peut s'acquitter du contrat, ce qui relance l'évaluation des offres et l'attribution du marché à un autre agent.

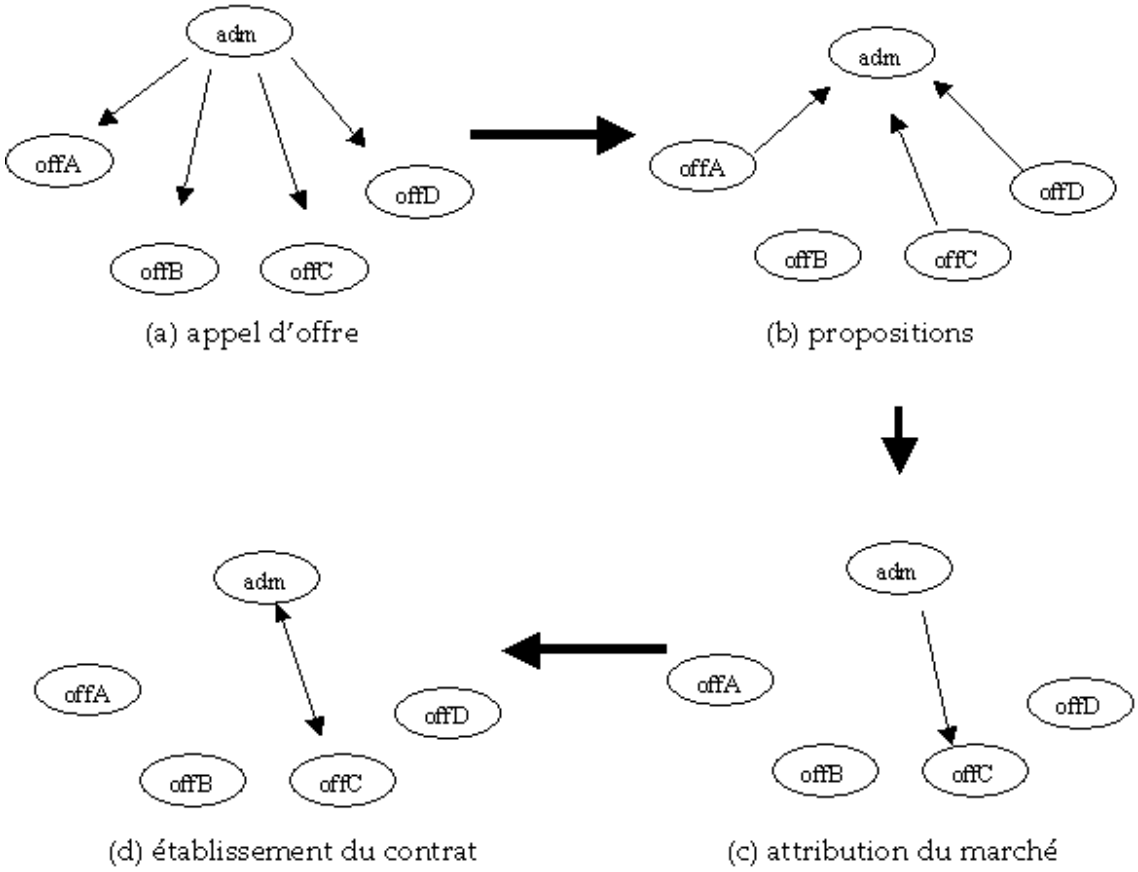


Fig. 2.5 – Principe du protocole Contract-Net

Si le système ne permet qu'une seule enchère à la fois, on dit que l'allocation est séquentielle et le système ne profite pas de la capacité des robots de faire plusieurs appel d'offres simultanément, l'avantage majeur est la cohérence du système. Sinon on dit qu'elle est parallèle, l'inconvénient majeure est que les robots offreurs puissent tomber dans le problème de la surcharge et/ou du chômage technique [29].

S'il y a un seul administrateur (fixe) dans le système on dit que l'allocation est centralisée. Sinon on dit qu'elle est décentralisée.

Dans les mécanismes d'allocation de tâches, on remarque que la notion d'utilité (revenu/coût) joue un rôle très important

2.8.1.2 Notion de l'utilité

L'utilité est un amalgame, parfois implicite, c'est un concept fondamental dans les sciences économiques, théorie des jeux, et recherche opérationnelle. L'idée est que chaque robot peut estimer individuellement une valeur interne (revenu/coût) pour réaliser une tâche ; appelée aussi aptitude, évaluation ou encore capacité. Puisque la formulation exacte se change d'un système à un autre, une définition d'utilité pour les MRS suppose que chaque robot soit capable d'estimer leur aptitude pour chaque tâche dont il est capable d'exécuter. Cette évaluation doit inclure deux facteurs fondamentaux, à savoir, la tâche et le robot :

- Revenu (ou qualité) prévue d'exécution de la tâche : on le définit par la méthode et les équipements à employer (par exemple, l'exactitude d'une carte (map) est liée à la qualité du télémètre de laser utilisé).
- Coût prévu des ressources : on l'obtient par les conditions spatio-temporelles de la tâche (par exemple, l'énergie exigée par les moteurs et le télémètre de laser afin de construire une carte).

Supposons un robot R et une tâche T, si R est capable d'exécuter T, alors nous pouvons définir, sur une certaine échelle normalisée, Q_{RT} et C_{RT} comme qualité et coût, respectivement, ont compté résulter de l'exécution de T par R. Nous pouvons maintenant définir une mesure d'utilité combinée et non négative :

U = $Q_{RT} - C_{RT}$ si R est capable d'exécuter T et $Q_{RT} > C_{RT}$.
0 sinon

L'évaluation d'utilité des robots sera inexacte à cause de quelques raisons, comme le bruit des capteurs, l'incertitude générale, et le changement environnemental. Ces caractères inévitables dans le domaine de la robotique limitera nécessairement l'efficacité avec laquelle la coordination peut être réalisée.

Les recherches sur les MRS supposent que le contrôle du bas niveau de robot est fiable, robuste, et précis que possible et ainsi que nous sommes incapables de l'améliorer.

2.8.1.3 Techniques d'allocation

Une allocation efficace de tâches dans un MRS doit réitérer l'allocation (réallocation ou encore réordonnancement distribué) de tâches, afin de traiter des changements des tâches, des robots et de l'environnement. Les architectures existantes réalisent cette itération de différentes manières :

- Quelques approches permettent l'allocation et la réallocation de toutes les tâches à chaque itération (ex ; BLE et ALLIANCE), d'autres ne réallouent jamais à nouveau les tâches, seulement en cas d'échec d'un robot (ex : MURDOCH).
- D'autres approches effectuent périodiquement une réallocation des tâches (ex : BLE), tandis que d'autres effectuent une réallocation des tâches pendant qu'elles sont offertes pour réallocation (apériodique) (ex : ALLAINCE).

Pour l'approche proposée dans ce mémoire, nous adopterons l'allocation à la demande (apériodique) et seulement en cas d'échec du robot à accomplir une de ses tâches.

Afin de décrire les problèmes de MRTA, trois axes ont été proposés [23] :

- Single-Task robots (ST) vs. Multi-Task robots (MT) : ST signifie que chaque robot est capable d'exécuter une seule tâche à la fois. Par contre MT signifie que chaque robot peut exécuter plusieurs tâches simultanément
- Single-Robot tasks (SR) vs. Multi-robots tasks (MR) : SR signifie que la réalisation d'une tâche exige exactement un robot, tandis que MR veut dire qu'une tâche peut exiger plusieurs robots.
- Instantaneous Assignment (IA) vs. Time-extended Assignment (TA) : IA signifie que les informations disponibles concernant les robots, les tâches et l'environnement permettent seulement une attribution instantanée des tâches aux robots, sans planification pour des futures attributions. TA signifie que plus d'informations sont disponibles, comme l'ensemble de toutes les tâches qui devront être assignées ou d'un modèle de prédiction pour l'arriver des tâches.

2.8.1.4 Approches relatives à la (ré)allocation coopérative de tâches

MURDOCH [23] : est un système de contrôle Multi-robots hétérogène à base de comportement (behavior-based). Il se base sur un mécanisme dynamique d'allocation de tâche à base d'enchère, pour cela, il utilise une variante du protocole de réseau contractuel (CNP). Les tâches sont assignées à travers un appel d'offre à premier-prix

séquentiellement pendant qu'elles sont présentées au système. Il n'y a pas de réallocation de tâche, seulement s'il y a un échec dans l'accomplissement d'une tâche par les robots.

L'idée est de construire des sous-ensembles anonymes orienté-tâche en se basant sur les compétences des robots (les ressources à ses dispositions (camera, mobilité, sonars, etc). MURDOCH utilise une technique de communication dite publier/abonner (publish/subscribe), chaque robot peut s'abonner dans un ou plusieurs sous-ensemble selon leurs compétences. Par exemple, mobile, sonar, camera, etc. elle est basée aussi sur l'information d'état actuel (niveau d'énergie, etc).

MURDOCH utilise l'adressage à base du contenu, le message d'une enchère pour une tâche à attribuer est orienté-sujet (c'est à dire ce message s'adresse seulement aux sous-ensembles dont leurs sujets correspondent aux sujets du message, au lieu de le diffuser dans tout le système. Par exemple, une tâche exigeant le sonar, le laser, et la vision publie en utilisant le tuple (sonar, laser, caméra), ce mécanisme est similaire au multicast dans le domaine des réseaux informatique), et est accompagné d'un métrique comme les ressources requises pour l'exécution de la tâche(coût). Après ça, chaque robot enregistré et compétent, retire son abonnement, et estime sa propre utilité (d'une manière spécifique) basée sur le métrique reçu, et communique son score à l'administrateur. Après réception des scores des autres robots, l'administrateur effectue des comparaisons entre les offres(utilités) reçues, et informe le robot gagnant, ce dernier entame la réalisation de la tâche sous un contrat avec l'administrateur. Alors que les autres réabonnent dans les sous-ensembles adéquats.

L'algorithme best-fit est employé pour choisir le meilleur parmi les robots qui sont inscrits à un sujet particulier. La décomposition de la tâche est assurée par un utilisateur humain ou un composant externe du système.

Cette approche est valable même s'il y a un nombre assez grand de robots dans l'équipe (dans le sens des capacités de calcul et de communication), mais puisqu'il ne permet pas une réallocation dynamique durant l'exécution d'une mission multi-robots, cela peut diminuer les performances et l'efficacité de l'organisation de robots à cause de la dynamique des environnements multi-robots.

DEMiR-CF (Distributed and Efficient Multi-robots - Cooperation Framework) [48] : est une plate-forme générale pour la coopération multi-robots, avec communications limités. Cette plate-forme intègre également un mécanisme d'allocation distribuée de tâche, basé sur la méthode d'enchère à un tour, un mécanisme de formation de coalitions et de sélection d'action, etc.

Il y a une mission globale composée d'un ensemble de cibles à visiter. Chaque robot sélection les cibles qui lui sont appropriées en utilisant une équation qui mène à

sélectionner et constituer l'ensemble des endroits les plus proches d'un robot r_j. Puis, chaque robot planifiera sa mission pour visiter les endroits choisis. Chaque fois le robot prend la cible la plus prioritaire à visiter et annonce un appel d'offres pour celle-ci. Les offreurs qui sont capables de visiter cet endroit envoient leurs coûts comme offres pour la tâche, celui qui a offert la meilleure offre va exécuter la tâche.

Si l'exécution d'une tâche nécessite plus d'un robot, la plate-forme utilise également la méthode d'enchère pour former une coalition de robots. Ainsi, les robots appropriés et leur nombre seront bien déterminés.

RACHNA (Robot Allocation through Coalitions using Heterogeneous Non-Cooperative Agents) [58] : est une autre solution pour MRTA à base du marché, utilise la formation de coalitions pour contrôler un MRS hétérogène non coopératif. RACHNA utilise également la méthode d'enchère pour allouer les tâches aux robots hétérogènes constituant le système.

Cette architecture comporte deux types d'agent logiciels :

- Les agents de service : sont des médiateurs par lesquels les tâches sont offertes pour le service. Chaque robot contient un ensemble de services ou de rôles qu'il peut exécuter. Les rôles sont déterminés par les capteurs individuels et les capacités comportementales de chaque robot. Un agent de service est nécessaire pour chaque type de service qu'un robot peut fournir. Un agent de service peut communiquer avec n'importe quel robot qui fournit le service particulier auquel l'agent correspond, les agents de service résident sur n'importe quel robot capable de fournir le service. Ainsi, l'information globale de la tâche est acquise d'une façon décentralisée par l'intermédiaire des agents de service.
- Les agents de tâche placent les offres de la part des tâches afin d'acquérir les services nécessaires. Les agents de tâche communiquent seulement avec les agents de service pendant les négociations. Une fois que la tâche est assignée, l'agent de tâche peut communiquer directement avec les robots alloués à la tâche. Les agents de tâche peuvent résider sur un poste de travail ou un robot et communiquer avec les agents de service nécessaires. Il y a trois types de tâches, Urgente, Standard, Non préemptive. Deux types d'allocations ont été utilisés : Allocation Instantanée, Allocation préemptive.

2.8.2 Réordonnancement coopératif de tâches

Peu de travaux ont été menés sur ce type d'ordonnancement, un exemple typique de ce type d'approches a été proposé par Toukal [55]. Il a présenté une approche pour

le réordonnancement dynamique, mais sans utilisation de l'allocation de tâches dans une organisation multi-robots en vue d'absorber les effets des perturbations qui se produisent pendant la réalisation d'une mission.

Le traitement d'une perturbation passe d'abord par une résolution locale (au niveau du robot perturbé) du problème de réordonnancement en se basant sur une algorithmique spécifique, et en cas d'échec, il utilise des mécanismes de négociation entre robots qui visent à restreindre (voire à absorber) la propagation de la perturbation au reste de l'organisation. Le réordonnancement des tâches s'effectue afin de minimiser deux critères : le premier correspond au nombre d'agents de l'organisation affectés par une perturbation se produisant au niveau d'un robot membre. Le second représente la somme des retards induits au niveau de l'organisation. Les mécanismes de négociation ont pour objectif de mettre en œuvre un partage de connaissances en vue de compléter les connaissances a priori du robot perturbé et ainsi permettre d'établir un nouveau plan de tâches. Pour garantir l'émergence d'une solution coordonnée dans tous les cas, la résolution peut passer par la propagation des contraintes de l'agent perturbé à un autre. Une telle approche a pour objectif de garantir une certaine robustesse et par conséquent de faire émerger un comportement global de l'organisation tendant à satisfaire son objectif.

2.8.3 Meilleure approche

Il n'y a pas à proprement dit une meilleure approche, mais ça dépend de l'application et du contexte pour lesquels l'organisation multi-robots est conçu. Chacune des approches présentées ci-dessus a ses forces et faiblesses, et toutes jouent des rôles importants et réussies dans certains configurations (problèmes et applications). Les différentes approches sont souhaitables pour différentes situations, nature des tâches, et les capacités matérielles et logicielles des robots.

Dans les environnements caractérisés par une dynamique faible et constantes, si on peut obtenir une spécification complète et précise de la mission (buts, tâches, etc) à accomplir et un temps suffisant pour planifier et ordonnancer les plans des robots hors-ligne, une approche de réordonnancement sans allocation va permettre aux robots de réagir rapidement contre les perturbations en appliquant un réordonnancement coopératif concerne seulement les tâches constituant le plan local d'un robot, sans besoin d'allouer certaines de ses tâches qui peuvent amener à des situations complexes et par fois délicates s'il y a des contraintes de précédence et de ressources. Cependant, étant situé dans un monde bruyant et dynamique rend habituellement ceci impossible (prédiction de la dynamique de l'environnement et du future), alors les approches de

réallocation dynamique de tâches présentent une bonne solution pour un ordonnancement en-ligne (dynamique) de tâches. Dans les environnements multi-robots dans lesquels des nouvelles tâches peuvent s'apparaître durant la réalisation d'une mission et qu'elles doivent être accomplir par le système dès que possible. Dans ce cas, il faut intégrer les deux mécanismes, dans un premier temps nous construisons un ordonnancement hors-ligne en se basant sur les informations acquises, et dans un deuxième temps, il suffit de faire un appel d'offre pour chaque nouvelle tâche, choisir le robot qui offre la meilleure proposition et lui attribuer la tâche correspondante. Finalement, il faut réordonnancer les plans de tâches pour adapter la mission avec les nouveaux changement (perturbations).

La majorité des approches d'allocation de tâches traitent le problème d'ordonnancement avec prise en compte seulement des contraintes d'échéances d'exécution. Par contre, les approches de réordonnancement abordent les contraintes de précédence et de ressources mais supposent qu'il n'y a pas de nouvelles tâches. L'intégration de ces contraintes dans une même approche est rarement abordée puisque l'ajout de ces contraintes rend le problème d'ordonnancement beaucoup plus difficile à traiter.

2.9 Conclusion

Dans ce chapitre, nous avons présenté les différents types de problèmes d'ordonnancement et les principales méthodes de résolution existantes dans la littérature.

Les méthodes exactes calculent la solution globalement optimale, mais nécessitent souvent des capacités calculatoires importantes et un temps de calcul croissant exponentiellement avec la dimension du problème. Les méthodes approchés essayent de trouver une solution approximative, généralement obtenue en un temps borné. Ces méthodes ont été spécialement mis en œuvre dans les systèmes distribués soumis à des contraintes de temps. Chacune présente des caractéristiques propres selon le type de contraintes traité, selon que l'ordonnancement s'effectue localement ou globalement et selon qu'il soit bien adapté à telle ou telle famille d'applications. Donc, le type d'ordonnancement approprié dépend essentiellement de ces caractéristiques. La plupart des approches de ces deux types souffrent de quelques limitations (le manque de prise en compte de la nature intrinsèquement distribuée du problème d'ordonnancement, la sous utilisation de la coopération, etc). Ainsi dans la plupart de ces approches, le problème d'ordonnancement est perçu comme un problème d'allocation de ressources. Par ailleurs, les processus coopératifs utilisés sont fondés sur un mode de coopération unilatéral car n'engageant qu'un seul robot. En effet, ces processus ont été implémentés dans des systèmes où la notion d'interaction entre entités est peu utilisée. Les solutions proposées sont de type centralisées et affecte tout le système alors que dans plusieurs applications,

la résolution n'implique qu'une partie du système.

Dans le contexte d'une application multi-robots, où l'ordonnancement s'effectue en fonction de l'état d'avancement de l'exécution d'une mission, la recherche s'est naturellement orientée vers des systèmes d'ordonnancement basés sur le paradigme multi-agents qui sont particulièrement bien adaptés à ce mode d'ordonnancement, du fait qu'ils permettent la négociation distribuée et dynamique entre entités indépendantes.

L'intégration de méthodes de réordonnancement pour supporter des processus de résolution coopératifs, permet cependant d'envisager une plus grande souplesse dans la gestion des perturbations. Aussi, l'intégration de méthodes d'allocation permet de supporter de plus en plus une très grand variété d'applications multi-robots et d'augmenter leur flexibilité dans la gestion des perturbations

Dans notre approche, nous voulons ajouter aux agents une attitude de bienveillance qui les conduit à prendre en compte les intérêts (contraintes) des autres agents.

Au vu de ces conclusions, nous proposons dans le cadre de cette recherche une approche de résolution de problème d'ordonnancement dynamique et distribué faisant appel à la coopération entre agents, cette approche intègre quatre stratégies de résolution que nous allons bien détailler dans le chapitre 4.

Chapitre 3

Modèle d'organisation et d'architecture d'agents

3.1 Introduction

Dans les configurations où la prise de décision est distribuée, nous proposons un modèle d'ordonnancement distribué et dynamique de tâches pour la gestion de perturbations et de ressource selon une approche agents d'une organisation multi-robots, qui peut être intégrer dans la couche délibérative dotée d'outils de planification de tâches, adapté à un large éventail d'applications multi-robots.

Nous avons montré dans le Chapitre I que les robots mobiles autonomes constituent des systèmes complexes. Compte tenu de la nature distribuée et dynamique des environnement multi-robots et pour des raisons de souplesse et de modularité, nous proposons naturellement une approche basée sur l'utilisation d'un système multi-agents privilégiant à la fois la distribution de la décision, l'autonomie et la réactivité des agents ainsi que la coordination décentralisée d'actions en cas de perturbations. Pour ce faire, nous avons modélisé la résolution de ce problème par une organisation d'agent-robots autonomes de type hybride. Chaque agent a une mission spécifique au sein de l'organisation et doit posséder un niveau substantiel d'autonomie à travers des capacités d'ordonnancement autonome de tâches, de perception de son environnement qui est considéré ici dynamique, de communication/négociation avec d'autres agents membres, de navigation autonome (évitement d'obstacles imprévus), etc. Le système est composé d'un agent-central, qui représente la station centrale et qui joue le rôle du point d'entré au système (interface avec l'utilisateur, planification, allocation des nouvelles tâches, etc) et d'un ensemble d'agent-robots autonomes.

Le modèle à développer doit permettre à un ensemble de robots de coopérer a fin d'accomplir de manière robuste la réalisation d'une mission complexe décomposable en

un ensemble de tâches par un planificateur spécialisé dans la station centrale, mais, les tâches d'une mission ne sont pas nécessairement toutes connues avant son démarrage, certaines peuvent être introduites durant son exécution (ajoutées par l'utilisateur ou bien générées automatiquement par le système). On souhaite alors que chacun des robots parvient à réadapter leur plan avec ou sans échange d'informations et/ou des tâches avec les autres. Avant le démarrage d'une mission, le planificateur dans la station centrale doit décomposer une mission en tâches et les ordonner pour construire des plans de tâches selon le nombre de robots constituant le système (planification manuelle ou automatique). La planification de la mission est basée sur les informations actuellement existantes, ces informations peuvent se changer après le démarrage de la mission, dans ce cas, un réordonnancement (réadaptation) des plans initiaux est nécessaire.

Dans ce chapitre, nous exposons d'abord, la démarche générale de la méthode d'ordonnancement proposée, pour pouvoir par la suite présenter les modèles que nous avons retenus (d'organisation, d'agent, etc.) pour développer une approche d'ordonnancement coopérative. En fin, nous présentons la définition et la représentation des comportements fondamentaux assignés aux agents-robots pour qu'ils puissent réagir de manière adéquate en cas de perturbations.

3.2 Démarche générale de la méthode coopérative d'ordonnancement distribué

La méthode d'ordonnancement proposée fournit aux robots membres d'une organisation, la capacité pour contrôler l'exécution de leurs missions individuelles et/ou globales en temps réal et de gérer les événements indésirables ou perturbateurs (retard, insertion d'une nouvelle tâche, etc) arrivant durant l'exécution et qui peuvent ruiner leurs plans de tâches individuelles. Par conséquent, ils peuvent affecter la mission globale. Pour ce faire, à chaque occurrence d'un tel événements, un traitement local doit être exécuté en vue d'absorber les effets localement (basé sur les informations locales) sans solliciter les autres robots de l'organisation. Suite à la tentative de réordonnancement local et en cas d'échec, un traitement local avec négociation doit se lancer pour mettre à jour les connaissance acquises a priori, ensuite, le réordonnancement local est relancé à nouveau en prenant en compte les nouvelles connaissances reçues à partir des autres robots. Si ce deuxième réordonnancement n'arriverait pas à absorber la perturbation, on fait appel à un réordonnancement global qui vise l'allocation d'une ou de plusieurs tâches qui se trouve dans le plan de tâches du robot perturbé aux autres robots jugés plus prioritaires. Si les trois premiers réordonnancements n'aboutiraient pas à absorber tous les effets de la perturbation et pour que la méthode d'ordonnancement

proposée garantie toujours l'obtention d'une solution globalement cohérente, elle doit intégrer un mécanise de réordonnancement global par propagation de contraintes vers d'autres robots, qui seront par la suite, les nouveaux robots perturbés et devront donc appliquer la même démarche de résolution.

3.3 Modèle multi-agents

Nous avons vu dans les chapitres précédents que les systèmes multi-agents représentent un outil intéressant de modélisation des systèmes multi-robots. Dans un problème d'ordonnancement, chaque agent représente une unité de résolution de problème, associé à un robot de l'organisation et doté d'une capacité de réaction aux perturbations. Les agents doivent être aussi capables de coopérer entre eux pour atteindre les objectifs individuels (minimisation des effets des perturbations sur leurs plans locaux.) et les objectifs collectifs (minimisation des effets des perturbations sur tous les plans). Alors, leurs interactions consistent à échanger des informations sur leurs états internes (avancement dans la réalisation des plans locaux, etc) et des tâches ((ré)allocation) pour atteindre les objectifs cités précédemment.

3.3.1 Modèle d'environnement

Tout système multi-agents est fondé sur l'interaction d'agents dans un environnement. Nous n'entrerons pas dans le débat théorique de la définition de l'environnement d'un agent et nous nous baserons sur la définition suivante :

> *L'environnement dans lequel les robots s'évoluent est dynamique et constitué d'un ensemble d'obstacles. Il y a deux types d'obstacles : les obstacles statiques (connus ou inconnus : murs, objets fixe, etc) et les obstacles mobiles (les autres robots, opérateurs humains, etc)*

Nous appelons environnement dynamique un environnement dont les propriétés physiques ou structurelles sont susceptibles d'être modifiées immédiatement à tout moment. Un environnement dynamique est modifié du fait des robots qui y évoluent (actions des robots sur leur environnement) ou du fait de sa propre dynamique interne (évolution de ses composants). Dans le cas d'un environnement robotique, cette dynamique est souvent doit être traitée en temps réel. Le temps réel est un mécanisme où le déroulement des actions en cours ne découle pas d'une relation synchrone avec les robots dans l'environnement. C'est à dire que les propriétés d'un environnement évoluent sans être soumises à un cadençage imposé par le robot.

3.3.2 Modèle d'organisation

L'organisation multi-agents qui est proposée, modélise un système multi-robots comme celui décrit dans la section 5.2.1 du Chapitre 2. Cette modélisation intègre les concepts spécifique multi-agents et ordonnancement. Il y a deux types d'agents : un agent-central et plusieurs agent-robots (cf. Fig.3.1).

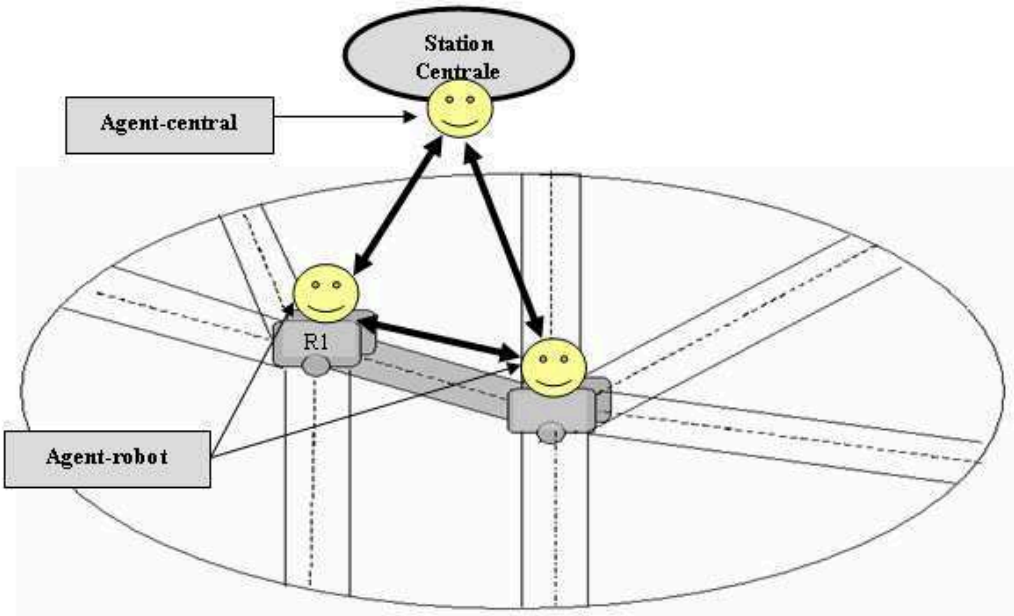


FIG. 3.1 – Modèle d'organisation d'agents

3.3.2.1 Modèle de l'agent-central

Pour garantir une certaine cohérence dans le système, nous estimons qu'il est nécessaire d'utiliser un agent-central capable d'assurer les fonctionnalités suivantes :

- **Planification :** c'est l'élaboration hors-ligne des plans de tâches de tous les robots constituant l'organisation (cf. Fig.3.2). Chaque plan est élaboré en tenant compte de la nature et des contraintes de l'application (contraintes de précédence). Ensuite, les plans seront transmis aux agent-robots pour leurs réalisation. Dans cette recherche, nous n'abordons pas le problème de planification puisque la réalisation d'un planificateur de tâches sous contraintes est un sujet de recherche à part entière, et les outils existants, comme par exemple I_XT_ET qui est un outil développé au LAAS [24], ou GRAMMPS développé à CMU (Carnegie Mellon University) [13], sont lourds à mettre en œuvre. En effet, ces planificateurs ne donnent pas non plus des solutions optimales et ne sont pas spécifiquement adaptés pour résoudre des problèmes d'ordonnancement dynamique.

- **Ordonnancement des requêtes d'autorisation :** les trois dernières étapes de la méthode d'ordonnancement proposée doit être effectuées par un seul agent à la fois. Les agents mettent en œuvre les traitements impliqués dans ces étapes à tour de rôle. Pour ce faire, il y a deux techniques généralement utilisées, soit les agents prennent la main dans un ordre prédéterminé, soit une autorité centrale leur donne la main. Dans notre approche, c'est l'agent-central qui joue le rôle de cette autorité. Si un agent déciderait d'activer un traitement qui nécessite une communication avec d'autres agents, il doit d'abord, obtenir une autorisation (permission) à partir de l'agent-central, pour ce faire, il lui envoie une requête demandant cette autorisation. L'agent-central reçoit les requêtes des agents et les ordonne selon la gravité des perturbations. Ensuite, il attribue l'autorisation aux agent selon cet ordre.
- **Ordonnancement des nouvelles tâches :** après le démarrage d'exécution d'une mission, plusieurs tâches peuvent se présenter au système et qui doivent être réalisées par les robots, dans cas, c'est l'agent-central qui est responsable de négocier leurs insertion dans les plans de tâches des robots en appliquant un protocole dérivé du protocole Contract-Net [29].

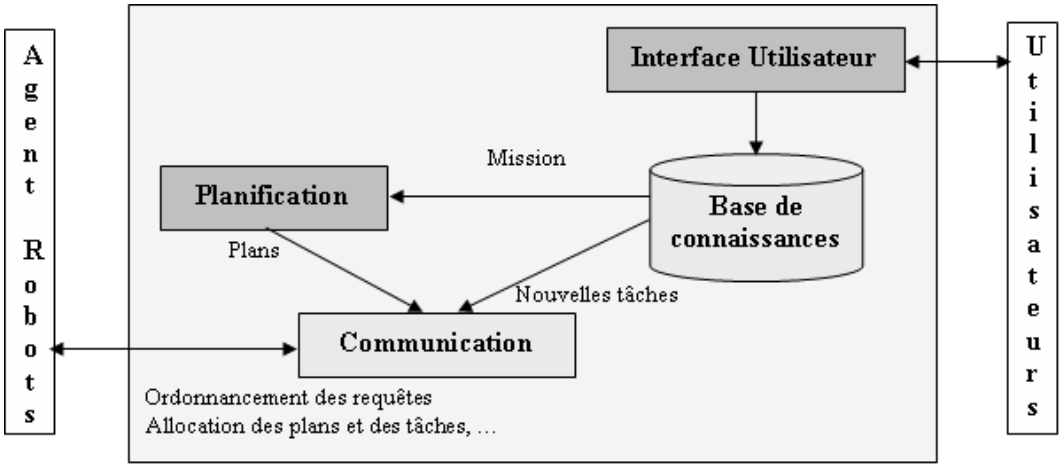


FIG. 3.2 – Architecture de l'agent-central

3.3.2.2 Modèle d'un agent-robot

Dans l'approche proposée, chaque agent-robot dispose d'un but et d'un plan lui permettant d'y atteindre. La difficulté porte sur la résolution des conflits qui peuvent intervenir pendant la réalisation des plans. La solution que nous préconisons pour traiter ce problème, consiste pour les agent-robots à partager des connaissances portant sur leurs plans et leurs objectifs individuels, afin que chacun puisse satisfaire ses propres objectifs et les objectifs du groupe. Par conséquent, chaque agent-robot dispose d'un

Ordonnanceur local lui permettant en cas de perturbation de tenter à réordonnancer son plan de tâches initial, soit localement, soit par négociation avec d'autres agents, et cela en vue de minimiser la propagation de la perturbation à ces derniers. En plus de ses interactions avec les autres agents, chaque agent doit prendre en compte, l'état de son environnement, l'état d'avancement dans son plan de tâches. La mise en œuvre de cette approche implique de doter chaque agent-robot de plusieurs niveaux de réactivité (niveau organisation, niveau local, etc.)

Il est nécessaire de doter les agent-robots par les capacités de perception, de communication, de raisonnement et d'action. Le couple perception/communication sont utilisées pour mettre à jour les informations sur l'environnement et sur l'organisation. Le système décisionnel (de raisonnement) doit être capable de gérer de manière intelligente les différentes situations possibles du robot et contrôler les activités du robot pour atteindre les objectifs souhaités.

Le degré d'intelligence d'un agent-robot est étroitement lié aux types d'environnement dans lequel il doit s'évoluer, ainsi qu'à la complexité des tâches qui lui sont assignées. Dans le contexte d'une application multi-robots où les robots ont des plans de tâches couplés, L'intelligence concerne d'une part, la stratégie individuelle de l'agent quant à la réalisation de son plan de tâches et d'autre part, la stratégie de l'agent vis-à-vis du reste de l'organisation. Dans le premier cas, l'agent doit présenter dans une situation donnée, un comportement tendant à satisfaire ses propres objectifs. Ce comportement représente dans notre contexte la capacité de l'agent à réordonnancer son plan de tâches en cas de perturbation. Pour ce faire, l'agent doit disposer de connaissances a priori sur les plans de tâches des autres agents et d'une représentation de son état suite à l'occurrence d'une perturbation. Comme l'agent a une perception limitée de l'organisation puisque la connaissance est distribuée, ceci lui impose d'avoir un comportement coopératif lui permettant de partager des connaissances et de négocier avec les autres agents en vue d'élaborer un nouveau plan de tâches. Par conséquent, il est nécessaire de doter l'agent d'un mécanisme qui permette de gérer cette connaissance partielle et distribuée afin d'avoir un raisonnement global cohérent. Dans le deuxième cas, un agent peut être sollicité pour une négociation par un agent perturbé. Une telle situation impose à l'agent sollicité, de mettre en œuvre un comportement coopératif consistant en un partage de connaissances, pour traiter la perturbation et satisfaire les objectifs de l'organisation.

3.3.3 Modèle d'interaction

Le modèle d'interaction qui est proposé est basé sur l'utilisation du concept de la coordination d'actions pour la gestion de perturbations au niveau de l'organisation.

Par exemple, dans le cas de l'exploration de couloirs, chaque agent-robot a au départ la mission d'accomplir son propre plan de tâches indépendamment des autres. Malgré cette indépendance au niveau de la propriété sur chacune des tâches d'un plan, un robot perturbé peut empêcher un autre d'atteindre son objectif.

La coordination a pour objectif d'aboutir à un réordonnancement coordonné des tâches par négociation afin de satisfaire les contraintes temporelles et fonctionnelles imposées par l'application. Ces contraintes peuvent être de type temps de réalisation à ne pas dépasser, ou de type échéance limite autorisée dans la réalisation d'une tâche ou encore, de type précédence dans la réalisation de deux ou plusieurs tâches. Pour ce faire, cette coordination s'appuie sur le concept de la coopération au travers d'un mécanisme de négociation dont l'objectif est de minimiser la propagation d'une perturbation se produisant au sein d'un agent-robot vers les autres agents. Le critère à minimiser est dans ce cas, la somme des retards engendrés par la perturbation au sein de tous les agents robots. L'analyse de la littérature montre que la résolution de ce problème reste ouverte dans sa généralité. Notre objectif est donc de donner un cadre général pour la résolution du problème d'ordonnancement distribué de tâches de robots multiples. Avant de présenter en détails dans le chapitre IV, le modèle de coopération proposé pour traiter ce problème, nous présentons dans ce qui suit les principes sur lesquels il repose :

En cas de perturbation, l'agent-robot procède d'abord à un réordonnancement local à partir des connaissances a priori dont il dispose. Ces connaissances correspondent aux contraintes de couplage qui existent entre ses propres tâches et celles des autres agent-robots. Dans notre modèle d'organisation, ces connaissances sont communiquées à chaque agent par l'agent-central avant le démarrage de l'application. Cette procédure de recherche d'un réordonnancement local a pour objectif de satisfaire toutes les contraintes de couplage imposées par l'application à l'agent perturbé sans contacter les autres. Si ce réordonnancement autonome ne satisfait pas toutes ces contraintes, l'agent perturbé lance alors, un autre réordonnancement local avec négociation. Il consiste à envoyer des requêtes aux agents qui ont des tâches successeurs associées à ses tâches prédécesseurs. L'objectifs de ces requêtes est de demander les nouvelles dates de début d'exécution des tâches successeurs, c'est à dire, de mettre à jour les connaissances de l'agent. S'il y a des modifications dans ces dates, le réordonnancement local doit être relancer. Si malgré l'exécution de ce réordonnancement global, le nouveau plan élaboré ne satisfait pas toutes les contraintes, l'agent passe à une autre négociation visant l'allocation de tâches aux autres agents. Sinon, il procède a une propagation de contraintes.

En résumé, le modèle de coopération qui est proposé ici, est basé sur trois protocoles fondamentaux d'interaction : réordonnancement local avec négociation, réordonnancement global par allocation de tâches et réordonnancement global par propagation de

contraintes :

Dans le premier, l'agent perturbé identifie l'état d'exception engendré par la perturbation (conflit, etc.). Puis il envoie aux agents successeurs une requête demandant la négociation. Cette négociation a pour objectif d'enrichir et de mettre à jour sa base de connaissances à partir des connaissances obtenues des autres agents coopérants. Chaque agent sollicité pour la négociation, transmet à l'agent perturbé une réponse contenant des informations locales. L'agent perturbé interrompe la négociation et vérifie s'il y a des modifications dans ses connaissances, s'il est le cas, il recommencera une tentative de réordonnancement local.

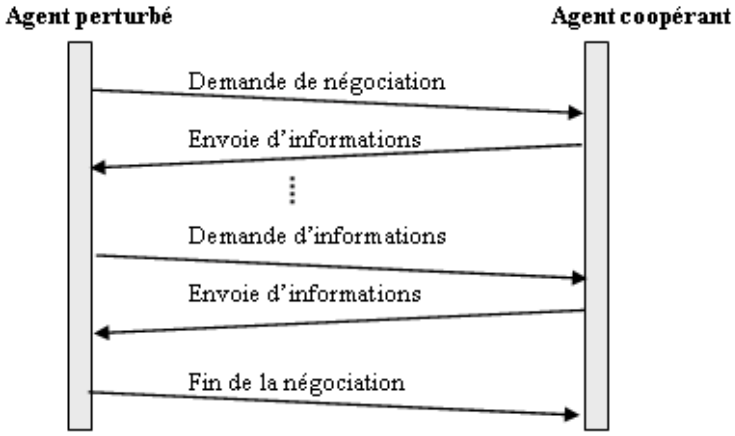


FIG. 3.3 – Interaction en réordonnancement local avec négociation

S'il y a toujours des contraintes qui ne sont pas satisfaites, l'agent entre dans le second protocole, dans ce dernier, l'agent perturbé envoie à tous les agents une requête demandant la négociation. Cette négociation a pour objectif d'allouer une ou plusieurs tâches de son plan. Pour ce faire, il diffuse un appel d'offre aux autres agents. À la reception de ce message, chaque agent doit estimer et attribuer l'utilité (ce n'est que le retard et le nombre de contraintes non respectées) de l'insertion de la tâche dans le plan de tâches local et l'envoyer en tant que proposition à l'agent perturbé. Après la réception des propositions, l'agent perturbé trie et sélectionne la meilleure offre (proposition) et confirme l'allocation de la tâche à l'agent correspondant et interrompe la négociation.

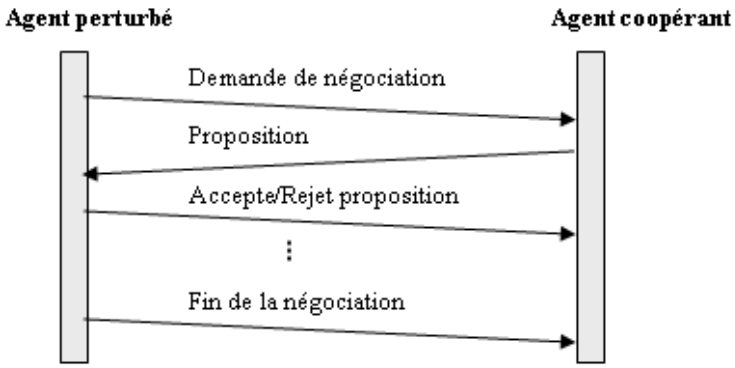


FIG. 3.4 – Interaction en réordonnancement global par allocation coopérative

S'il y a d'autres contraintes non satisfaites, l'agent perturbé passe au troisième protocole d'interaction, dans ce cas, l'agent perturbé envoie à chaque agent successeur une demande de négociation pour enrichir et mettre à jour sa base de connaissances selon les nouvelles connaissances imposées par l'agent perturbé. Chaque agent sollicité met à jour ses connaissances. S'il détecte une perturbation (des tâches dépassant leurs échéances), il doit faire une tentative pour l'absorber en exécutant la démarche complète de la méthode d'ordonnancement proposée.

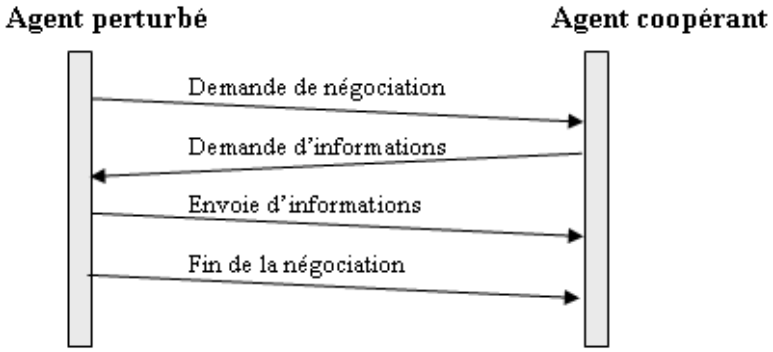


FIG. 3.5 – Interaction en réordonnancement global par propagation de contraintes

Le processus de réordonnancement peut être réitéré jusqu'à la résolution du problème. Cette approche peut s'apparenter à une résolution par négociation et propagation de contraintes. L'idée sous-jacente de cette approche consiste à restreindre la perturbation à un nombre minimal d'agents.

3.4 Modèle détaillé de l'architecture d'un agent-robot

Le modèle d'architecture d'un agent-robot représente son architecture générique (cf. Fig.3.6). Les agents sont homogènes dans leur structure et se différencient par leurs connaissances.

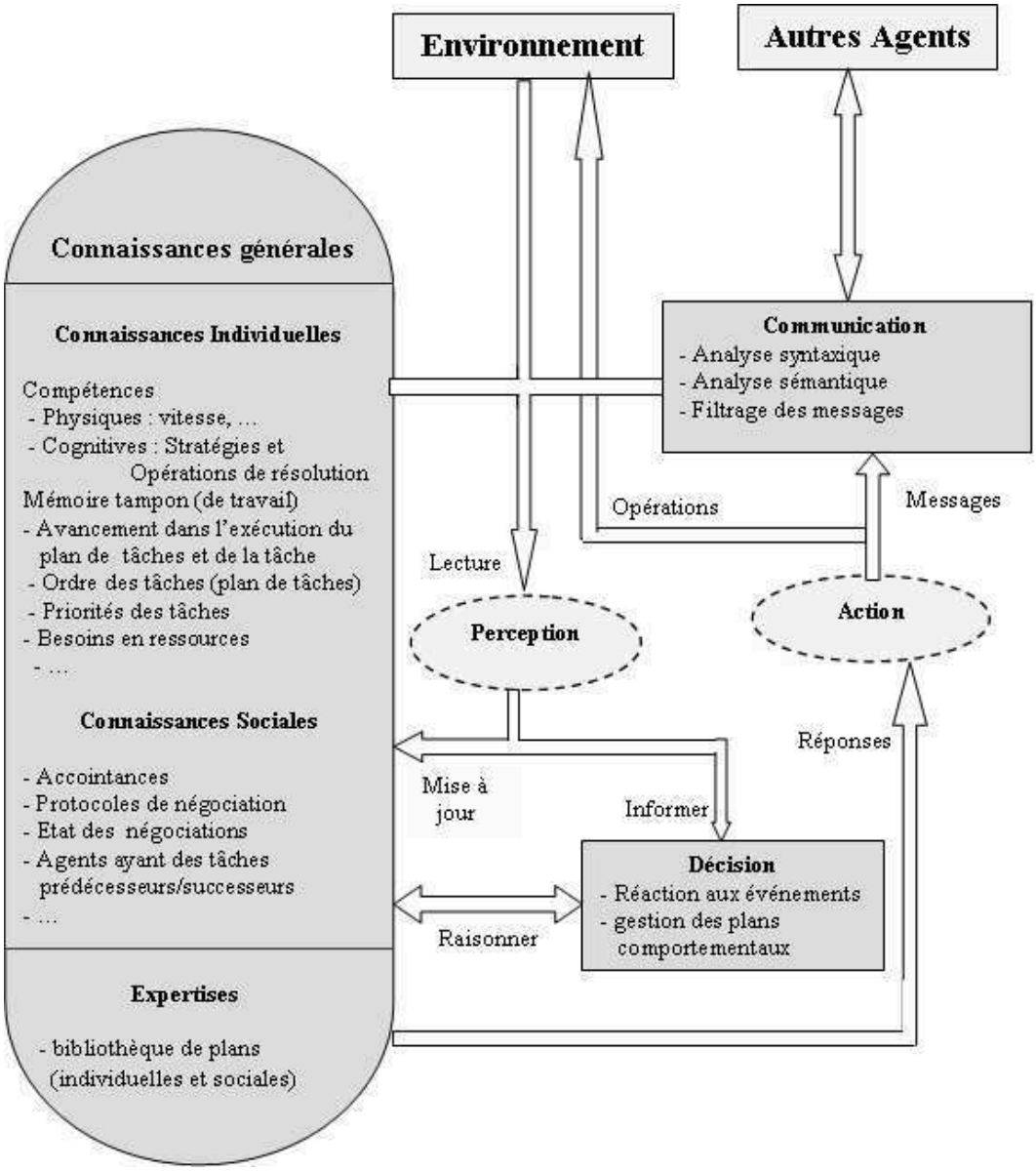


FIG. 3.6 – Architecture d'un agent-robot

Cette architecture montre qu'un agent-robot est constitué de trois modules fondamentaux lui permettant d'être capable d'accomplir certaines fonctionnalités (ou comportements : Perception, Communication, Raisonnement et action) : le module de Connaissances, le module de Communication et le module de Décision. Pour qu'un

agent-robot puisse réagir rapidement aux différents événements imprévus, ces modules doivent fonctionner en parallèle (en concurrence). Par exemple, un agent peut contrôler l'exécution d'une tâche de son plan de tâches et négocier en même temps. Pour cela, nous avons proposé un modèle de fonctionnement d'un agent-robot basé sur le concept de comportements. Cette architecture confère aux robots à la fois des capacités délibératives pour raisonner dans des situations complexes et des capacités réactives pour respecter des échéances. Plus concrètement, elle s'appuie sur une décomposition en modules concurrents (appelés "agents comportementaux", Section 3.5) dont l'interaction permet de doter le robot de comportements de type réactif, délibératif ou hybride. Les concepts développés dans ce chapitre constituent une réponse aux exigences suivantes :

- L'agent-robot doit avoir plusieurs comportements (interaction asynchrone et concurrente). Suivant sa situation, le robot doit mettre en œuvre des mécanismes de raisonnement intelligents sur ses tâches, son plan de tâches et les intentions avec les autres robots.
- L'agent-robot doit contrôler de manière autonome l'ensemble de ses tâches en fonction de son état interne et de son environnement. Il doit donc connaître à tout moment son état d'exécution qui est lié au contexte de son environnement (perturbations, présence d'une nouvelle tâche, etc.), s'y adapter et élaborer un nouveau plan de tâches.

3.4.1 Module de Connaissances

Le module "Connaissances" contient l'ensemble des informations détenues par un agent sur lui-même (connaissances individuelles), ou sur les autres agents de l'organisation (connaissances sociales). Les connaissances individuelles d'un agent constituent son identité, c'est-à-dire qu'elles identifient ce qu'il sait faire (compétences physiques et cognitives) et ce qu'il fait ou va faire (croyances et mémoire de travail). Les compétences physiques étant reliées au modèle du robot, elles sont les même pour tous les agent-robots. Les compétences cognitives énumèrent les capacités de coopération, de calcul d'ordonnancement, soit en d'autres termes son expertise. Les connaissances sociales caractérisent l'implication d'un agent dans le SMA. Elles portent tout d'abord sur les accointances d'un agent, c'est à dire l'ensemble des agents dont il a connaissance de l'existence. Cette connaissance peut être complétée par des extraits des connaissances individuelles de certains de ces agents. Lorsqu'un agent recherche une coopération avec d'autres agents il se réfère à ses connaissances sociales pour ne s'adresser qu'à un ensemble restreint et pertinent d'agents. Ce mécanisme minimise les flux de communication en évitant un recours systématique au *Broadcast*. Alors que les compétences dans le module connaissance décrivent ce que sait faire un agent, la partie "Expertise" détaille comment atteindre les compétences (physiques et cognitives). L'Expertise contient en

effet l'ensemble des Plans Comportementaux et des actions élémentaires exécutables par un agent. Un plan Comportemental spécifie une séquence d'actions élémentaires tels que l'envoi de message, des calculs divers (dont d'ordonnancement), la mise à jour des connaissances, etc. Nous précisons dans les paragraphes suivants comment cette spécification est faite.

3.4.2 Module de Communication

La communication inter-agents est à la charge de ce module et elle est effectuée explicitement par échange de messages. Ce module ne traite donc que les échanges entre les agents du système. Pour cela, l'agent utilise un module d'accointances structuré en une liste d'agents connus. La communication peut être de différents types. Elle peut être sélective, si l'agent lance un réordonnancement local avec négociation ou un réordonnancement par propagation de contraintes, tel qu'il s'adresse seulement aux agents qui ont des plans couplés avec son plan (mode "Multicast"). Par contre, dans une allocation de tâches il sollicite généralement tous les agent du système (mode "Broacast").

La gestion de l'envoi et de la réception des messages effectuée par ce module concerne le médium linguistique de la transmission des messages, c'est à dire l'expression et l'interprétation d'un message. Les messages doivent être exprimés correctement par un langage connu des agents et qu'ils portent sur des informations interprétables, pour qu'ils soient compréhensibles par les agents. Nous avons retenu, comme langage d'expression des messages le "Langage de Communication entre Agent" (ACL) de la FIPA [2]. ACL présente une sémantique plus rigoureuse ainsi qu'un effort important de standardisation. par rapport au langage KQML [22]

3.4.3 Module de Décision

Les capacités de décision de l'agent constituent le centre décisionnel des différentes compétences de l'agent. Il contrôle l'exécution des actions entreprises en se basant sur l'expertise de l'agent au niveau du module "Connaissances". Ce contrôle peut être délibératif, le responsable est le gestionnaire des plans comportementaux, ou réactif si la perception détecte des événements spécifiques. En fonction du comportement déclenché, un module de décision associé à ce comportement détermine une réponse adaptée à la situation dans laquelle se trouve le robot. En effet, il doit adapter les comportements d'un agent selon son contexte d'exécution. C'est à dire, il peut activer ou désactiver les agents comportementaux après avoir vérifié leurs conditions de déclenchement. Il maintient également à jour la mémoire de travail du module "Connaissance". Le caractère réflexe, comme nous le verrons plus loin, est pris en charge par des agents comportementaux dédiés.

3.5 Caractéristiques d'un agent-robot pour l'ordonnancement coopératif

Nous précisons dans cette section, le contenu des deux modules "Connaissance" et "Décision" d'un agent-robot, ainsi que d'autres capacités impotentes pour accomplir de manière robuste la mission qui lui est assignée.

3.5.1 Module de connaissances

La particularité du module "Connaissance" d'un agent-robot réside essentiellement dans les compétences physiques et cognitives de l'agent et ses croyances. La représentation des compétences physiques de l'agent-robot, énumèrent les spécifications techniques du robot à lequel il est associé. Alors que, les compétences cognitives sont constituées essentiellement des stratégies et des mécanismes de résolution de problèmes que l'agent peut appliquer pour bien accomplir sa mission. Alors que, ses croyances sont constituées de plan de tâches du robot qui lui est associé, ainsi que des éventuelles perturbations pouvant survenir. C'est sur la base de ces connaissances que le calcul de réordonnancement s'appuie. Le plan de tâches contient un ensemble de tâches décrites au travers des informations caractéristiques définies dans la Section 5 du chapitre 2. Le contenu du module de connaissances varie alors d'un agent-robot à un autre

L'expertise d'un agent-robot est essentiellement constituée par les méthodes de calcul de réordonnancement qui sont formalisées à l'aide de Plans Comportementaux (PC) dédiés pouvant, selon le cas, nécessiter une interaction (i.e. une coopération) avec d'autres agents ou non. Ces méthodes étant les mêmes pour tous les agent-robots, cette partie de connaissances est, a priori, commune à tous les agent-robots, mais, leur contexte d'application se diffère d'un agent-robot à un autre.

3.5.2 Module de décision

Le module de Décision d'un agent-robot est similaire à sa description générale applicable par tous les agent-robots. On peut remarquer seulement une particularité au niveau des modes de réactions aux événements et au niveau des critères de décision. Le mode de réaction assure la gestion des évènements perturbateurs et traite les requêtes provenant des autres agents. Comme nous avons évoqué, ce traitement peut se faire de manière indépendant puisqu'un agent peut informer ou négocier avec un autre agent, sans interrompre un comportement en cours. Cette remarque souligne la capacité d'un agent-robot à accomplir plusieurs activités en même temps.

La gestion des agents comportementaux n'intervient pas directement dans le processus de résolution. En effet, toutes les alternatives de traitement d'une perturbation sont décrites dans des plans comportementaux au niveau de l'Expertise de l'agent-robot.

3.5.3 La perception

Cette fonction s'occupe à créer et contrôler les représentations de la situation locale de l'agent. Aucun modèle global de l'organisation n'est employé. Les représentations global que les agents peuvent établir sont obtenues par échange de messages contenant les états internes des agents

Le contrôle continu de la situation du robot assure sa réaction rapide à l'occurrence d'événements imprévus à tout moment durant la réalisation d'une mission. La fonction de perception permet à l'agent-robot d'une part, de percevoir l'état de son environnement et d'autre part, de percevoir ses états internes. Dans le premier cas, la fonction de perception permet de collecter les informations issues des différents capteurs qui équipent un agent-robot. Pour assurer une telle capacité, un robot mobile doit être équipé de capteurs lui permettant d'appréhender son environnement (capteurs de proximité) et de se localiser par rapport à son environnement (capteurs odométriques, balises, etc.), ainsi que les mécanismes d'émission et de réception des messages. Dans le deuxième cas, la fonction de perception sous-entend une estimation des états internes de l'agent (état d'exécution d'une tâche, du plan de tâches, comportements déclenchés, etc.)

La perception permet à un agent d'acquérir différents types d'informations à partir des sources multiples (l'environnement et les autre agents) et de les accumuler dans une zone de stockage gérée par le module de connaissances et employée par le module de décision pour déterminer les prochaines actions du robot. Par conséquent, elle doit mettre à jour ces informations de façon continue pour qu'elles soient valides à tout moment durant l'exécution de la mission.

3.5.4 Le raisonnement

Dans le contexte d'une application multi-robots, un agent-robot doit présenter plusieurs comportements intelligents :

- **Ordonnancement et négociation :** Cette capacité permet à l'agent de choisir la meilleure façon de réaliser son plan de tâches, c'est-à-dire le meilleur séquencement des actions pour réaliser ses objectifs suivant un critère qui est le plus souvent temporel (réalisation la plus rapide du plan). Plus concrètement, lorsqu'un agent est perturbé, celui-ci procède à un réordonnancement de ses tâches.

Deux situations peuvent alors se présenter : Dans la première, un ordonnancement est élaboré localement par un module d'ordonnancement sans sollicitation d'autres agents. Dans la seconde, l'agent met en œuvre un mécanisme de négociation avec les autres agents de façon à minimiser la propagation de la perturbation à ces agents. Ce mécanisme fait appel à une fonction de négociation dont le rôle est d'étendre le raisonnement à un ordonnancement coordonné. Ces comportements délibératifs seront décrits dans la suite de ce chapitre sous forme des plans comportementaux

- **Contrôle de l'exécution du plan :** durant l'exécution d'un plan, on peut distinguer plusieurs niveaux de contrôle. Le premier consiste à identifier des situations d'exception puis à déclencher au niveau de l'agent perturbé des comportements tendant à traiter ces exceptions. Ainsi dans le cas d'un déplacement, il s'agit de déclencher l'habileté de l'agent-robot à éviter un obstacle si ce dernier se trouve sur sa trajectoire nominale. Une telle situation engendre un examen de l'incidence de l'évitement sur le plan de tâches et un réordonnancement de ce dernier si nécessaire. Un autre exemple, est celui de l'agent qui reçoit une demande de négociation d'un autre agent. Une telle situation peut conduire à déclencher un comportement permettant la négociation pour résoudre collectivement un problème d'ordonnancement.
- **Réflexes :** en ce qui concerne le raisonnement de l'agent basé sur des habiletés, ceci se traduit par une simple réaction à un stimulus (présence d'un obstacle imprévu) par l'utilisation des informations partielles ou complètes rarement mémorisées (informations capteurs de proximité). Ce type de raisonnement conduit à des comportements réflexes de l'agent-robot que l'on peut qualifier d'innés.

3.5.5 L'actions sur l'environnement et sur les autres agents

Dans les capacités d'action d'un agent, on distingue deux types : celles qui agissent sur l'environnement et celles qui agissent sur les autres agents. Les actions qui agissent sur l'environnement réel de l'agent visent à satisfaire ses objectifs. Parmi ces actions, nous pouvons citer, le démarrage d'un plan, l'activation ou la désactivation d'une tâche. Certaines actions peuvent nécessiter la mise en œuvre d'une communication. Plus concrètement, ce sous-ensemble d'actions correspond à des situations où l'agent doit solliciter d'autres agents comme c'est par exemple le cas lors d'une négociation. Ainsi, ces actions sont définies comme des modules émetteurs de messages émis par un agent suivant un protocole d'interaction donné ; chaque message est fonction de l'état de l'agent, de son environnement et de ses croyances (l'état perçu des autres agents). Par ailleurs, dans le mécanisme d'interaction entre agents, un agent peut être sollicité par un autre pour lui fournir un service (partage de connaissances, examen d'insertion

d'une tâche). Dans ce cas, les réponses (messages) émises par l'agent sollicité constituent des actions sur l'agent solliciteur.

3.6 Les comportement d'un agent-robots pour la gestion de perturbations

Dans cette section, nous présentons plus en détail les comportements de réordonnancement par lesquels un agent-robot doit être doté pour qu'il puisse gérer une perturbation. Ces comportement doit agir à deux niveaux : le niveau local et/ou le niveau global

Le premier que nous appelons Agent Comportemental Stratégique Individuel est dédié à la gestion de la stratégie individuelle de l'agent par rapport à son plan de tâches.

Le second que nous appelons Agent Comportemental Stratégique de Groupe est quant à lui dédié à la gestion de la stratégie collective de l'agent ; c'est-à-dire la gestion du plan de tâches de l'agent en collaboration avec les autres membres de l'organisation. Les comportement d'un agent comportemental sont représenté par des graphes à états que nous appelons Plans Comportementaux, ces plans appartiennent à la bibliothèque des plans au niveau de l'expertises de l'agent. L'activation et la désactivation de ces plans (ou agent comportementaux) est contrôlé par un autre agent comportemental que nous appelons Agent Superviseur qui supervise le traitement d'une perturbation.

3.6.1 Définition d'un agent comportemental

Un agent comportemental peut être défini comme étant l'entité virtuelle, autonome et experte qui confère à l'agent-robot un certain comportement face à son environnement et aux autres agents. Chaque agent comportemental doit d'une part, être doté d'une certaine autonomie permettant à l'agent-robot d'évoluer sous sa seule influence et d'autre part, posséder une fonction de satisfaction explicitable selon le type de comportement (éviter les obstacles, ordonnancer, négocier, etc.). La mise en œuvre d'un agent comportemental se traduit par l'activation d'une ou de plusieurs capacités (cf. Fig.3.10). Ce mécanisme d'activation peut être vu comme un service demandé par un agent. Un exemple typique d'encapsulation de capacités est celui de l'ordonnancement coordonné de tâches d'un agent-robot. Un tel comportement utilise les services des modules de communication, de décision et de connaissances.

Un comportement est la façon dont un agent agit. Le comportement du même agent

peut varier selon le contexte dans lequel il se trouve, c'est-à-dire la même structure peut donner naissance à plusieurs comportements différents. Le comportement est également une notion relative : deux observateurs différents peuvent décrire le fonctionnement du même agent dans le même contexte comme deux comportements différents.

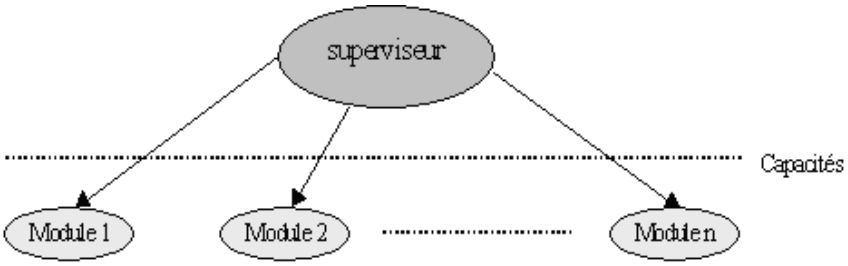


FIG. 3.7 – Agent comportemental agrégeant des capacités

3.6.2 Agent Comportemental Superviseur

Pour choisir l'agent comportemental adéquat au traitement d'une perturbation, on a besoin d'un agent superviseur pour piloter les autres agents comportementaux et contrôler le bon déroulement de la mission en cours. Son plan comportemental est représenté dans la Figure (Fig.3.8).

Pour ce faire, une première phase d'analyse du contexte de la perturbation s'avère obligatoire. Puis, une phase de sélection du comportement adéquat. Ensuite, la phase d'exécution (ou d'activation), ce qui va donner un nouvel ordonnancement (solution). En fin, une évaluation du nouvel ordonnancement permet de décider la mise en pratique ou non du nouveau plan de tâches obtenu (satisfaction des contraintes).

La première phase est une perception du contexte d'ordonnancement. L'agent collecte les informations d'ordonnancement, comme le type de la perturbation (panne, retard, nouvelles tâches, échec d'un agent comportemental précédent, etc.), l'urgence du calcule, etc.

En se basant sur les résultats de la phase précédente, on doit décider quel comportement doit être mis en œuvre. Par exemple, s'il s'agit d'une panne et que le robot ne peut plus exécuter quelques (ou toutes les) tâches de son plan, le réordonnancement avec allocation de tâches est activé ou si le réordonnancement local a échoué, le réordonnancement global doit être activé, etc. Du point de vue "réordonnancement", on distingue deux types de comportements, ordonnancement local et ordonnancement global

Dans la phase d'exécution, il ne nous reste qu'à activer le comportement sélectionné, le résultat de cette phase représente un nouvel ordonnancement (solution).

Pour évaluer la solution d'ordonnancement obtenue, une fonction d'évaluation est utilisée, cette fonction est composée d'un ensemble des sous fonctions, par exemple le retard moyen, le nombre de tâches dont les contraintes ne sont pas respectées, etc.

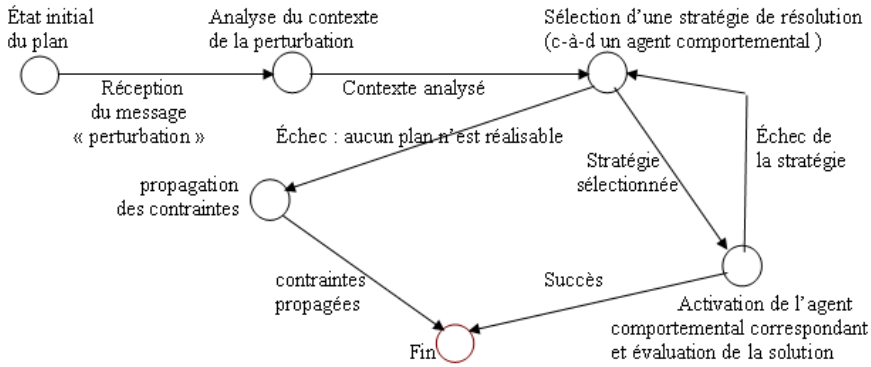


FIG. 3.8 – Plan comportemental de l'agent superviseur

3.6.3 Agent Comportemental stratégique individuel

Cet agent implémente la stratégie de résolution interne (local) du problème tel que, suite à l'occurrence d'une perturbation et après l'analyse de son contexte, l'agent-robot décide de mettre en œuvre le comportement stratégique individuel. Dans ce cas, l'agent-robot adopte un comportement délibératif fondé sur des connaissances disponibles au niveau de sa base de connaissances sans contacter aucun autre agent, afin d'élaborer un nouveau plan de tâches.

L'agent comportemental perçoit l'état interne de l'agent-robot perturbé, puis lance un cycle d'ordonnancement en se basant sur une algorithmique spécifique et en utilisant les informations collectées localement. L'état interne d'un agent-robot est caractérisé par les paramètres suivants : la nature de la perturbation, le retard engendré par cette dernière, le temps écoulé depuis le démarrage du plan, l'indice et le type de la tâches en cours d'exécution et enfin l'état d'exception vers lequel l'agent est entraîné. Après l'élaboration d'un nouvel ordonnancement (plan de tâches), le modèle d'informations ainsi obtenu, permet d'évaluer les effets du plan élaboré sur l'agent-robot lui-même et sur les autres agents. Les effets sur l'agent correspondent aux retards induits par la perturbation sur ses tâches et en particulier sur celles qui sont de type prédécesseur. Les effets sur les autres agents concernent les éventuels changements dans l'ordre d'exécution de certaines tâches de type successeur. Á la fin du processus d'ordonnancement, on distingue deux cas possibles : le premier correspond à l'obtention d'un plan de tâches qui satisfait toutes les contraintes (locales et globales) ; le second correspond à l'obtention

d'un plan qui ne satisfait pas au moins une contrainte. Notons que l'approche proposée pour l'ordonnancement est exposée plus en détail dans le chapitre suivant.

Si le plan de tâches obtenu satisfait toutes les contraintes, l'agent-robot continue la réalisation normal de son plan de tâches. Ce qui signifie la sortie de la situation d'exception et la reprise de l'exécution normale du nouveau plan de tâches. Dans le cas où le plan obtenu ne satisfait pas au moins une contrainte, l'agent superviseur génère un événement pour déclencher le comportement stratégique de groupe et activer ainsi l'agent comportemental associé

Le plan comportemental suivant décrit les comportement d'un agent pour un réordonnancement local. Dans ce cas, l'agent ne compte que sur ses propres connaissances pour gérer la perturbation.

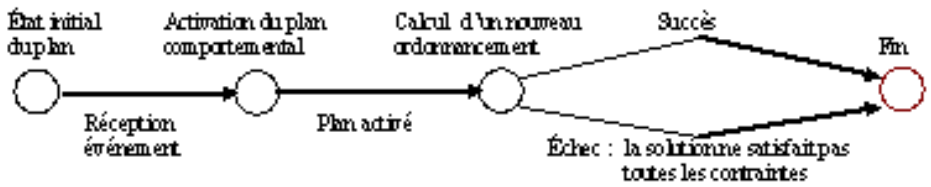


FIG. 3.9 – Plan comportemental pour l'ordonnancement local

3.6.4 Agent comportemental stratégique de groupe

Le comportement stratégique de groupe implémente les trois stratégies de négociation décrites dans le modèle d'interaction. Ce comportement est mis en œuvre par l'agent-robot quand ce dernier a échoué dans la phase d'ordonnancement local. Dans ce cas, l'agent-robot adopte un comportement délibératif lui permettant de rechercher une solution coordonnée du problème d'ordonnancement. Ce comportement vise à faire émerger un comportement global robuste de l'organisation, en s'appuyant sur la base de connaissances enrichie par les sources de connaissances associées aux autres agents de l'organisation. Les connaissances fournies par chaque agent sollicité sont essentiellement la liste des dates de début d'exécution des tâches de type successeurs. Á partir de ces connaissances, l'agent-robot évalue l'impact de la perturbation sur l'organisation. Le rôle de l'agent est ensuite de procéder à un traitement coordonné de la perturbation à partir des buts poursuivis par l'organisation, de son état interne (modèle mental) et de ses croyances.

Un agent sollicité ne communique pas tout son état interne à l'agent perturbé mais

plutôt une partie qui correspond à l'état d'exécution de son plan de tâches. Ces informations sont ensuite utilisées en vue d'effectuer un pré-traitement (priorités des tâches) destiné à l'ordonnancement. L'affectation des priorités s'effectue par délibération sur une base de connaissances distribuée. Cette base est constituée des connaissances situées au niveau de la base centrale de l'agent et de celles des sources de connaissances associées aux autres agents. Ce fonctionnement est équivalent à celui d'un tableau-Noir (Blackboard) de négociation, situé au niveau de l'agent-robot perturbé (cf. Fig.3.10).

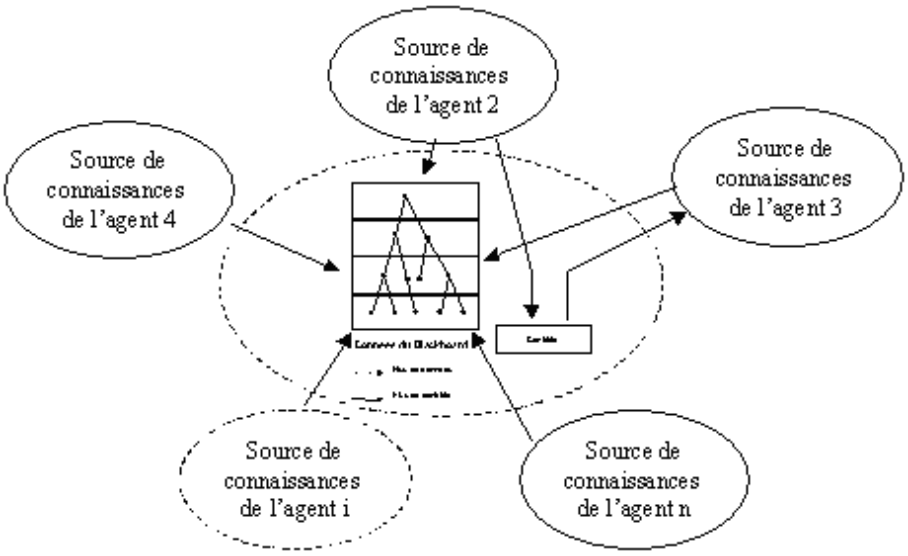


FIG. 3.10 – Blackboard de négociation

Ensuite, l'agent perturbé doit procéder à l'élaboration d'un nouveau plan de tâches qui satisfait toutes les contraintes à partir d'une algorithmique spécifique. Pour ce faire, il utilise en plus des paramètres décrits précédemment, les priorités des tâches, établies par la négociation. L'évaluation des effets du nouveau plan obtenu s'effectue de la même façon que dans le cas du comportement stratégique individuel. Ainsi que si le nouveau plan ne viole aucune contrainte sur les tâches, l'agent perturbé revient en un état normal.

Le plan comportemental suivant décrit les comportement d'un agent pour un réordonnancement local avec négociation, du point de vue de l'agent perturbé et de l'agent coopérant. Dans ce cas, l'agent compte non seulement sur ses propres connaissances pour gérer la perturbation, mais, aussi sur les connaissances des autres agents.

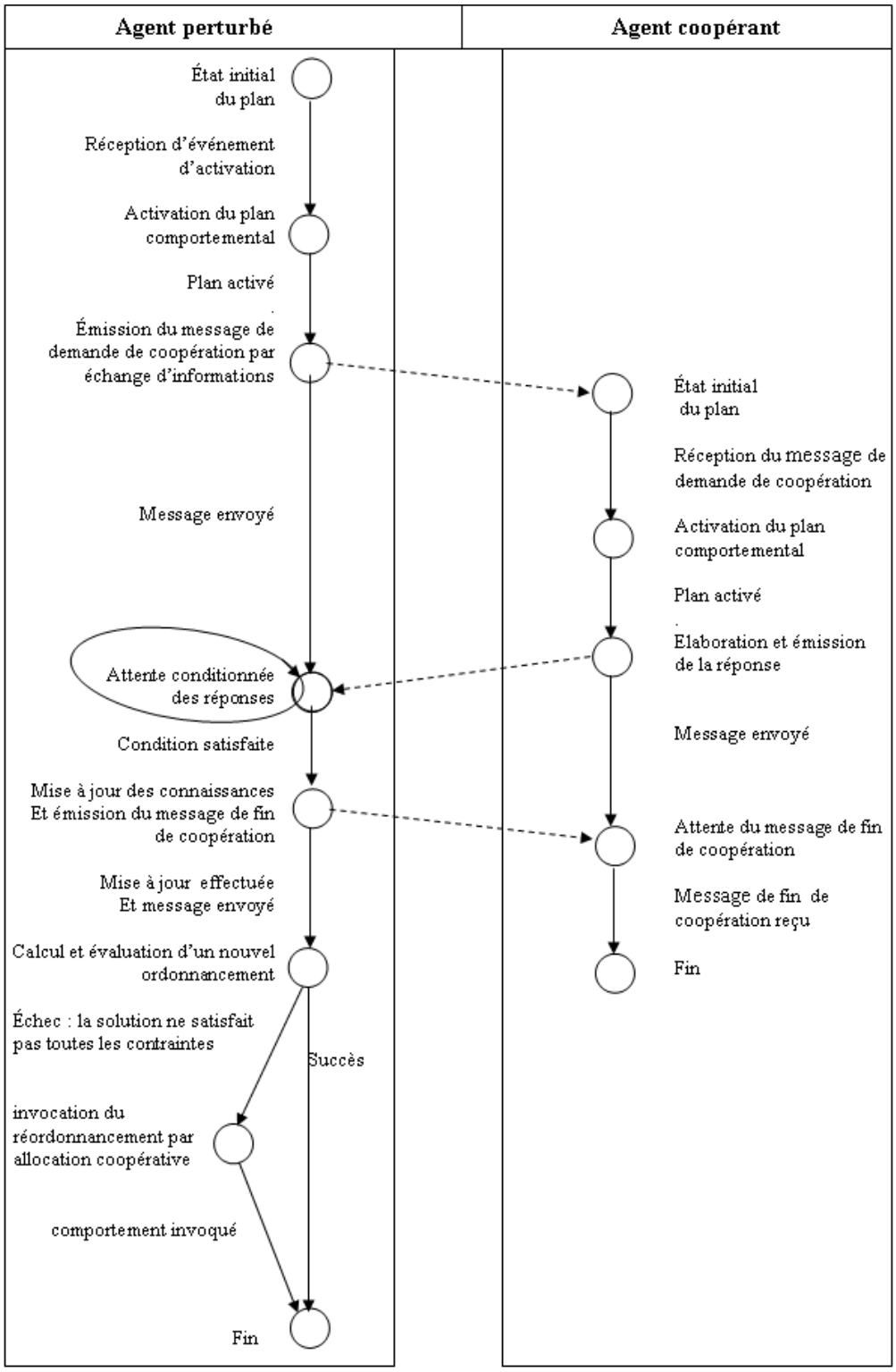


FIG. 3.11 – Plan comportemental pour l'ordonnancement local avec négociation

Si le nouveau plan ne satisfait pas toutes les contraintes, l'agent perturbé sollicite une nouvelle négociation. Mais cette fois-ci, la négociation vise l'allocation d'une tâche

à partir du plan de l'agent perturbé aux autres agents coopérants en vue de respecter la contrainte y associée et de libérer sa place dans le plan du premier agent qui peut être utilisée pour absorber l'effet de la perturbation.

Le protocole de négociation dans ce cas doit, être très simple afin d'assurer l'allocation de la tâche le plus vite possible. Nous avons donc décidé d'utiliser un protocole d'enchère à premier-prix et offre-cachée (first-price sealed-bid, cf. Fig.3.12), parce que ce protocole utilise moins de messages que d'autres protocoles tels que les enchères ascendantes, descendants [56], etc.

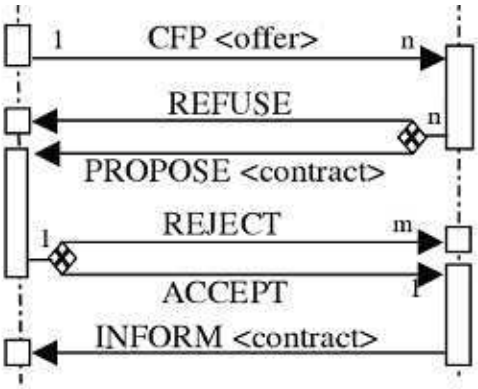


FIG. 3.12 – protocole de négociation pour l'allocation de tâches

L'évaluation des effets du nouveau plan de tâches obtenu s'effectue toujours de la même façon que dans le cas du comportement stratégique individuel. Ainsi que si le nouveau plan ne viole aucune contrainte sur les tâches, l'agent sorte de l'état d'exception vers un état normal.

Sinon, si le nouveau plan ne satisfait pas toutes les contraintes, l'agent perturbé sollicite une autre nouvelle négociation. Cette négociation à pour but la propagation des contraintes vers les agents dont les plans de tâches sont couplés. À l'occurrence d'une perturbation, il faut assurer dans tous les cas, l'existence d'au moins une solution réalisable à l'issue de la mise en œuvre de la méthode d'ordonnancement. Cette exigence est d'autant plus cruciale en situation d'inexistence d'une solution satisfaisant toutes les contraintes sur les tâches. La stratégie de *propagation amortie* de la perturbation permet de couvrir ce type de situation. Cette stratégie est assez primaire, puisqu'elle consiste à accepter les retards induits par la perturbation, c'est à dire à propager les effets de la perturbation sur les plans de tâches couplés. Cette propagation est néanmoins amortie du fait que chaque agent impliqué essaye de l'absorber à son niveau.

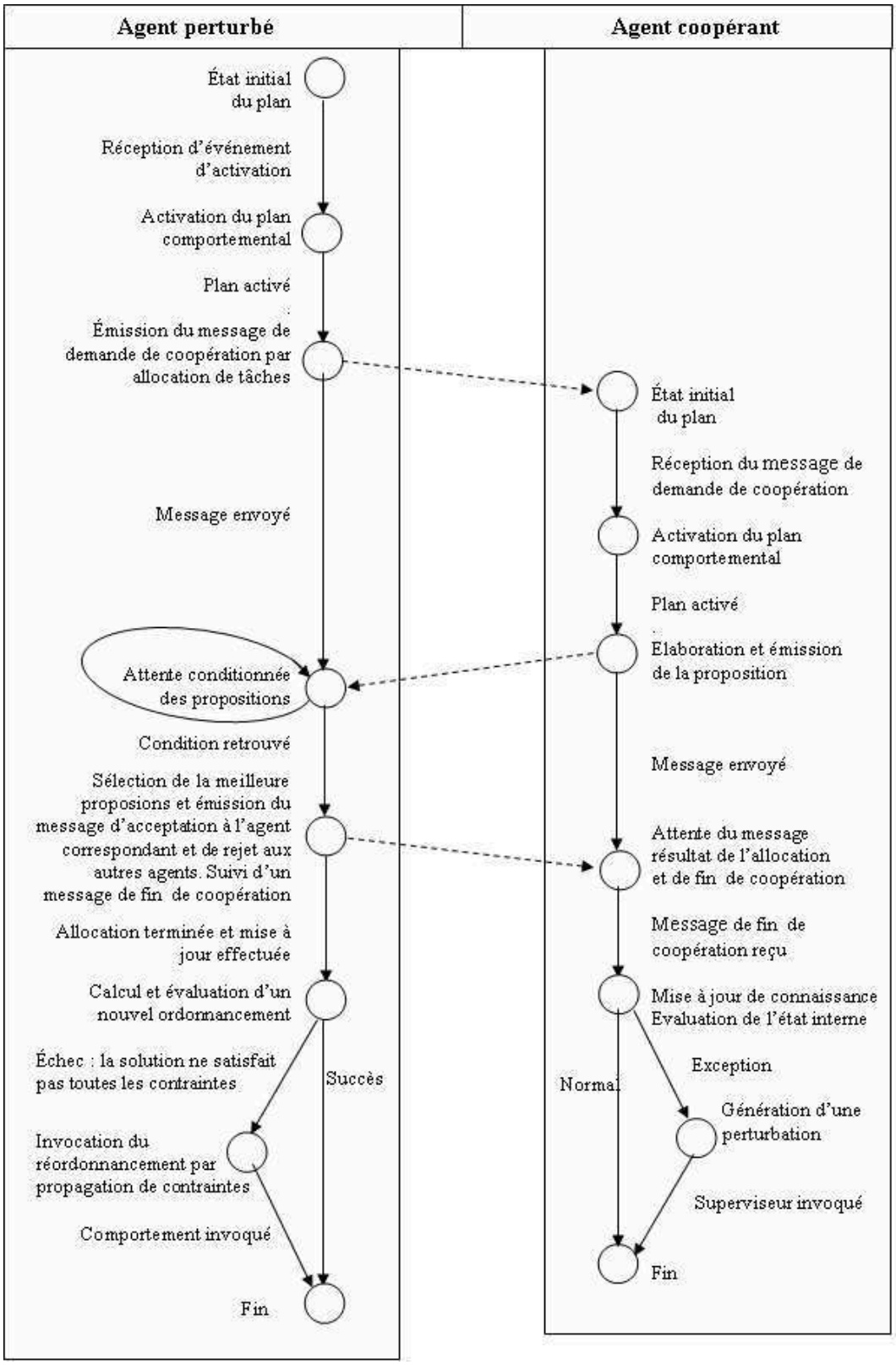


FIG. 3.13 – Plan comportemental pour l'ordonnancement global par allocation de tâches

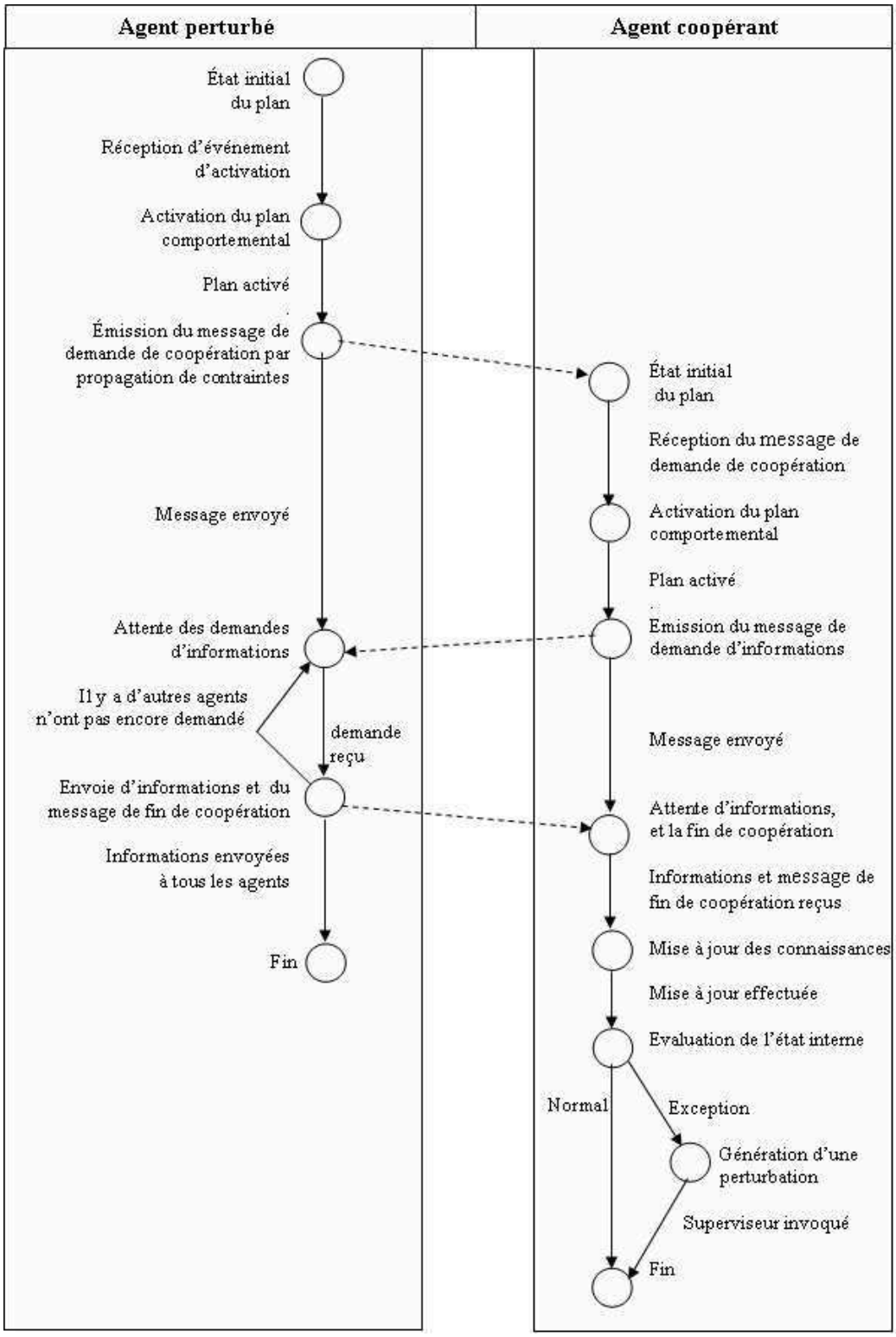


FIG. 3.14 – Plan comportemental pour l'ordonnancement global par propagation de contraintes

3.7 Conclusion

Nous avons présenté dans ce chapitre un modèle multi-agents pour la mise en œuvre d'applications robotiques complexes exigeant un ordonnancement de tâches sous de fortes contraintes de réactivité. Nous nous positionnons ainsi dans un contexte de minimisation de la remise en cause d'un ordonnancement construit initialement face à des perturbations tout en conservant les objectifs individuels et globaux de l'organisation.

Dans un premier temps, nous avons proposé un modèle d'organisation qui privilégie à la fois la distribution de la décision, l'autonomie et la réactivité des agents ainsi que la coordination décentralisée. Pour parvenir à une telle organisation, nous avons proposé une architecture d'agent-robot qui intègre une dimension à la fois délibérative et réactive. auxquelles des agents comportementaux sont associés, chacun met en œuvre une boucle de "perception-décision-action". Pour tenir compte du caractère dynamique de l'environnement et de l'organisation, et pour garantir une certaine cohérence du système, nous avons introduit un agent que nous appelions agent-central qui, permet l'introduction des paramètres de la mission, démarrage, suivi, etc.

Pour garantir la réactivité des robot face aux perturbations, nous avons défini deux agents comportementaux : Le premier que nous appelons agent comportemental stratégique individuel est dédié à la gestion de la stratégie individuelle de l'agent par rapport à son plan de tâches. Le second que nous appelons agent comportemental stratégique de groupe est quant à lui dédié à la gestion de la stratégie collective de l'agent ; c'est-à-dire la gestion du plan de tâches de l'agent en collaboration avec les autres membres de l'organisation. Ainsi, l'agent comportemental stratégique de groupe qui apparaît dans les phases de négociation a pour rôle de contribuer à faire émerger un comportement global robuste de l'organisation. La perception de cet agent comportemental est un partage de connaissances avec les autres robots pour la résolution collective de problèmes.

Dans le prochain chapitre, nous développons les solutions méthodologique et algorithmique que nous proposons pour la gestion des interactions entre les agents définis ci-dessus. Ces solutions repose sur des mécanismes de coordination et ont pour objectif d'aboutir en cas de conflit à un réordonnancement coordonné des tâches par négociation afin de satisfaire les contraintes temporelles et fonctionnelles imposées à l'organisation.

Chapitre 4

Modèle détaillé de coopération pour l'ordonnancement de tâches

4.1 introduction

Nous présentons dans ce chapitre, les stratégies de réordonnancement coopératif de tâches pour l'accomplissement robuste d'une mission multi-robots en temps réel. Compte tenu des contraintes de réactivité imposées, l'approche algorithmique proposée dans ce chapitre, doit être à la fois puissante et simple pour permettre la recherche rapide de solutions efficaces, tel qu'elle doit agir comme si le système global disposait d'une véritable vision globale. L'objectif de ce chapitre est de proposer un modèle coopératif détaillé pour la gestion de perturbations qui repose sur un traitement local des perturbations et en cas d'échec, conduit à la mise en œuvre de protocoles de négociation entre agents. Le but est de minimiser la propagation d'une perturbation se produisant au sein d'un agent aux autres agent-robots de l'organisation. Le modèle proposé met en œuvre un réordonnancement de plans avec une formalisation permettant l'analyse qualitative et quantitative des effets d'une telle approche sur le comportement de l'organisation.

Dans le chapitre précédent nous avons vu que, la méthode de réordonnancement proposée est basée sur quatre mécanismes de résolution qui sont activés à tour de rôle selon le type de perturbation, dans la plupart des cas, ils sont activés dans l'ordre suivant :

1. **Réordonnancement local :** dans ce cas, l'agent perturbé ne compte que sur ses propres connaissances locales sans contacter aucun autre agent.

2. **Réordonnancement local avec négociation :** l'agent perturbé doit interroger tous les agents dont les plans sont couplés, en vue de mettre à jour ses connaissances locales, ce qui va mener les agents à une coopération informationnelle.
3. **Réordonnancement global par allocation de tâches :** l'agent perturbé doit interroger tous les agents, dont le but de leur déléguer une ou plusieurs tâches de son plan de tâches local.
4. **Réordonnancement global par propagation de contraintes :** l'agent perturbé informe les agents dont les plans sont couplés, qu'ils doivent mettre à jour leurs connaissances.

Le choix de cet ordre d'exécution est justifié par la complexité croissante des stratégies de résolution, ainsi que la qualité de la solution obtenue, comme le montre la suite de ce chapitre.

Á chaque occurrence d'un événement perturbateur, l'agent passe d'un état appelé "normal" vers un autre état dit d'"exception" et devient donc l'"agent perturbé", dans ce cas, ce dernier doit tenter d'absorber les effets de cet événement en activant d'abord le premier mécanisme e réordonnacement. Si ce dernier arriverait à neutraliser l'événement perturbateur, alors l'agent perturbé sort de l'état d'exception et revient dans un autre état normal. Par conséquent, il interrompe le processus de résolution. Sinon, l'agent perturbé repasse à un autre état d'exception et invoque le deuxième mécanisme avec désactivation du premier, et ainsi de suite. Ainsi, cette méthode présente une forte réactivité face aux événements perturbateurs et assure l'existence d'une solution dans toutes les situations possibles. En effet, à la fin de traitement d'une perturbation, l'agent perturbé revient dans un état normal avec absorption de tous ou au moins de certains de leurs effets sur sa mission locale et même globale.

La méthode de réordonnancement proposée se base sur un ensemble d'opérations de "réparation d'ordonnancement" permettant aux agent-robots d'élaborer localement une nouvelle solution malgré une vue partielle du problème d'ordonnancement. Ces opérations sont particulièrement adaptées à un contexte perturbé et nous pouvons l'intégrer facilement dans un processus de résolution distribué. Cette réparation permet d'éviter la remise en cause des efforts calculatoires faites précédemment pour calculer un ordonnancement initial de tout le système. Donc, le calcul d'un nouvel ordonnancement se focalise sur une partie de l'ancien ordonnancement, au lieu de relancer un calcul global.

Dans ce chapitre, nous mettons tout d'abord les hypothèses et les conditions pour pouvoir proposer une algorithmique d'ordonnancement, puis nous exposons plus en détail les opérations de réparation d'ordonnancement sur lesquelles est basée cette algorithmique. Nous présentons ensuite la formulation algorithmique des différents mécanismes de résolution avec leurs complexités.

4.2 Hypothèses

Avant d'exposer le modèle d'ordonnancement, nous formulons les conditions et les hypothèses suivantes :

- Les agent-robots constituant l'organisation sont homogènes et disposent des mêmes mécanismes de raisonnement.
- Aucun agent-robot n'a une connaissance complète sur l'organisation d'agents.
- Chaque agent robot connaît toutes les tâches successeurs des autres agent-robots, qui sont liées à ses propres tâches prédécesseurs.
- Les tâches ne peuvent pas être interrompues durant leur exécution (non préemptives), une tâche ne nécessite qu'un seul robot à la fois et un robot ne peut réaliser qu'une seule tâche à la fois.
- Une ressource est utilisée par un seul robot à la fois (contrainte disjonctive). Dans le cas d'une mission multi-robots, les ressources sont le plus souvent des zones de l'espaces de travail (couloirs).
- Les objectifs des agent-robots peuvent être distincts ou bien communs, et leurs plans sont dépendants les uns des autres. Donc, les agents doivent satisfaire ses objectifs tout en tenant compte de cette dépendance.
- Il existe un canal de communication fiable et persistant entre les robots. Néanmoins, il suffit que ce canal de communication soit actif au début de chaque mission, et pendant toute la durée de la négociation.
- Pour l'ordonnancement de tâches, nous privilégions la recherche d'une solution réalisable plutôt qu'optimale, ce qui va permettre une grande flexibilité du système.
- Tous les plans initiaux de tâches des agent-robots sont élaborés et synchronisés par l'agent-central avant le démarrage de la mission.
- Des nouvelles tâches soumises aux contraintes d'échéances peuvent se présenter au système après le démarrage de la mission.
- Toutes les tâches présentes avant le démarrage d'une mission sont placées à une durée fixe équivalente à la prévision initiale faite par un algorithme de prévision de durée implémenté dans le module de planification de l'agent-central avant l'élaboration des plans de tâches. Ces durées ne seront pas mises à jour durant toute l'exécution, seulement :
    1. Si une tâche dépasse sa durée d'exécution, cette durée est prolongée d'une unité de temps jusqu'à ce qu'elle soit complètement accomplie.
    2. Si une tâche finit tôt, sa durée dans le plan de tâches est rétrécie quand elle a accompli.
    3. L'ordonnancement utilise une heuristique de réparation qui modifie les dates d'exécution, mais seulement avec des tâches non exécutées.

4.3 Opérations de réparations d'ordonnancement

L'autonomie décisionnelle permet à un agent-robot de garantir un ordonnancement dynamique en prenant en compte les objectifs locaux et globaux, les informations perçues en temps-réel et les informations reçues à partir des autres agents. Pour ce faire, l'agent perturbé doit appliquer des opérations de réparation sur l'ancien ordonnancement. Nous exposons celles dont nos agent-robots doivent mettre en œuvre.

4.3.1 Décalage de tâches

Le décalage est un déplacement d'une tâche non exécutée vers la gauche ou vers la droite dans un plan de tâches.

Le décalage à gauche, essaye d'avancer autant que possible la date de début d'exécution d'une tâche non-exécutée. Dans la Figure (Fig.4.1.a), la tâche $T_{1.3}$ ne peut pas commencer avant la date $Min\_date_{(1.3)}$, donc, son exécution peut être avancée au maximum jusqu'à cette date. De même, la tâche $T_{2.4}$ ne peut pas débuter avant la date $Min\_date_{(2.4)}$ (cf. Fig.4.1.c). Min_date d'une tâche T_i représente la date de fin d'exécution de la tâche T_{i-1} qui la précède dans un même plan de tâches du robot_l ou bien, si T_i est de type successeur, sa Min_date est équivalente à la date de fin d'exécution de la tâche prédécesseur T_j correspondante dans le plan de tâches d'un autre robot_k, avec $i \neq j$. Dans le décalage à droite, on essaie de retarder autant que possible la date de début d'exécution d'une tâche non exécutée. Dans la Figure (Fig.4.1.a), normalement, la tâche $T_{1.3}$ doit être terminée avant la date $Max\_date_{(1.3)}$ (contrainte d'échéance), alors, son exécution peut être retardée au maximum jusqu'à cette date (cf. Fig.4.1.d). Par contre, dans le deuxième plan (cf. Figure (Fig.4.1.b)), on remarque que la date de fin d'exécution de $T_{2.4}$ n'est pas bornée. Max_date d'une tâche T_i représente la date de début d'exécution de la tâche T_{i+1} qui la suit dans un même plan de tâches du robot_l, avec $i \neq j$, ou bien, si T_i est de type prédécesseur, sa Max_date est équivalente à la date de début d'exécution de la tâche successeur T_j correspondante dans le plan de tâches d'un autre robot_k.

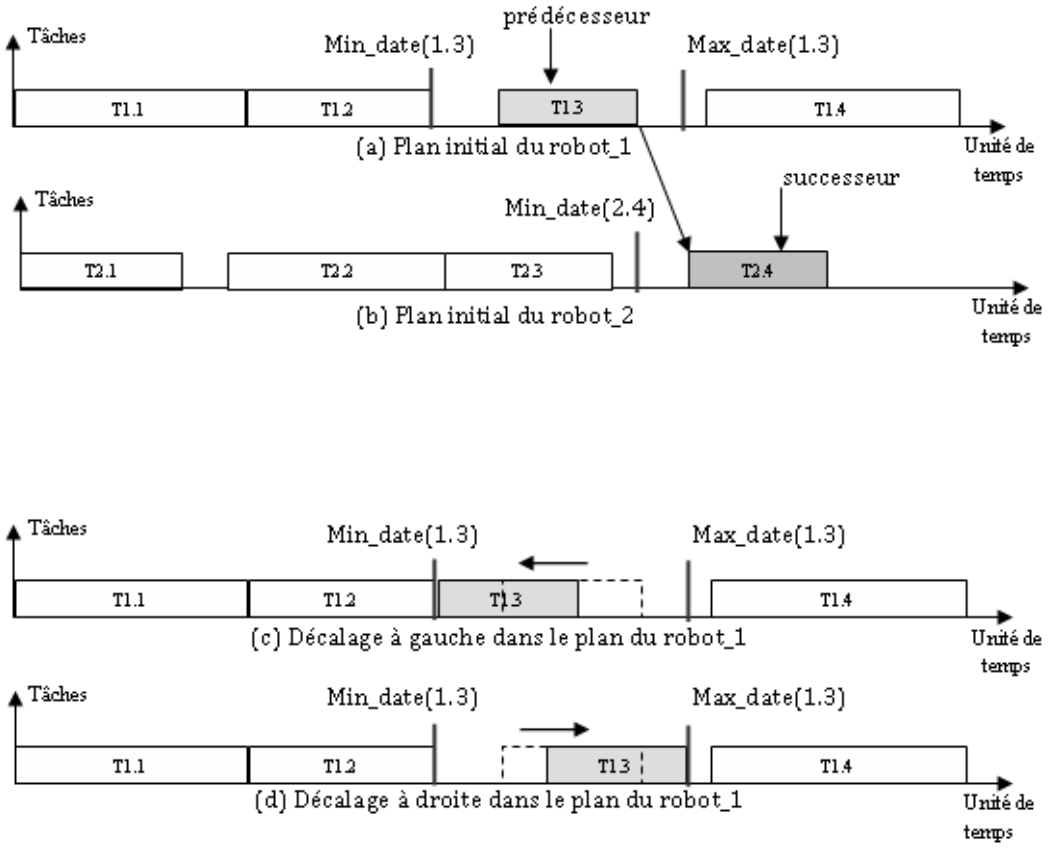


FIG. 4.1 – Exemple d'opérations de réordonnancement par décalage

4.3.2 Permutation de tâches

Elle permet d'échanger les emplacements de deux tâches non exécutées dans un même plan de tâches, c'est à dire échanger leurs dates de début d'exécution. Elle est un peu plus complexe que les opérations de décalage. En fait, il faut prendre en compte les contraintes de ces deux tâches.

La permutation ne peut s'effectuer que si les fenêtres temporelles (c'est l'intervalle temporel [Min_date, Max_date]) des tâches se croisent suffisamment. Par exemple, dans la Figure (Fig.4.2.a), les fenêtres temporelles des tâches $T_{1.4}$ et $T_{1.5}$ sont suffisantes pour leur permutation (cf. Fig.4.2.b), ceci permet de respecter $Max\_date_{(1.5)}$. Par contre, la fenêtre temporelle de la tâche $T_{1.3}$ n'est pas suffisante pour faire une permutation.

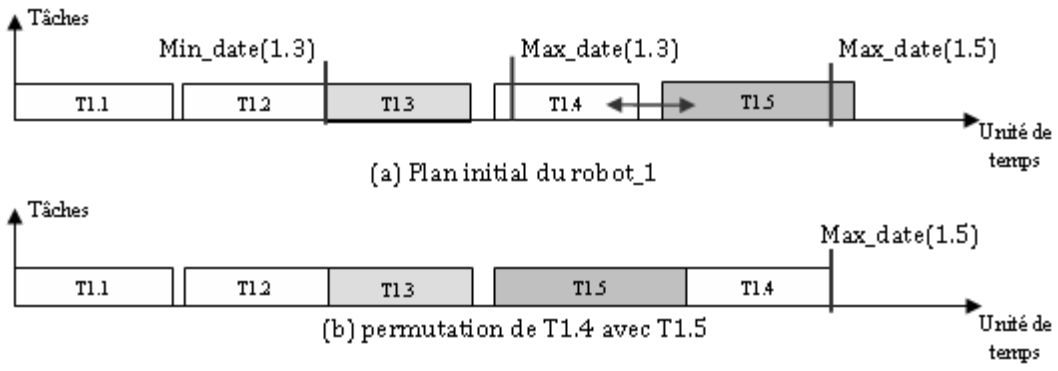


FIG. 4.2 – Exemple de réordonnancement par permutation

4.3.3 Insertion et annulation de tâches

Ces opérations sont liées à un contexte d'interaction entre plusieurs agent-robots ou entre l'agent-central et les agent-robots pour l'allocation d'une tâche. Rappelons que l'allocation se base sur un algorithme d'enchère adapté.

Dans le premier cas (entre agent-robots), l'agent-robot perturbé joue le rôle du manager (administrateur de l'enchère) et les agent-robots jouent le rôle d'offreurs (bidders, en anglais). L'agent-robot manager et l'agent-robot offreur sélectionné (gagnant) doivent réordonnancer leurs plans de tâches initiaux afin prendre en compte les dernières modifications dans leurs plans de tâches : transfert d'une tâche à partir du plan de l'agent manager vers le plan de l'agent offrant, c'est à dire que la tâche annulée par le premier agent-robot doit être insérée dans le plan du deuxième agent-robot.

Dans le deuxième cas (entre l'agent-central et les agent-robots), c'est l'agent-central qui joue le rôle du manager et les agent-robots jouent le rôle des offreurs. Dans ce cas, seulement L'agent-robot gagnant doit réordonnancer son plan de tâches initial pour prendre en compte les modifications dans son plan de tâches : insertion de la nouvelle tâche.

Dans la Figure (Fig.4.3), après un tour d'enchère, le premier agent-robot (cf. (Fig.4.3.a)) qui est le manager décide d'allouer la tâche $T_{1.2}$ au deuxième agent-robot (cf. (Fig.4.3.b)), donc le premier supprime cette tâche de son plan (cf. (Fig.4.3.c)) et le deuxième doit l'insérer (cf. (Fig.4.3.d)).

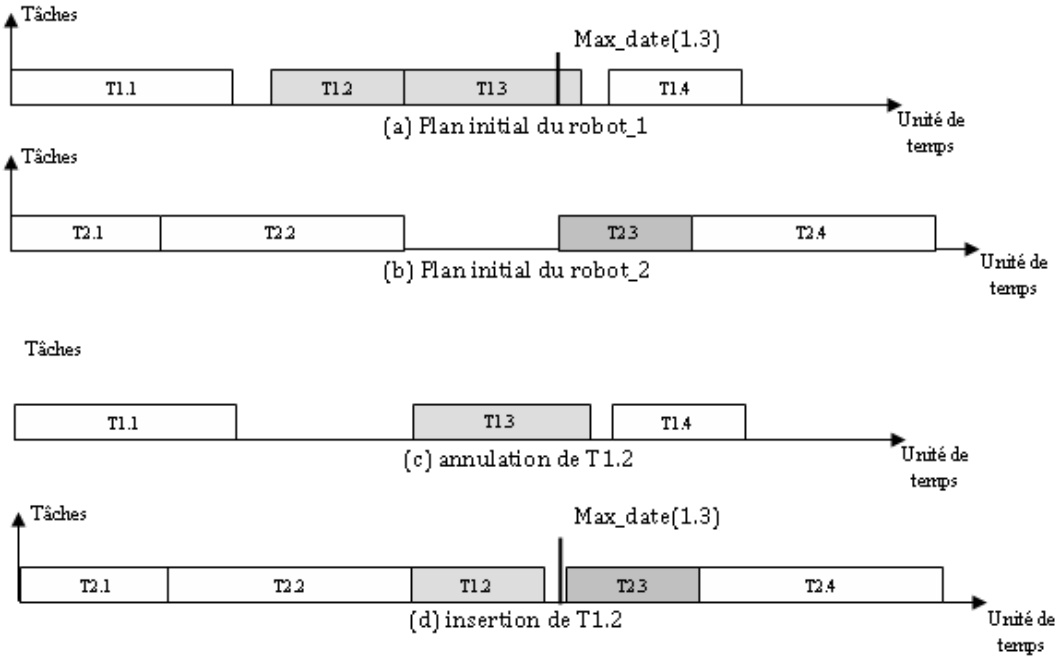


FIG. 4.3 – Exemple de réordonnancement par insertion et annulation de tâches

Dans la section suivante, nous présentons un formalisme de représentation de notre approche pour la coopération entre les agents qui Contrôlent une organisation de robots. L'objectif de cette approche est la gestion coordonnée d'une perturbation se produisant au sein d'un agent-robot. Plus particulièrement, nous traitons le problème d'ordonnancement coordonné de tâches dans un système multi-robots soumis à des contraintes de type échéance et précédence entre tâches. Cette formalisation permettant une analyse qualitative et quantitative des effets d'une telle approche sur le comportement de l'organisation.

4.4 Définitions

4.4.1 Décision

On appelle décision toute action permettant de :

- Activer ou désactiver un agent comportemental (réordonnanceur local, etc.).
- Appliquer une opération de réparation (décalage, etc.).
- Accepter ou rejeter une proposition.
- Ignorer ou répondre à un appel d'offre (date dépassée, etc.).

4.4.2 État normal

À un instant donné t, un agent-robot est dans un état normal si son plan de tâches est en exécution normale et ne subit aucune perturbation (cf. Fig.4.4).

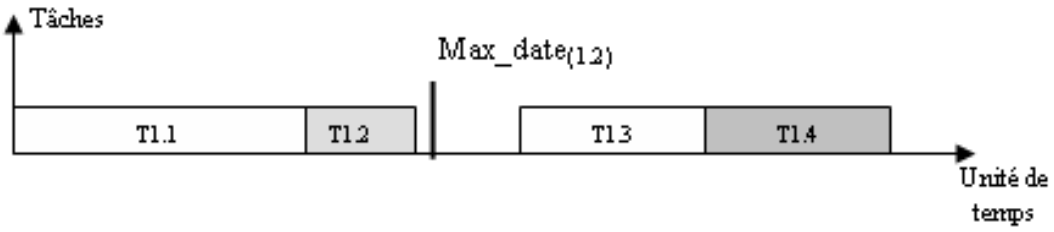


FIG. 4.4 – Exemple d'état normal d'un agent-robot

4.4.3 État d'exception

Un agent-robot est dans un état d'exception si une perturbation se produit pendant l'exécution d'une tâche de son plan de tâches (cf. Fig.4.5). L'agent-robot perturbé reste dans cet état depuis l'apparition de la perturbation jusqu'à la fin de son traitement, ainsi, l'agent-robot revient à nouveau en un autre état normal.

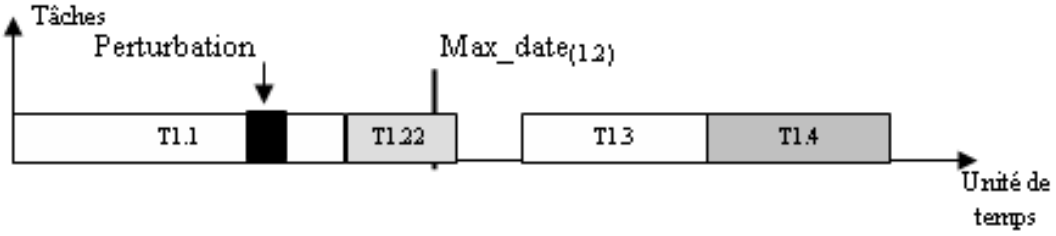


FIG. 4.5 – Exemple d'état d'exception d'un agent-robot

L'agent-central est dit dans un état d'exception, s'il y a de nouvelles tâches à réaliser, il est dans cet état d'exception dès la présence d'une nouvelle tâche jusqu'à son allocation.

Dans ce qui suit, nous présentons le modèle statique et dynamique de l'approche proposée.

4.5 Modèle statique

Le modèle statique représente une formalisation permettant de décrire le système (agent-central, agent-robot, ressources, les tâches à réaliser par les robots, etc.) en situation normale, c'est à dire en absence de perturbation :

- Ag_0 : l'agent central.
- Ag = $\{Ag_1, Ag_2, ..., Ag_n\}$: l'organisation d'agent-robot,
  Ag_i : l'agent-robot indicé i, $i \neq 0$.
- T = $\{T_1, T_2, ..., T_m\}$: l'ensemble des tâches réalisable par l'organisation de robots,
  T_j : tâche indicée j.
- PT_i : position de la tâche T_i dans le plan de tâches (son ordre) d'un agent-robot.
- $S_{i.j}$: la date de début d'exécution de la tâche T_j du plan de tâche de l'agent-robot Ag_i.
- $E_{i.j}$: la durée d'exécution de la tâche T_j du plan de tâche de l'agent-robot Ag_i.
- $Ep_{i.j}$: l'échéance de précédence de la tâche T_j de type prédécesseur du plan de tâche de l'agent-robot Ag_i.
- $Ec_{i.j}$: l'échéance de la nouvelle tâche T_j inséré dans le plan de tâche de l'agent-robot Ag_i.
- $Pr_{i.j}$: la priorité associée à la tâche T_j du plan de l'agent-robot Ag_i. Initialement vaut 0.
- P = $\{P_1, P_2, ..., P_n\}$: les plans de tâches des agent-robots,
  $P_i = \{T_{i.1}, T_{i.2}, ..., T_{i.j}\}_j \leq m$: plan de tâche de l'agent-robot Ag_i,
  $T_{i.j}$: la tâche j dans le plan de tâche de l'agent-robot Ag_i

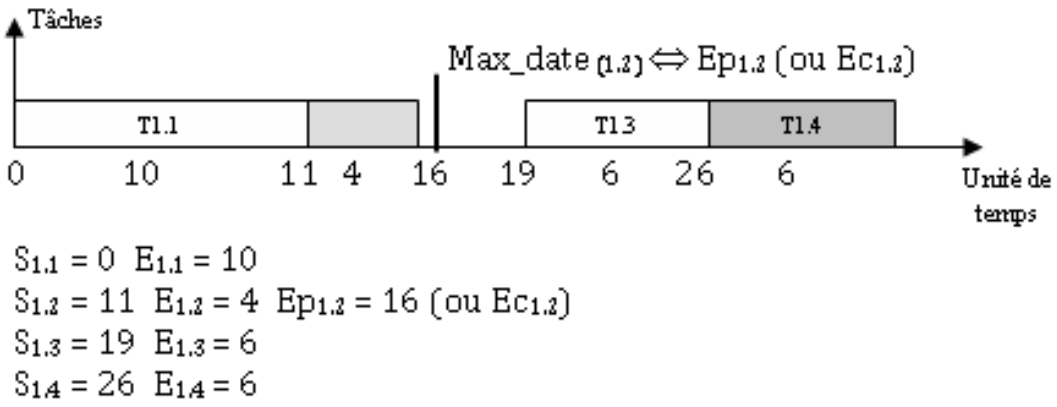


FIG. 4.6 – Exemple de plan de tâches

- $T_{i.p}$ (resp. $T_{i.s}$) : l'ensemble des tâches de type prédécesseur (resp. successeurs) situées dans le plan de tâches de l'agent-robot Ag_i.
- $T_{i.p.s}$ (resp. $T_{i.np.ns}$) : l'ensemble des tâches de type prédécesseur et successeurs

(resp. non prédécesseurs et non successeurs) situées dans le plan de tâches de l'agent-robot Ag_i.

- $StT_{i.j}$: l'état courant de la tâche T_j dans le plan de tâches de l'agent-robot Ag_i : en attente, en exécution, suspendue, terminée.
- $C_{i.j} = C_{j.k,i.l} \cup C_{i.l,j.r}$: l'ensemble des contraintes de précédence entre les tâches de l'agent-robot Ag_i et les tâches de l'agent-robot Ag_j (cf. Fig.4.7).

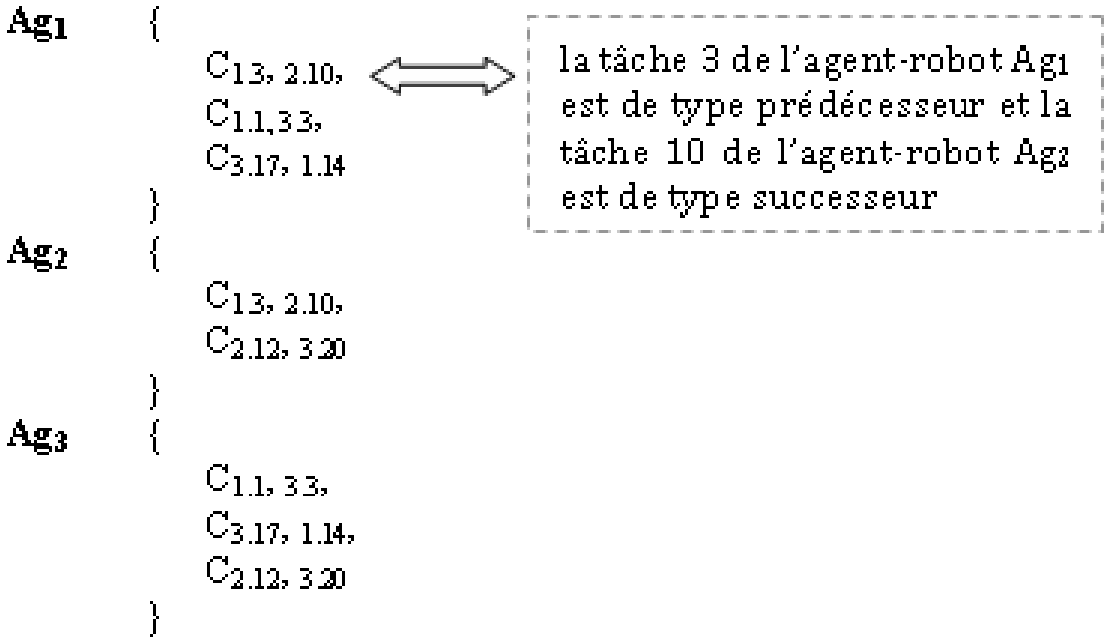


FIG. 4.7 – Exemple de contraintes de précédence

4.6 Modèle dynamique

Le modèle dynamique définit un formalisme permettant d'évaluer l'effet d'une décision prise par un agent-robot, sur l'agent-robot lui même et sur les autres agent-robots de l'organisation. Ce modèle permet d'analyser le comportement dynamique de l'organisation face à une perturbation qui se produit au sein d'un agent-robot membre.

4.6.1 Effets d'une perturbation

- $R_{i.j}$: le retard engendré par une perturbation sur la la tâche T_j de l'agent-robot Ag_i.
- $P_{i-c} = \{T_{i.1}, T_{i.2},..., T_{i.j}\}$: le plan de tâches de l'agent-robot Ag_i, en cours de réalisation.

- St = $\{St_1, St_2, ..., St_n\}$: l'ensemble des états des agent-robots,
  $St_i = \{St_{i.1}, St_{i.2}, ..., St_{i.x}\}$: l'ensemble des états de l'agent-robot Ag_i.
- StN = $\{StN_1, StN_2, ..., StN_r\}$: l'ensemble des états normaux des agent-robots,
  $StN_i = \{StN_{i.1}, StN_{i.2}, ..., StN_{i.y}\}$: l'ensemble des états normaux de l'agent-robot Ag_i.
- StE = $\{StE_1, StE_2, ..., StE_p\}$: l'ensemble des états d'exceptions des agent-robots,
  $StE_i = \{StE_{i.1}, StE_{i.2}, ..., StE_{i.z}\}$: l'ensemble des états d'exceptions de l'agent robot Ag_i , cet ensemble contient des situations à partir desquelles l'agent-robot peut sortir vers un état normal avec ou sans propagation de la perturbation aux autres agent-robots.
- D = $\{D_1, D_2, ..., D_q\}$: l'ensemble des décisions prises par les agent-robots.
  $D_i = \{D_{i.1}, D_{i.2}, ..., D_{i.m}\}$: l'ensemble des décisions prises par l'agent-robot Ag_i. On associe à cet ensemble, l'ensemble A, défini ci-après,
- A = $\{A_1, A_2, ..., A_q\}$: les ensembles des actions effectuées par les agent-robots pendant le processus de réordonnancement,
  $A_i = \{A_{i.1}, A_{i.2}, ..., A_{i.m}\}$: l'ensemble des actions effectuées par l'agent-robot Ag_i. Une action $A_{i.j}$ désigne l'action j effectuée par l'agent Ag_i. Elle correspond à $D_{i.k}$, décision k prise par ce dernier.

On peut définir l'effet d'une action réalisée par l'agent-robot Ag_i sur l'état d'un autre agent-robot Ag_j par la fonction suivante :

$$\Im_{i.j} : A_{i,k} \times St_{j.l} \longrightarrow St_{j.r}$$

Si l'agent Ag_j se trouve dans un état $St_{j.l}$, l'action $A_{i.k}$ prise par l'agent Ag_i conduit l'agent Ag_j vers l'état $\Im_{i.j}(A_{i.k}, St_{j.l})$. De même, si l'agent Ag_i se trouve dans un état $St_{i.l}$, l'action $A_{i.k}$ mène ce dernier vers l'état $\Im_{i.i}(A_{i.k}, St_{i.l})$. À partir de cette fonction de transition, on peut énumérer toutes les décisions prises au niveau d'un agent quelconque, qui peut produire une perturbation sur les autres agents. Si l'agent Ag_i se trouve dans un état d'exception $StE_{i.j} \in StE_i$, on définit alors l'ensemble suivant :

- $Ag_i(Pe_j) = \{Ag_j : Ag_j \in Ag \text{ et } i \neq j, \exists St_{j.l} \in Stj : \Im_{ij}(A_{i.k}, St_{j.l}) \in StE_i\}$: l'ensemble des agents qui peuvent être perturbés si une action $A_{i.k}$ est effectuée par l'agent Ag_i.

4.6.2 Paramètres à évaluer

Les effets d'une perturbation Pe_i sur un agent Ag_i au sein duquel se produit ou sur un autre agent Ag_j, sont évalués à partir des paramètres suivants :

- Re_i : Le retard induit sur le plan de tâches de l'agent Ag_i.

– $Td_i = \{Td_{i.1}, Td_{i.2}, ..., Td_{i.m}\}$: l'ensemble des tâches qui dépassent leurs échéances $(Ec_{i.j}, Ep_{i.j})$. Par conséquent, on peut obtenir l'ensemble des agent-robots affectés par ces dépassements.

Donc, la résolution du problème d'ordonnancement revient à optimiser deux critères, soit :

– Minimiser le retard induit par la perturbation au niveau des agents de l'organisation. De façon formelle, on écrit :

$$Min \sum_{i=1}^{n} Re_i$$

– Minimiser le nombre de contraintes non respectées. De façon formelle, on écrit :

$$Min(Card(Td_i))_{i=1..n},$$

4.7 Formulation Algorithmique

Nous présentons dans cette section les algorithmes sur lesquels repose l'approche d'ordonnancement proposée en les illustrant avec des exemples simples. Néanmoins, les cas pratique sont autant plus compliqués que ces simples exemples.

4.7.1 Algorithme de réordonnancement local

Lorsqu'une perturbation Pe_i se produit au niveau de l'agent-robot Ag_i, son agent superviseur sollicite les services du réordonnancement local. Dans ce cas, une heuristique permet d'élaborer un nouvel ordonnancement des tâches, à partir des connaissances a priori dont dispose localement l'agent. Les traitements associés à cette étape sont effectués par l'agent comportemental stratégique individuel associé à l'agent-robot perturbé. Pour ce faire, cet agent comportemental procède selon les étapes suivantes :

1. Identifier la perturbation Pei produite pendant la réalisation d'une tâche ainsi que le retard induit. Cette étape a pour objectif d'identifier l'état d'exception $StE_{i.j}(StE_{i.j} \in StE_i)$ dans laquelle se trouve l'agent
2. Prendre une décision $D_{i.k}(D_{i.k} \in D_i)$ qui détermine l'action $A_{i.k}(A_{i.k} \in A_i)$ à entreprendre dont l'application doit minimiser le plus que possible les effets de la perturbation (Re_i et Td_i). Cette action tente de ramener l'agent de l'état d'exception $StE_{i.k}$ dans lequel il se trouve vers un nouvel état normal soit l'état $\Im_{ii}(A_{i.k}, StE_{i.k}) = StN_{i.k}$.

3. Élaborer un nouveau plan de tâches en prenant en compte le retard induit par la perturbation tout en essayant de maintenir l'ordre initial des tâches (réordonnancement partiel). Soit Pn_i, ce nouveau plan de tâches.
4. Réitèrer les étapes 2 et 3, jusqu'à ce que toutes les tâches du nouveau plan respectent leurs échéances ou bien, il ne reste plus de décision utile à prendre.
5. S'il y a des contraintes non respectées, on procède par un réordonnancement complet.
6. Informer le superviseur du résultat.

L'agent comportemental stratégique individuel met en œuvre l'algorithme de réordonnancement ci-après :

Réordonnancement partial

DEBUT

1. L'agent perturbé décale à droite les tâches non exécutées de son plan de tâches, c'est à dire on propage la perturbation dans le plan local de l'agent-robot perturbé. Si toutes les contraintes de précédences sont respectées, le traitement doit être interrompu et l'agent-robot revient en un état normal. Sinon :
2. Créer et ordonner en ordre décroissant des échéances l'ensemble Cp_1, dont les éléments sont les tâches qui dépassent leurs échéances.
3. Créer et ordonner en ordre décroissant des échéances l'ensemble Cp_2, dont les éléments sont les tâches de type prédécesseur (appartiennent aux ensembles $T_{i.p}$, $T_{i.p.s}$ décrits dans le paragraphe 4.4) associées aux tâches de type successeur des autres agent-robots, et qui dépassent leurs échéances de précédence.
4. **Pour** d := 1 **jusqu'à** 2 **faire**
   Pour i := 1 **jusqu'à** $taille(Cp_d)$ **faire**
   Pour j := PT_c + 1 **jusqu'à** PT_i - 1 **faire** // PT_c : position de la tâche courante ;
   Si Début_exec(T_i) + $Max(Temps\_exec(T_i), Temps\_exec(T_j))$ ≤ Échéance(T_j)
   Et Début_exec(T_j) ≥ $Min_date(T_i)$ **alors**
   Si après la permutation, le nombre de contraintes violées ne serait pas changé **alors**
   $Permuter(T_i, T_j)$;
   Fin-si ;
   Fin-si ;
   Fin-pour ;
   Fin-pour ;
   Fin-pour ;

FIN.

S'il y a des tâches ne respectent pas leurs échéances on procède par :

Réordonnancement complet

DEBUT

1. Créer et ordonner l'ensemble Cp_1 des tâches soumises à une contrainte d'échéance.
2. Créer et ordonner l'ensemble Cp_2 des tâches de type prédécesseur et successeur.
3. Créer et ordonner l'ensemble Cp_3 des tâches de type prédécesseur.
4. Créer l'ensemble Cp_4 des tâches de type successeur.
5. Créer l'ensemble Cp_5 des tâches restantes.
6. Placer les tâches de l'ensemble Cp_5 dans l'ordre initial
7. Insérer chaque tâche de l'ensemble Cp_4 selon l'ordre établit, dans la première position supérieure ou égale à sa Min_date (en appliquant des décalages à droite).
8. Insérer chaque tâche de l'ensemble Cp_3 dans la première position permettant de minimiser le dépassement de sa Max_date.
9. Insérer chaque tâche de l'ensemble Cp_2 dans la première positon permettant de minimiser le dépassement de sa Max_date et supérieure ou égale à sa Min_date.
10. Insérer chaque tâche de l'ensemble Cp_1 dans la première positon permettant de minimiser le dépassement de sa Max_date.

FIN.

FIG. 4.8 – Algorithme de réordonnancement local

Par conséquent, on distingue 5 ensembles avec des priorités différentes selon :

$$Pr(Cp_1) > Pr(Cp_2) > Pr(Cp_3) > Pr(Cp_4) > Pr(Cp_5)$$

```
    Décaler_1(retard, d, f)
Début
        Fin_exec(T_d) := Fin_exec(T_d) + retard ;
        Pour i := d + 1 jusqu'à f faire
            j := Fin_exec(T_{i-1}) - Début_exec(T_i) ;
            Si j > 0 alors
                Début_exec(T_i) := Début_exec(T_i) + j ;
            Fin-si ;
        Fin-pour
    Fin.
```

Décaler_2(d, f) : elle est appliquée dans le processus de réordonnancement pour supprimer les intervalles inoccupés dans le plan de tâches. C'est un décalage à gauche.

```
Début
        Pour i := d jusqu'à f faire
            Si Fin_exec(T_{i-1}) ≥ Min_date(T_i) alors
                Début_exec(T_i) := Fin_exec(T_{i-1}) ;
            Sinon
                Début_exec(T_i) := Min_date(T_i) ;
        Fin-si ;
        Fin-pour
    Fin.
```

```
    Permuter (Ti, Tj) // Début_exec(T_i) > Début_exec(T_j)
Début
        X := Début_exec(T_i)
        Début_exec(T_i) := Début_exec(T_j) ;
        Début_exec(T_j) := x ;
        Permuter_pos(T_i, T_j) // permuter les positions de T_i et T_j dans le plan de tâches
        Si Temps_exec(T_i) - Temps_exec(T_j) > 0 alors
            Décaler_1(Temps_exec(T_i) - Temps_exec(T_j), PT_j + 1, PT_i − 1) ;
        Sinon
            Si Temps_exec(T_i) - Temps_exec(T_j) < 0 alors
                Décaler_2(PT_i + 1, taille (P_c)) ;
            Fin-si
        Fin-si
    Fin.
```

FIG. 4.9 – Algorithmes des fonctions de décalage et de permutation

La Figure (Fig.4.10) illustre un exemple simple de l'activité de l'agent comportemental stratégique individuel d'un agent-robot Ag_i perturbé en appliquant l'algorithme

de réordonnancement local.

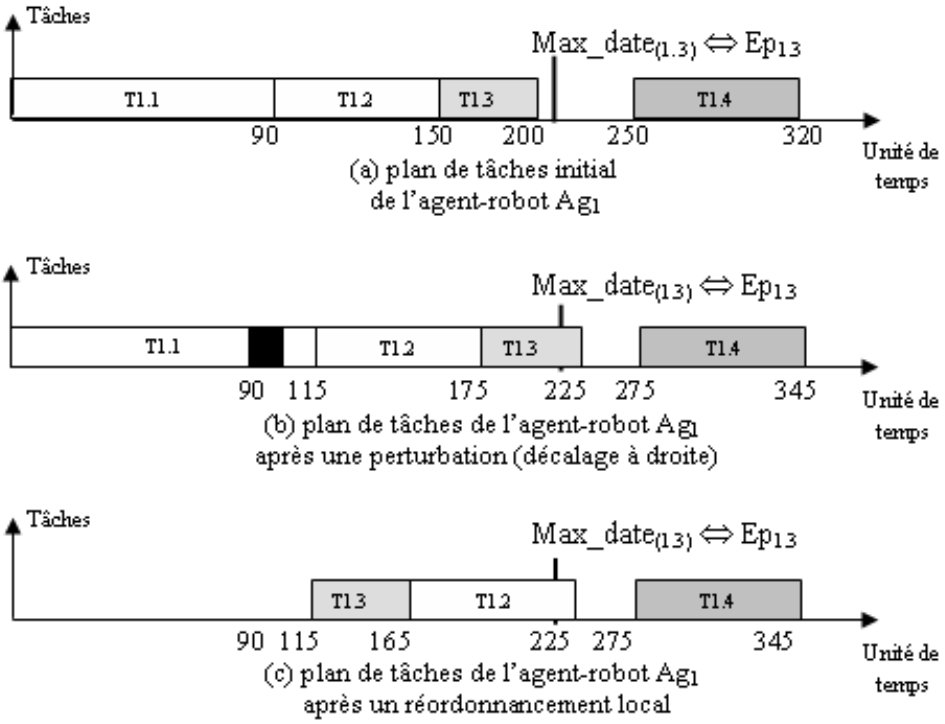


FIG. 4.10 – Exemple de réordonnancement local

Voici un exemple non complet de représentation de la mission :

Ag = $\{Ag_0, Ag_1, Ag_2\}$.

T = $\{T_{1.1}, T_{1.2}, T_{1.3}, T_{1.4}, T_{2.1}, T_{2.2}, T_{2.3}, T_{2.4}\}$.

P_1= {

$S_{1.1} = 0$ $E_{1.1} = 90$

$S_{1.2} = 90$ $E_{1.2} = 60$

$S_{1.3} = 150$ $E_{1.3} = 50$ $Ep_{1.2} = 210$ $S_{1.4} = 250$ $E_{1.4}$=70

}

$C_{1.2}$= {

$C_{1.3}, 2.4$

$C_{2.3,1.4}$

}

$T_{1.p} = \{T_{1.3}\}$

$T_{1.s} = \{T_{1.4}\}$

etc

Dans cet exemple, l'agent-robot Ag_1 a subit une perturbation durant la réalisation de la tâche $T_{1.1}$ de son plan initial (cf. Fig.4.10.a), cette perturbation se traduit sous la forme d'un retard de 25 ut (unité de temps) pour les deux tâches $T_{1.2}$ et $T_{1.3}$, alors que la tâche $T_{1.4}$ n'a subit aucun retard. Le nouveau plan de tâches de l'agent Ag_1 est obtenu par l'utilisation des capacités cognitives de l'agent comportemental stratégique individuel et sans contacter aucun autre agent-robot de l'organisation. Le réordonnancement consiste ici à décaler les tâches $T_{1.2}$ et $T_{1.3}$ vers la droite. Après l'analyse de ce dernier, l'agent trouve que la tâche $T_{1.3}$ dépasse son échéance de précédence, alors, il a décidé de permuter les deux tâches $T_{1.2}$ et $T_{1.3}$. Finalement, le nouvel ordonnancement respecte toutes les contraintes.

4.7.2 Algorithme de réordonnancement local avec négociation

Lorsque toutes les tentatives de réordonnancement local échouent, l'agent superviseur de l'agent-robot perturbé doit mettre en œuvre à la fois les capacités d'ordonnancement et de négociation. Il invoque alors le négociateur et attend que celui-ci mettre à jour les connaissances locales (ce n'est que les échéances de précédence des tâches de type prédécesseur, c'est à dire les dates de début d'exécution des tâches de type successeur associées et qui se trouvent dans les plans de tâches des autres agent-robots). Ce processus est observé donc, comme une coopération informationnelle entre les agent-robots. En suite, le négociateur doit attribuer les priorités aux tâches restantes (non-exécutées). Finalement un réordonnancement local de tâches doit être réexécuté. Le traitement effectué dans cette étape est assuré par l'agent comportemental stratégique de groupe. Les étapes suivantes résument le comportement d'un agent-robot perturbé Ag_i :

1. L'agent perturbé envoie une demande de négociation par partage d'information à tous les agent-robots dont les plans de tâches sont couplés. Les messages envoyés ne concernent que les tâches de type prédécesseur non exécutées (situées après la tâche perturbée), c'est à dire chaque message destiné à un agent-robot Ag_k contient seulement les noms des tâches de type successeur dans le plan de tâches de ce dernier et qui sont liées aux tâches de type prédécesseur de l'agent-robot Ag_i. ce qui permet d'envoyer un seul message par agent-robot impliqué. Aussi, ce message peut être considéré comme demande de négociation (implicite) et conteneur d'information. Il y a donc, minimisation des messages échangés. Le format d'un message de requête est le suivant :

```
(message-acl
    :performative
    :content
    :conversation-id
    :enveloppe
    :in-reply-to
    :language
    :ontology
    :protocol
    :receiver
    :reply-by
    :reply-with
    :sender
)
```

```
(message-sended
    request
    task_3 task_5
    agent_1-help-agent_2-6
    nil
    nil
    language_1
    ontology_1
    global_rescheduling
    agent_2
    now
    nil
    agent_1
)
```

FIG. 4.11 – Format d'une requête en réordonnancement local avec négociation

2. L'agent se met ensuite en attente des réponses de tous les agent-robots contactés. Chacun de ces derniers peut formuler leur réponses dans un seul message dans le but de minimiser encore les messages échanges. Le format d'un message de réponse est :

```
inform
task_3 =95 task_5 =160
agent_1-help-agent_2
nil
nil
language_1
ontology_1
global_rescheduling
agent_1
now
nil
agent_2
```

FIG. 4.12 – Format d'une réponse en réordonnancement local avec négociation

3. Une fois toutes les connaissances requises par le négociateur sont disponibles, l'agent-robot perturbé doit vérifier à nouveau son plan.
4. Si toutes les contraintes de précédences sont respectées, le traitement va être interrompu et l'agent-robot revient en un autre état normal.
5. Sinon, il suit les mêmes étapes et applique le même traitement que l'agent comportemental stratégique individuel.

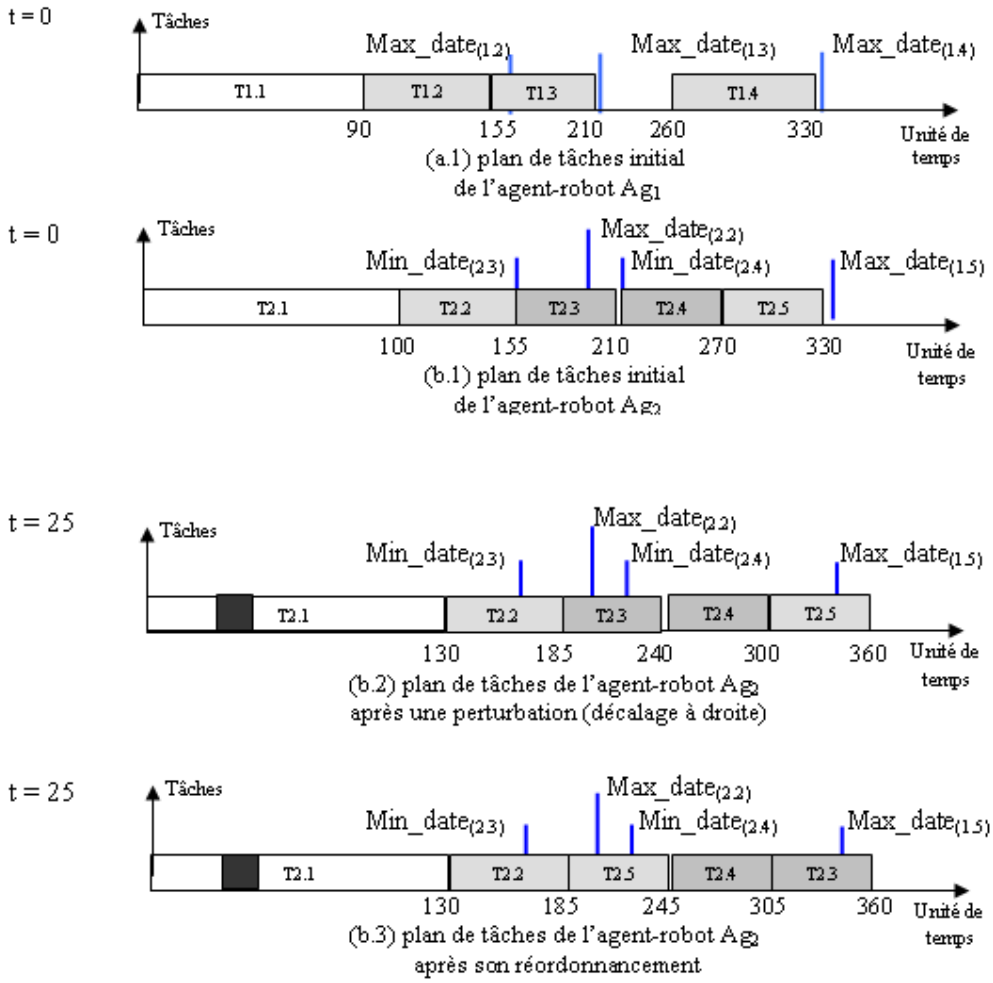
6. En plus, il doit informer les agent-robots coopérant des nouvelles dates de fin d'exécution des tâches de type prédécesseur associées à leurs tâches de type successeur pour qu'ils puissent mettre à jour les Min_date de ces dernières.
7. répéter les étapes de 1 à 6 jusqu'à ce que la négociation ne puisse apporter aucune amélioration pour l'ordonnancement en cours.
8. La coopération va être implicitement considérée comme terminée après un intervalle de temps donné ou à la réception d'une nouvelle demande de négociation.

Le langage de communication utilisé est celui d'ACL (Agent Communication Langage). Les champs d'un message ACL utilisés par les agent-robots sont :

- `performative` : *type de l'acte de communication.*
- `content` : *le message lui-même.*
- `conversation-id` : *identifiant de la conversation en cours.*
- `language` : *langage utilisé dans "content".*
- `ontology` : *le vocabulaire utilisé dans le message.*
- `protocol` : *protocole d'interaction utilisé.*
- `receiver` : *nom de l'agent récepteur.*
- `reply-by` : *indique la date/heure d'expiration de la réponse.*
- `sender` : *nom de l'agent émetteur.*

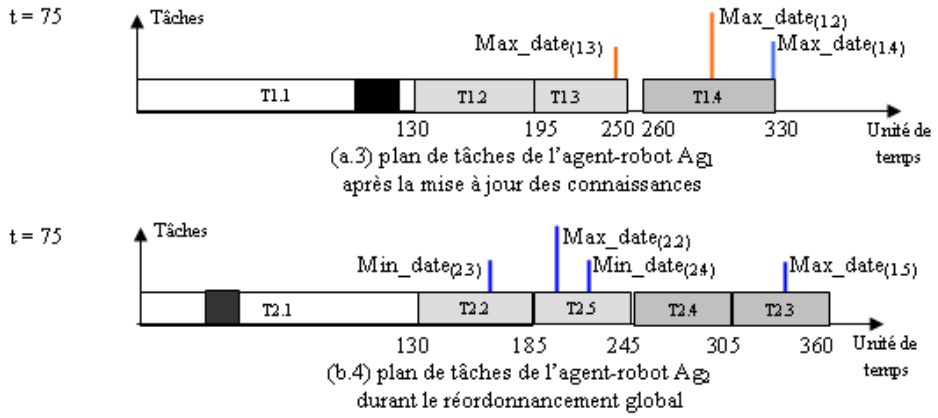
Pour illustrer cet algorithme de réordonnancement, nous présentons un exemple de deux agent-robots perturbés successivement (cf. Fig.4.13).

Dans un premier temps, une perturbation se produisant au sein de l'agent-robot Ag_2, lui oblige d'activer un processus de réordonnancement. Ce dernier a réussi à produire localement un ordonnancement valide (satisfait toutes les contraintes). Puis, dans un deuxième temps, l'agent-robot Ag_1 est soumis à une perturbation de type retard, il effectue d'abord un ordonnancement local, conduit à l'élaboration d'un nouveau plan ne satisfaisant pas toutes les contraintes de précédence, il va tenter alors d'absorber la perturbation par négociation. Finalement, les deux agents-robots ont pu sortir des états d'exceptions vers d'autres états normaux.



le réordonnancement local ne peut pas absorber la perturbation sur le plan de l'Ag1

=> il doit faire un réordonnancement global



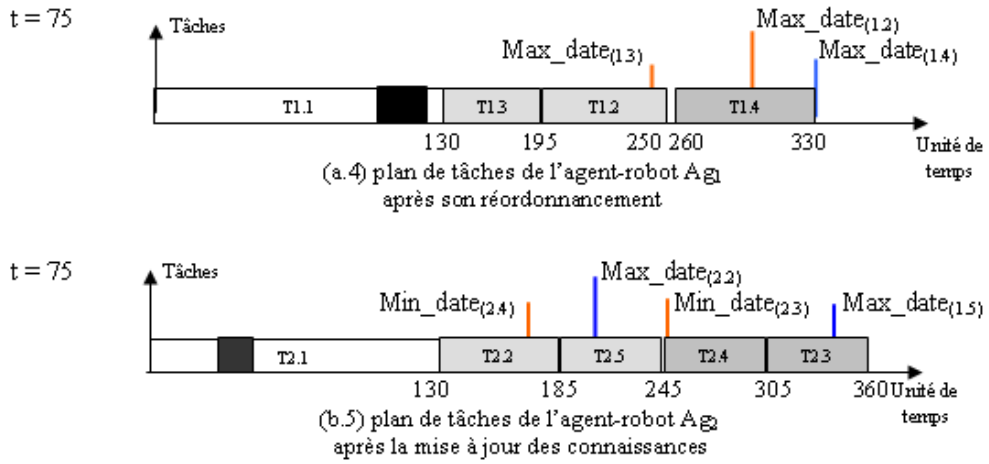


FIG. 4.13 – Exemple de réordonnancement local avec négociation

Dans cet exemple, le nouveau plan de tâches de l'agent-robot Ag_1 est obtenu par l'utilisation des capacités cognitives de l'agent comportemental stratégique de groupe après l'échec de l'agent comportemental stratégique individuel.

L'agent-robot Ag_1 a subit une perturbation avant 15 ut de la fin de la réalisation de la tâche $T_{1.1}$, c'est à dire à la date 75 (cf. Fig.4.13.a.2), cette perturbation se traduit sous la forme d'un retard de 40 ut pour les deux tâches $T_{1.2}$ et $T_{1.3}$ qui dépassent leurs échéances de précédence, alors que la perturbation n'affecte pas la tâche $T_{1.4}$. L'utilisation des capacités cognitives de l'agent comportemental stratégique individuel n'a apporté aucune solution réalisable à ce problème. Dans ce cas, l'agent-robot doit solliciter les services de l'agent comportemental stratégique de groupe. Plus précieusement, le service de réordonnancement local avec négociation. Le réordonnancement consiste d'abord à mettre à jour les connaissances centrales de l'Ag_1. Alors, il envoie le message suivant (les champs inutilisés ne seront pas indiqués) à l'Ag_2 lui demandant quelques informations :

request
T2.3 T2.4
Ag_1-help-Ag_2-6
l1
o1
global_rescheduling
Ag_2
now
Ag_1

FIG. 4.14 – Exemple de requête en réordonnancement local avec négociation

D'un autre coté, l'agent-robot Ag_2 qui a un plan couplé avec le plan de l'Ag_1 (cf. Fig.4.13.b.1), suite à une perturbation à l'instant 25 ut, qui se traduit sous la forme d'un retard de 30 ut (cf. Fig.4.13.b.2), l'Ag_2 se trouve obligé de faire d'abord un réordonnancement local de son plan initial, ce réordonnancement a permet de produire une solution valide (cf. Fig.4.13.b.3). Il consiste à permuter les deux tâches $T_{2.3}$ et $T_{2.5}$.

À la réception du message envoyé par l'Ag_1, l'Ag_2 doit répondre par l'envoi du message suivant :

inform
T2.3 =305 T2.4 =245
Ag_1-help-Ag_2-6
l1
o1
global_rescheduling
Ag_1
Ag_2

FIG. 4.15 – Exemple de réponse en réordonnancement local avec négociation

La tâche $T_{2.3}$ dans le plan de tâches de l'Ag_2 qui est la successeur associée à la tâche $T_{1.2}$ prédécesseur dans le plan de tâches de l'Ag_1, a été retardée jusqu'à la date 305 ut (cf. Fig.4.13.b.3), ce qui permet à l'Ag_1 d'augmenter l'échéance de précédence de $T_{1.2}$ vers 305 ut (cf. Fig.4.13.a.3) au lieu de 160 ut . De même, la tâche $T_{2.4}$ dans le plan de l'Ag_2 qui est la successeur associée à la tâche $T_{1.3}$ prédécesseur dans le plan de l'Ag_1, a été retardée de son tour jusqu'à la date 245 ut (cf. Fig.4.13.b.3), ce qui permet d'augmenter l'échéance de précédence de $T_{1.3}$ vers 245 ut (cf. Fig.4.13.a.3) au lieu de 210 ut. Après

la mise à jour des connaissances, seulement la tâche $T_{1.3}$ de l'Ag_1 dépasse son échéance de précédence. Ainsi, le réordonnancement du plan initial consiste ici à permuter les deux tâches $T_{1.2}$ et $T_{1.3}$. Ce qui donne un nouvel ordonnancement respectant toutes les contraintes de précédence (cf. Fig.4.13.a.4). En suite l'Ag_1 informe l'Ag_2 des nouvelles dates de fin d'exécution des tâches $T_{1.2}$ et $T_{1.3}$ en lui transmettant le message suivant :

```
inform
T1.2 =250 T1.3 =195
Ag1-help-Ag2-6
L1
O1
global_rescheduling
Ag2
Ag1
```

FIG. 4.16 – Exemple de message "inform" en réordonnancement local avec négociation

À la réception de ce message, l'Ag_2 doit mettre à jour ses anciennes connaissances (cf. Fig.4.13.b.5).

En fin, l'agent sort de l'état d'exception courant et rentre dans un état normal. Par conséquent, il décide d'arrêter le processus de résolution.

4.7.3 Algorithme de réordonnancement global par allocation de tâches

Le protocole de négociation utilisé dans cette étape de résolution est un protocole d'enchère de type *premier-prix_ offre-cachée* (section 3.5.3) adapté. Le processus d'allocation peut être mis en œuvre par un agent-robot perturbé et par l'agent-central (à la présence d'une nouvelle tâche).

Les étapes à suivre par l'agent-central sont :

- Il initie une enchère pour cette tâche en diffusant un message de négociation pour l'allocation de la tâche, ce message contient les informations nécessaires de cette dernière (description, échéance). Le format d'un message d'appel d'offre est similaire au message de requête utilisé dans l'étape de réordonnancement local avec négociation sauf le champ *"protocol"* qui deviendra *"task_ allocation"*.
- Il se met en attente des propositions des agent-robots. Plus précisément, il attend deux paramètres de chaque agent-robot sollicité $(Ag_i)_{i\neq 0}$. Ces paramètres traduisent les effets de la perturbation sur chaque agent-robot : Re_i et Td_i, décrits

dans le paragraphe 4.5.2

- À la réception du message d'appel d'offre, l'agent-robot Ag_j sollicité doit estimer le coût (effets : Re_j et Td_j) d'insertion de la tâche offerte dans son plan de tâches courant et envoie ces informations en tant qu'offre (proposition) vers l'agent-central. La structure d'un message de proposition est similaire au message de réponse utilisé dans l'étape de réordonnancement local avec négociation sauf le champ *"protocol"* qui deviendra *"task_allocation"*.
- Une fois que toutes les propositions soient disponibles ou après l'expiration d'un délai(pour éviter l'attente illimitée), l'agent-central sélectionne et attribue la tâche à l'agent-robot gagnant (correspondant à la meilleure proposition : $Min(Re_i)$ et $Min(Card(Td_i))$. Ce qui va probablement conduire à l'apparition d'une perturbation dans son plan de tâches. S'il est le cas, il doit suivre toute la démarche que nous décrivons ici, mais après l'obtention d'une permission à partir de l'agent-central.
- La coopération va être implicitement considérée comme terminée après un intervalle de temps donné ou à la réception d'une nouvelle demande de négociation, ce qui va diminuer encore le nombre de messages échangés.

Un agent-robot doit appliquer ce mécanisme dans les deux situations suivantes :

- Défaillance des outils utilisés par l'agent-robot. S'il n'est plus capable de réaliser toutes ou une partie de ses tâches non exécutées, il doit négocier avec les autres pour leur allouer ces dernières. Il met en œuvre à la fois les capacités d'ordonnancement et de négociation.
- Les tentatives de réordonnancements locaux ont échouées. Dans ce cas, l'agent superviseur doit mettre en œuvre à la fois les capacités d'ordonnancement et de négociation, mais avec un protocole de négociation et un algorithme de réordonnancement différents de ceux utilisés dans les étapes précédentes (réordonnancements locaux). Il essaye d'allouer les tâches de type prédécesseur en vue de respecter leurs échéances de précédence.

Dans le premier cas, il suit les même étapes que l'agent-central en annonçant les tâches non réalisables de son plan au lieu des nouvelles tâches. Il réitère ces étapes jusqu'à ce que toutes les tâches non réalisables soient attribuées aux autres agent-robots.

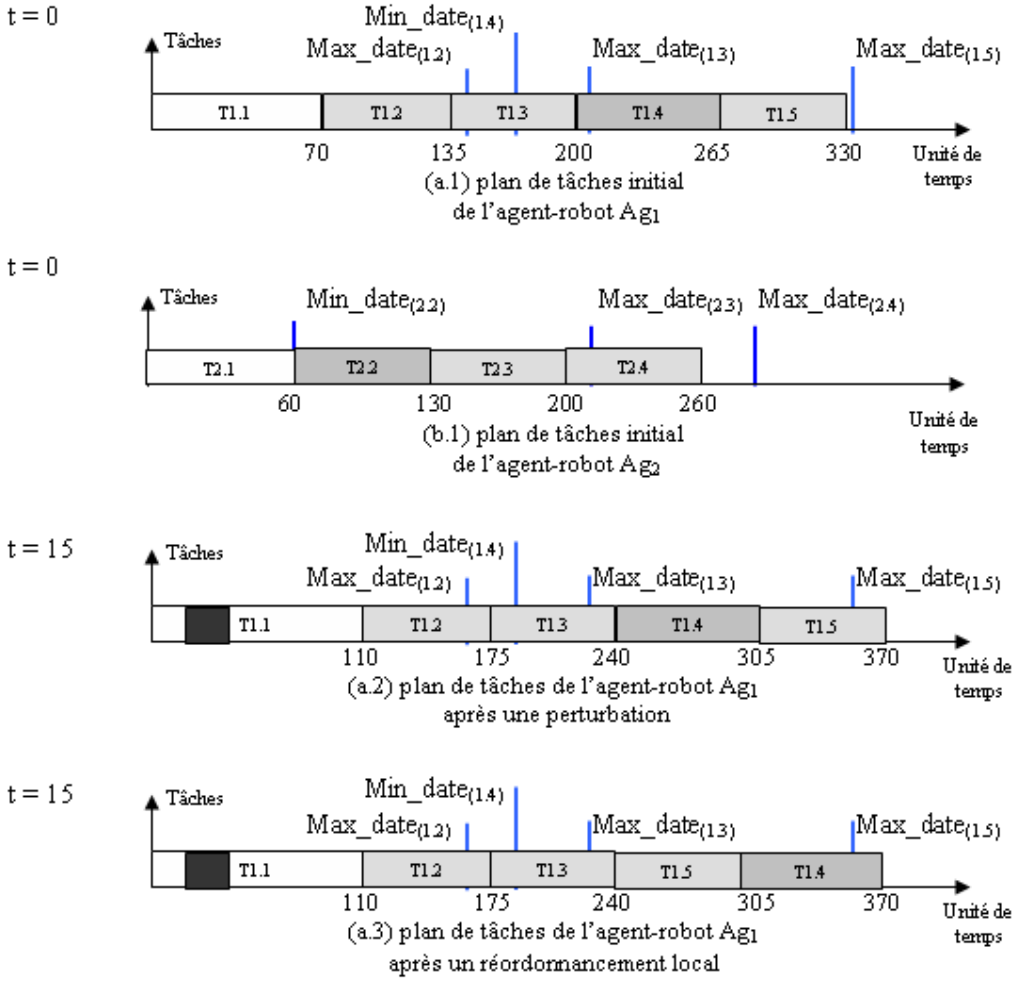
Dans le deuxième cas, voici les étapes à suivre :

- L'agent perturbé initie une enchère pour cette tâche en diffusant un message de négociation par allocation de tâches pour une tâche extraite de son plan de tâches, elle est de type prédécesseur et dépassant sa Max_date. Le message contient les informations nécessaires de la tâche (description, échéance) et la proposition de l'agent-robot manager, mais sans informations sur les contraintes de la tâche. Le

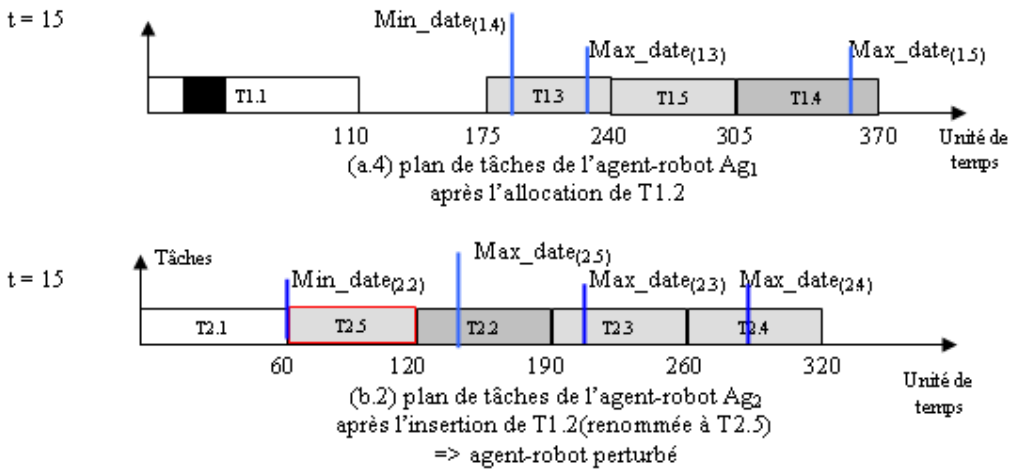
format d'un message d'appel d'offre est similaire au message de requête utilisé dans le cas de l'agent-central sauf le champ *"content"*, dans lequel il doit mentionner sa proposition.

- Chaque agent-robot sollicité doit élaborer sa propre proposition et l'envoie à l'agent-robot manager si la proposition est meilleure de ce dernier. Les agent-robots qui ont des propositions moins d'être meilleures que de celle reçue en appel d'offre, ne sont pas obligés de répondre. De ce fait, on a pu minimiser encore les messages échangés.
- À la fin de cette enchère, l'agent-robot perturbé attribue la tâche à l'agent-robot gagnant, dans ce cas, il envoie un seul message *"accept_proposal"* (s'il accepte une proposition) contenant des informations supplémentaires sur les contraintes de la tâche allouée. Il n'envoie pas de messages *"reject_proposal"*, un agent-robot coopérant considère implicitement que sa proposition est refusée s'il ne reçoit pas un message *"accept"* aprés un intervalle de temps donné ou à la réception d'une nouvelle demande de négociation.
- L'agent-robot gagnant confirme l'acceptation de l'allocation (il peut donc ère un nouvel agent-robot perturbé).
- L'agent-robot perturbé supprime la tâche allouée de son plan et le réordonne.
- L'agent-robot perturbé réitère les étapes de 1 à 6 jusqu'à ce que toutes les contraintes de précédences soient respectées ou il est toujours le gagnant (il n'y a aucune proposition).
- Si toutes les contraintes de précédences sont respectées, le traitement doit être interrompu et l'agent-robot revient en un état normal.
- La coopération va être implicitement considérée comme terminée après un intervalle de temps donné ou à la réception d'une nouvelle demande de négociation.

La Figure (Fig.4.17) présente un exemple illustrant l'utilisation de l'algorithme d'allocation de tâches dans un processus de réordonnancement de tâches d'un agent-robot perturbé Ag_1. Cet agent a subit une perturbation de type retard, tente de l'absorber d'abord par le réordonnancement local, suite à l'échec de ce dernier, il a fait une tentative de réordonnancement local avec négociation qui conduit de même à l'élaboration d'un nouveau plan qui ne satisfait pas toutes les contraintes de précédence, il doit donc tenter à absorber la perturbation par l'allocation de tâches. Nous ne montrons que l'évolution des plans des deux agent-robot Ag_1 et Ag_2 en considérant que le plan d'un Ag_3 est inchangé et sa participation dans la négociation n'a aucun effet



les réordonnancements locaux ne peuvent pas absorber la perturbation sur le plan de l'Ag_1 => il doit lancer un processus d'allocation de tâches



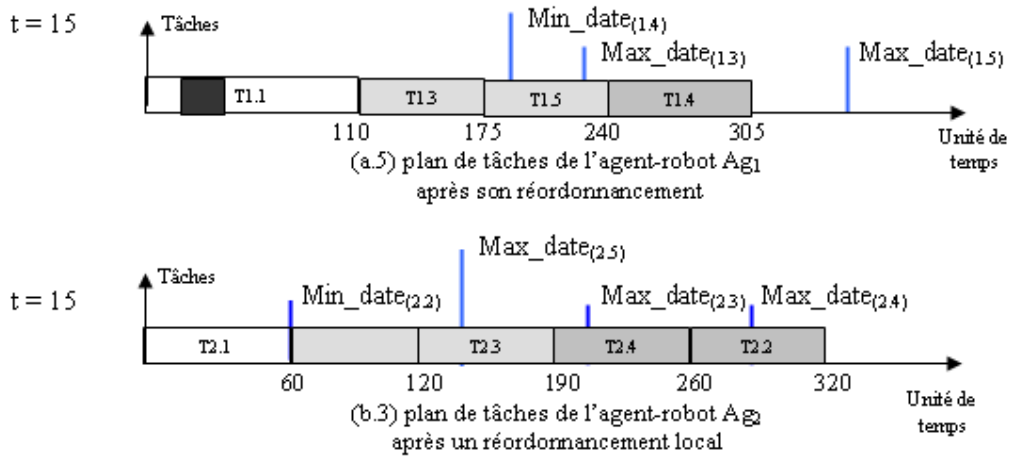


FIG. 4.17 – Exemple de réordonnancement global par allocation de tâches

Dans l'exemple présenté ci-haut, suite à une perturbation de type retard se produisant après 15 ut du lancement de la réalisation de $T_{1.1}$ de l'agent-robot Ag_1 (cf. Fig.4.17.a.2), cette perturbation s'est propagé sur toutes les tâches situées après $T_{1.1}$ en retardant leurs dates de débuts d'exécution par 40 ut. Les tâches $T_{1.2}, T_{1.3}$ et $T_{1.5}$ dépassent leurs échéances de précédence. L'Ag_1 a essayé d'absorber ces effets en exécutant l'algorithme de réordonnancement local, ce dernier a produit un nouveau plan (cf. Fig.4.17.a.2) dont la tâche $T_{1.5}$ peut respecter son échéance, alors que $T_{1.2}$ et $T_{1.3}$ dépassent toujours leurs échéances. Malgré l'exécution de l'algorithme de réordonnancement local avec négociation sur le plan obtenu par le réordonnancement local, aucune modification utile n'a été apportée sur le plan. À ce moment là, l'Ag_1 décide d'appeler le service d'allocation de tâches de l'agent comportemental stratégique de groupe. Le processus d'allocation consiste à faire une enchère pour la tâche $T_{1.2}$ en diffusant le message d'appel d'offre suivant à tous les agent-robots de l'organisation (dans ce cas, on a Ag_3, Ag_2 et Ag_1) et attendre toutes propositions :

cfp
$T_{1.2}$ R_1=370 Td_1=2
Ag_1-allocation-Ag_2-6
L1
O1
Task_allocation
Ag_2
now
Ag_1

FIG. 4.18 – Exemple de message d'appel d'offre

Á la réception du message envoyé par l'Ag_1. L'Ag_2 et l'Ag_3 estiment les effets de l'insertion de la tâche $T_{1.2}$ sur leurs plans initiaux (cf. Fig.4.17.b.1, pour l'Ag_2, l'Ag_2 trouve que l'insertion de $T_{1.2}$ dans son plan va retarder l'accomplissement de sa mission individuel vers 320 (Re_2) et deux tâches vont dépasser leurs échéances de précédence ($Card(Td_2)$) (cf. Fig.4.17.b.2), alors que la proposition de l'Ag_3 n'est pas meilleure par rapport à celle reçue de l'Ag_1 . par conséquent, l'Ag_2 répond par le message suivant :

propose
Re_2=320 Td_2=2
Ag_1-allocation-Ag_2-6
L1
O1
Task_allocation
Ag_1
Ag_2

FIG. 4.19 – Exemple de message de proposition

Après la réception des propositions, l'Ag_1 compare entre celles-ci et choisit la meilleure, dans cet exemple, il y a réception d'une seule proposition de Ag_2, dans laquelle on a : Re_2 =320 et $Card(Td_2)$ =2. Par conséquent, l'Ag_1 estime que la réalisation de $T_{1..2}$ soit améliorée si elle serait fait par l'Ag_2, il décide alors de lui déléguer sa réalisation. L'Ag_1 doit donc informer l'Ag_2 par le message suivant :

Accept_proposal
Succ=$T_{3,3}$
Ag_1-allocation-Ag_2-6
L1
O1
Task_allocation
Ag_2
now
Ag_1

FIG. 4.20 – Exemple de message d'acceptation de proposition

En fin, l'Ag_2 confirme l'acceptation de l'allocation par un message de confirmation simple. Par conséquent, l'Ag_1 supprime $T_{1.2}$ de son plan (cf. Fig.4.17.a.5) et l'Ag_2 l'insère (cf. Fig.4.17.b.2).

Suite à cette insertion, l'Ag_1 a appliqué un décalage à gauche sur son nouveau plan (il peut être considéré comme un réordonnancement local), ce qui a donné un nouveau plan valide qui respecte toutes les contraintes de précédence. Alors le processus de réordonnancement est terminé par rapport à l'Ag_1 qui rentre dans un état normal. Par contre, l'Ag_2 se trouve dans une situation d'exception à cause des dépassements des échéances de précédence (provoqués par l'insertion d'une nouvelle tâches) des tâches $T_{2.3}$ et $T_{2.4}$, il doit donc lancer un processus de réordonnancement. Heureusement, dans ce cas il se limite à un réordonnancement local. ce dernier a donné un plan valide (cf. Fig.4.17.b.2) et ramené l'Ag_2 en un état normal. Il consiste à déplacer $T_{2.2}$ vers la fin du plan et profiter de sa place pour décaler $T_{2.3}$ et $T_{2.4}$ vers gauche.

Bref les trois agent-robots se trouvent finalement dans des situations normales grâce à la résolution coopérative du problème d'ordonnancement.

Pour l'allocation d'une nouvelle tâche par l'agent-central, la différence est que ce dernier qui joue le rôle de l'enrichissement ne peut faire aucune proposition et doit donc allouer la tâche quelques soient les propositions, à condition qu'elles respectent certaines contraintes (par exemple, Re_i et/ou Td_i limité, etc.).

4.7.4 Algorithme de réordonnancement global par propagation de contraintes

Parmi les évènements perturbateurs, ceux qui peuvent conduire un agent-robot à des situations dans lesquelles la mise en œuvre des trois stratégies précédentes ne peut pas le ramener vers aucun état normal où toutes les contraintes de précédence sont respectées (ne peut produire aucun plan valide). Dans ce cas, il y a un risque de perdre la cohérence globale de la mission. Par conséquent, il peut ruiner l'organisation de robots où la mission se trouvera probablement bloquée. Pour contourner ce problème et donner aux agent-robot la capacité cognitive leur permettant de s'en sortir, c'est à dire d'élaborer un plan globalement cohérent, un agent-robot Ag_i doit être doté d'un quatrième mécanisme, soit : la propagation de contraintes. Pour ce faire, l'agent-robot perturbé doit suivre les étapes suivantes :

- Il envoie une demande de négociation par partage d'information à tous les agent-robots dont les plans de tâches sont couplés. Les messages envoyés ne concernent que les tâches de type prédécesseur non exécutées dont les contraintes de précédence sont dépassées, c'est à dire un message destiné à un $(Ag_j)_{j \neq i}$ doit contenir toutes les nouvelles dates de fin d'exécution des tâches de types prédécesseur dans l'Ag_i dont les tâches de type successeur sont situées dans le plan de l'Ag_j, ce dernier va mettre à jour ses connaissances locales en retardant l'exécution de ses tâches de type successeur associées, ce qui peut le rendre la cible d'une perturbation.
- Une fois tous les agent-robots concertés sont informés, l'ancien agent-robot perturbé Ag_i va sortir de l'état d'exception courant vers un autre état normal en mettant à jour ses connaissances (ajuster les dates des tâches perturbées).
- La coopération va être implicitement considérée comme terminée après un intervalle de temps donné ou à la réception d'une nouvelle demande de négociation.

Normalement, la mission globale va être accomplie dans les bonnes conditions malgré les événements perturbateurs se produisant durant sa réalisation.

La Figure (Fig.4.21 illustre l'utilisation de ce mécanisme pour réadapter les plans initiaux de tâches des agent-robots Ag_1 et Ag_2.

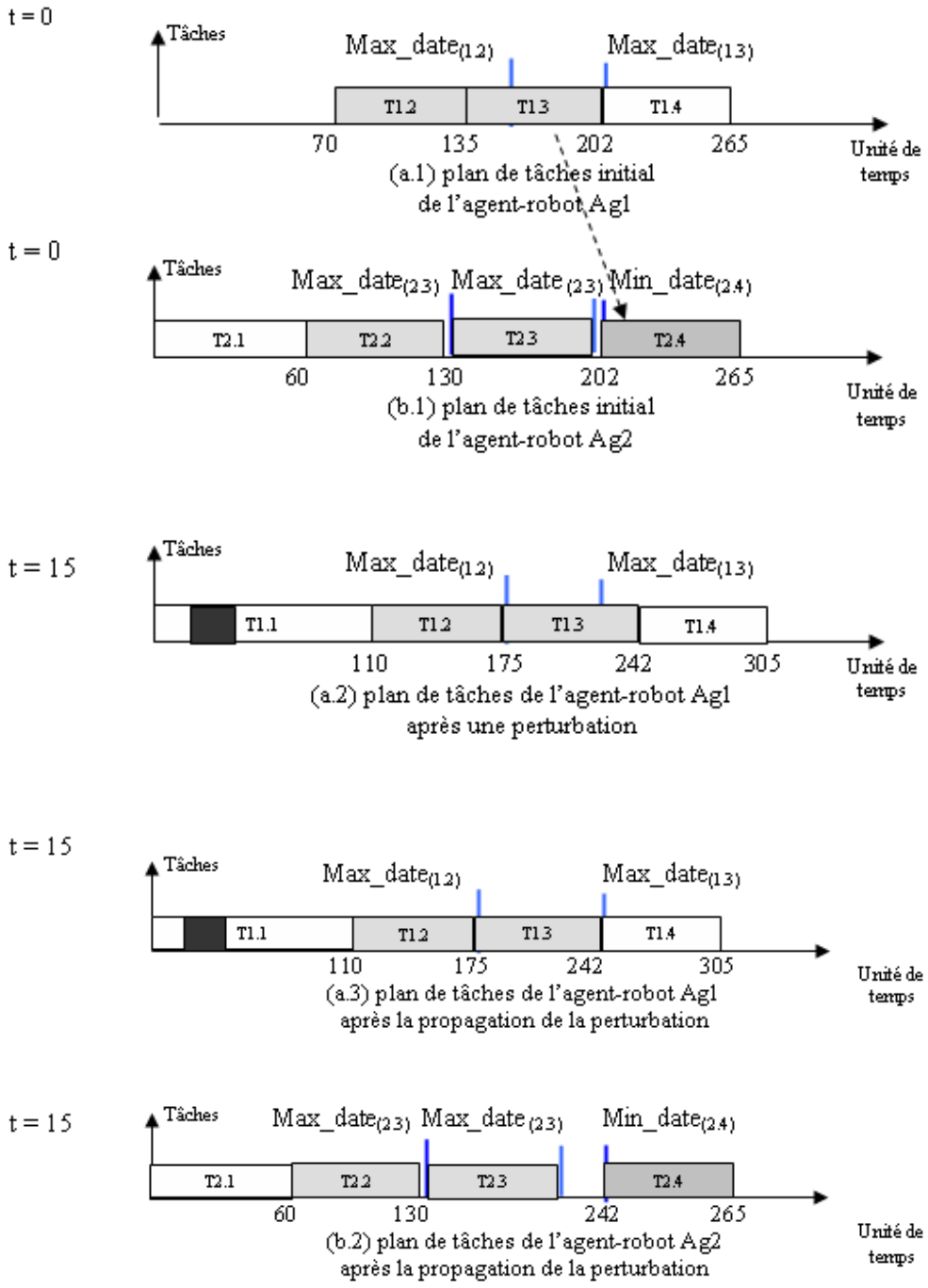


FIG. 4.21 – Exemple de réordonnancement global par propagation de contraintes

4.8 Respect de la cohérence globale

Un des problèmes liés à la nature distribuée de l'approche proposée est la synchronisation de réalisation des tâches. Les tâches à réaliser par des robots différents sont liées

par des contraintes de précédence. La décision pour démarrer la réalisation d'une tâche est prise en fonction de la vue de chaque participant. La conformité d'une décision par rapport aux contraintes imposées ne peut être assurée que si les paramètres nécessaires sont disponibles à l'instant et au robot appropriés. Pratiquement, pour que le respect de ces contraintes soit garanti, la solution retenue est de communiquer l'état des tâches de type prédécesseur aux différents robots qui ont les tâches de type successeur associées, c'est à dire que la réalisation d'une tâche de type successeur ne puisse se commencer que si le robot responsable de sa réalisation recevrait un message d'autorisation à partir du robot chargé de réaliser la tâche prédécesseur associée, c'est à dire ce message indique que cette dernière est terminée. Voici un exemple de ce type de message :

```
inform
T1,3 = 242
Ag1-propagation-Ag2-6
L1
O1
Constraint_propagation
Ag2
Ag1
```

FIG. 4.22 – Exemple de message d'autorisation de réalisation

4.9 Complexité

La qualité d'une solution *"S"*, produite par l'algorithmique d'ordonnancement proposée, est obtenue à l'aide d'une fonction d'évaluation mesurant sa complexité *C(s)* et son efficacité *P(s)*.

1. Nous mesurons l'efficacité d'une solution en utilisant deux paramètres
   - Le nombre global de contraintes non respectés (Fig.4.23.a).
   - Le temps moyen de réalisation (Fig.4.23.b).

$$\left(\sum_{j=1}^{n} Td_j\right)_{avec} - \left(\sum_{j=1}^{n} Td_j\right)_{sans} \quad \text{n : nombre de robots}$$

(a) nombre global de contraintes non respectés

$$\left(\frac{1}{N}\sum_{j=1}^{n} Re_j\right)_{avec} - \left(\frac{1}{N}\sum_{j=1}^{n} Re_j\right)_{sans}$$

(b) le retard moyen

FIG. 4.23 – Définition de l'efficacité d'une solution

2. La complexité est calculé par deux foncions :
   – Dans la première), nous nous basons sur 4 paramètres pour trouver la complexités calculatoire d'une solution d'ordonnancement. Pour les solutions obtenues en exécutant plus d'une stratégies, leur complexité est calculée en accumulant les complexités produites par chaque stratégie.

$$C1(s) = w_{s_1} \times \sum_{j=1}^{A_s} w_{a_1} \times nbr\ (a_j).$$

s : solution d'ordonnancement.
w_{sj} : poids de la stratégie s_j.
w_{aj} : poids de l'opération de réparation a_i.

FIG. 4.24 – Définition de la complexité d'une solution

   – Dans la deuxième, nous nous intéresserons au nombre de messages échangés durant un processus de réordonnancement. Pour ce faire, il faut calculer la somme des nombres de messages échangés dans chacune des trois dernières stratégies de réordonnancement

Réordonnancement local avec négociation Nombre de message = Somme (2 * le nombre d'agents liés à l'agent perturbé dont les contraintes de précédence de liaison sont dépassées) : pour chaque agent lié avec l'agent perturbé par des contrainte de précédence non respectée, l'agent perturbé doit lui adresser un message de requête concernant ces dernières et recevoir un message de réponse de chaque agent contacté. Puisque, on réitère ce processus jusqu'à ce qu'il n'y a

aucune amélioration d'ordonnancement, il faut sommer le nombre de messages échangés dans chaque tour de négociation.

Réordonnancement global par allocation de tâches On doit traiter les deux cas suivants :

1. Nombre de message échangés par l'agent-central = (2 * n) + L : n est le nombre d'agents, l'agent-central envoie un message d'appel d'offre (cfp) et recevoir un message de proposition pour tous les agents de l'organisation, L est le nombre de tentatives pour trouver un agent qui confirme l'acceptation de la tâche à allouée, d'où, L = 1..n+1 (on compte aussi le message de confirmation).
2. Nombre de message échangés par un agent-robot (perturbé) = Somme(n-1 + K + L) : l'agent-robot perturbé est exclu, K est le nombre d'agents qui peuvent apporter une amélioration de réalisation de la tâche en question, c'est à dire les agents qui peuvent proposer mieux (les autre ne sont pas obligés de répondre), L = 1..n+1 (en plus du message de confirmation, on a un message pour transférer les informations des contraintes sur la tâche allouée).

   Puisque, on réitère ce processus autant qu'il y a des tâches à allouer, il faut sommer le nombre de messages échangés pendant chaque processus d'allocation

Réordonnancement global par propagation de perturbation Nombre de messages échangés = 2 * le nombre d'agents liés à l'agent perturbé dont les contraintes de précédence qui leur relient sont dépassées

4.10 Conclusion

Nous avons présenté dans ce chapitre, le modèle de coopération proposé pour le réordonnancement et l'allocation dynamique de tâches dans le contexte d'un système multi-robots.

Nous avons ainsi développé une méthodologie d'ordonnancement et d'allocation de tâches aux robots selon un principe basé sur une analyse des contraintes définies dans la description de la mission. L'heuristique proposée minimise le nombre des contraintes de précédence non respectées entre les tâches gérées par des agent-robots différents, ainsi qu'elle minimise le retard induit par des événements perturbateurs dans les plans de tâches des robots. Pour ce faire, les tâches sont ordonnancées selon des priorités calculées au fur et à mesure de l'avancement d'exécution de la mission. Le calcul tient compte les échéances de précédence des tâches prédécesseurs ou non. Nous avons expliqué le comportement d'un agent-robot perturbé par des exemples simples qui ont montré l'utilisation de chacune des stratégies de réordonnancement proposées. Puis,

nous avons montré comment l'approche proposées peut garantir le respect de la cohérence globale de l'exécution d'une mission en utilisant la technique de notification pour les tâches liées. Nous avons terminé par l'étude de la complexité des solutions obtenues par cette approche, notamment les coûts calculatoires et communicationnels des algorithmes associés aux agents du système.

Chapitre 5

Implémentation et validation

5.1 Introduction

L'objet de ce chapitre est d'une part, de présenter l'implémentation de l'approche multi-agents proposée pour le contrôle d'organisation multi-robots, dans une application logicielle et d'autre part, d'étudier les performances du modèle de coopération proposé.

Dans un premier temps, nous décrivons les caractéristiques de l'application de simulation d'agents, que nous avons développé pour la validation de l'approche proposée. Nous présentons la plate forme sur laquelle nous avons basé et expliquons en particulier les concepts utilisés pour la mise en œuvre des mécanismes de concurrence et de communication.

Dans un deuxième temps, nous étudions les résultats de l'implémentation des différents algorithmes d'ordonnancement et d'exécution des missions, à partir d'un ensemble de scénarii générés aléatoirement. Chaque scénario est caractérisé par un nombre de tâches, d'agent-robots, de nouvelles tâches et de contraintes de précédence. Les résultats obtenus sont statistiquement évalués et comparés aux solutions obtenues par d'autre algorithmes.

5.2 Présentation de l'application de simulation d'agents

L'implémentation et la validation des modèles décrits dans les chapitres III et IV peut s'effectuer suivant deux approches. La première est l'implémentation physique du modèle logique de l'approche sur des composantes physiques de l'application. Ces

tests, bien que concrets, nécessitent des moyens (plusieurs robots et objets physiques) et un temps de mise en œuvre considérables. Pour pallier à ces problèmes, une autre alternative consiste à développer un simulateur pour la mise en œuvre et la validation des modèles proposés. Cette approche permet d'étudier et d'observer le comportement de chaque robot ainsi que l'évolution de l'organisation en temps réel. Il existe plusieurs plates-formes multi-agents, telles que :

- *ZEUS* [44] : est une plate-forme multi-agents conçue et réalisée par British Telecom, pour développer des applications collaboratives, fondée sur la notion de rôle. ZEUS est écrite en Java et repose sur le standard FIPA.
- *SWARM* [14] est une plate-forme multi-agents à base d'agents réactifs. Le modèle d'agent utilisé est inspiré par la vie artificielle. SWARM est l'outil privilégié de la communauté américaine et des chercheurs en vue artificielle Elle est implémentée en plusieurs langages (Java, Objective-C).
- *MADKIT* [25] est une plate-forme développée par le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) de l'Université Montpellier II. Elle est écrite en Java et est fondé sur le modèle organisationnel Alaadin, où les rôles des agents représentent leurs capacités à effectuer un service.

Les plates-formes disponibles offrent des services variés, comme par exemple des gestionnaires de messages, des moteurs de coordination, des outils de planification de tâches et des bases de données. La communication s'effectue en général par des messages dont l'ontologie est normalisée selon des standards comme FIPA [4].

La plate-forme multi-agents la plus utilisée est probablement JADE (Java Agent Development Framework) [11], développée en Java par CSELT (Groupe de recherche de Gruppo Telecom, Italie). Elle permet la mise en œuvre d'applications conformes à la norme FIPA. Elle utilise FIPA ACL comme langue de représentation des messages échangés. Chaque agent est identifié par une adresse globalement unique appelée AID qui contient le nom de l'agent, le nom et le port de l'hôte sur lequel l'agent écoute. Les agents doivent implémenter des tâches appelées Comportements ("Behaviors" en anglais) en JADE, elles sont ordonnancées par un algorithme Round-robin simple préemptif. Dans certains cas, lorsqu'un comportement est bloqué, il peut bloquer tout un agent jusqu'à l'occurrence de l'événement qui débloque le comportement (un comportement peut être bloqué jusqu'à la réception d'un message, il peut par conséquent bloquer le robot).

Il y a plusieurs modèles de comportement à choisir, les différence sont par exemple la manière qu'ils se terminent (jamais, après une ou plusieurs exécution, etc).

JADE comprend deux composants de base : une plate-forme d'agents et des *APIs* pour le développement des agents en Java.

Pour implémenter l'approche proposée dans ce mémoire, nous avons fait le choix de développer une application permettant la simulation de l'exécution d'un ensemble de scénarii générés aléatoirement, d'observer et d'étudier le comportement du modèle de contrôle proposé.

Le logiciel de simulation comporte trois modules indépendants (cf. Fig.5.1). Le premier est le "Générateur de scénarii". Il s'agit d'un logiciel non interactif développé en Java, invoqué à l'aide d'une ligne de commande avec des paramètres spécifiant les conditions générales pour la génération aléatoire d'un scénario de test. Ce scénario est enregistré dans un fichier XML qui est lu par le deuxième module : le module "Exécution", permettant de simuler l'évolution d'un ensemble d'agents autonomes. C'est dans ce module que nous avons implémenté nos algorithmes d'ordonnancement distribué et d'allocation de tâches. Les résultats de l'exécution sont enregistrés sous forme de texte dans des fichiers journaux. Le dernier module, que nous appelons "Solveur simple", fait appel au propagateur de contraintes permettant de garantir la cohérence globale. Il fournit une solution simple en propageant directement la perturbation aux autres agents sans tenter de l'absorber. Enfin, nous avons utilisé des outils statistiques pour l'analyse des résultats obtenus.

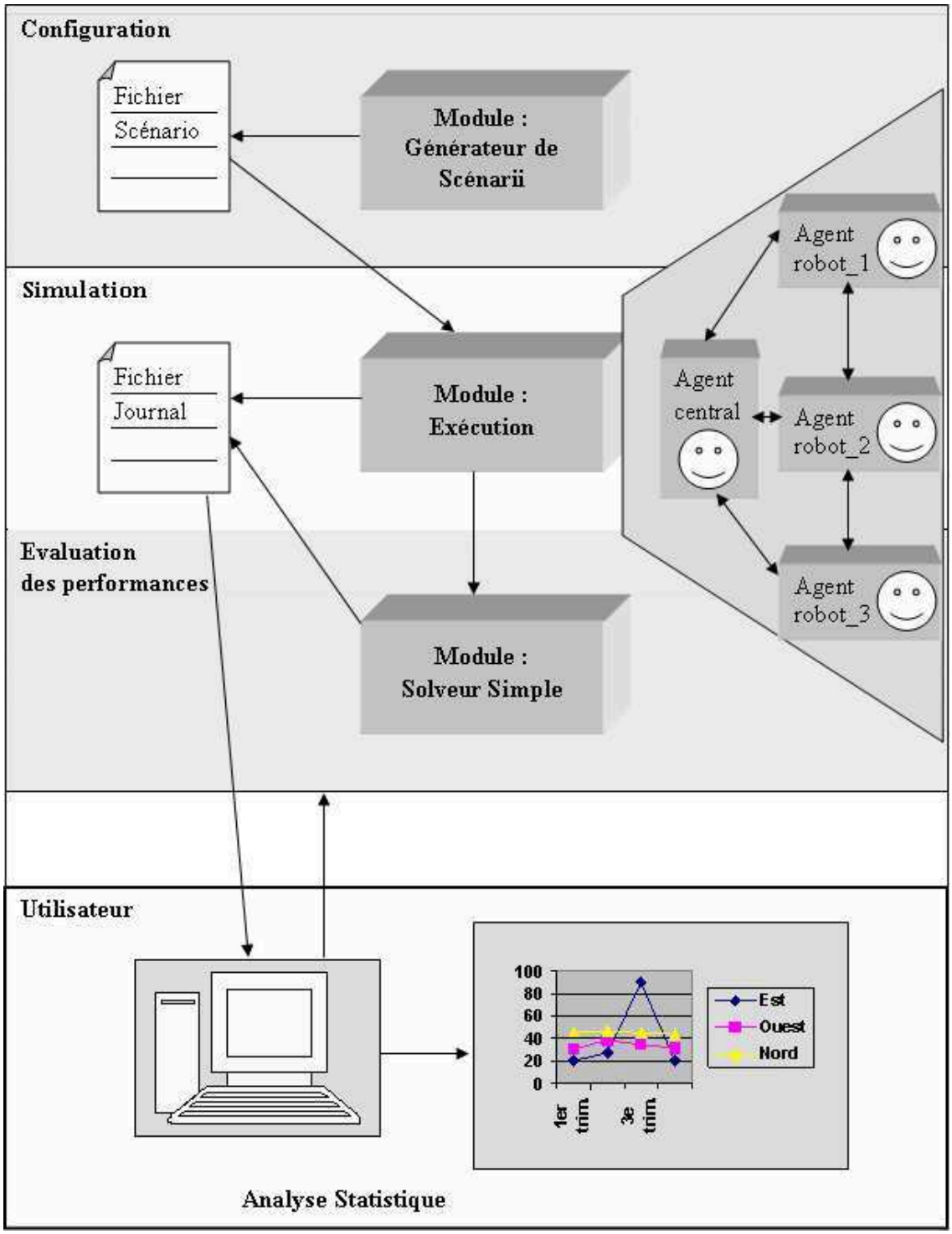


FIG. 5.1 – Modules du logiciel de simulation

5.2.1 Module Générateur de Scénarii

Pour la validation de l'approche proposée, un générateur de scénarii aléatoires a été mis en œuvre. Chaque scénario est caractérisé par :

- Le nombre d'agent-robots.
- Le nombre moyen de tâches à exécuter par chaque agent.
- La durée moyenne des tâches.
- La probabilité que deux tâches soient liées par des contraintes de précédence.
- La probabilité qu'une nouvelle tâche soit présente à chaque laps de temps.

Les trois premiers paramètres sont fixé par l'utilisateur. Alors que pour déterminer l'existence ou non d'une contrainte de précédence entre deux tâches, nous proposons l'algorithme suivant :

1. Prendre une paire de tâches T_i ; T_j. i,j $\in$ [1..nombre de tâches]. i $\neq$ j.
2. Si Début_exec(T_i) + Temps_exec(T_i) $\leq$ Début_exec(T_j), générer une valeur aléatoire équidistribuée entre 0 et 1.
3. Si la valeur générée est plus petite que la probabilité fixée, introduire une contrainte de précédence entre T_i et T_j.

De même, pour déterminer l'existence ou non d'une nouvelle tâche à un instant donné, nous proposons l'algorithme suivant

1. Á la fin de chaque laps de temps, générer une valeur aléatoire équidistribuée entre 0 et 1.
2. Si la valeur générée est plus petite que la probabilité fixée, programmer une nouvelle tâche à cet instant.

Le scénario ainsi généré est enregistré dans un fichier au format XML, où figure :

- Le nom de chaque robot.
- Les noms des tâches de chaque plan, la date de début d'exécution, la durée d'exécution et l'échéance de précédence. Ainsi que les noms des prédécesseurs et successeurs (tâche et agent) qui leurs sont associés.
- Le nom, la date de début d'exécution et l'échéance de toutes les nouvelles tâches.

5.2.2 Module Exécution

Le module "Exécution" est une application qui simule les activités et les interactions d'un ensemble d'agents logiciels. Un agent est implémenté sous forme d'une classe Java étend la classe "Agent" (définit par JADE) et redéfinit des méthodes du package "jade.core.behaviours" telles que : CyclicBehaviour, TickerBehaviour, SimpleBehaviour.

5.2.3 Module Solveur Simple

Pour quantifier les performances de l'approche proposée, nous comparons les résultats obtenus avec la solution qui cherche à garantir la cohérente globale. Cette dernière est obtenue en propageant la perturbation sur les agents auxquels l'agent perturbés est lié si la perturbation affectera les tâches de liaison. C'est la dernière étape dans la méthode d'ordonnancement proposée

5.3 Evaluation des performances

Dans cette partie, nous étudions les performances des algorithmes d'ordonnancement et d'allocation dynamique de tâches, à travers un ensemble de scénarii générés aléatoirement. Pour chaque scénario, les résultats obtenus sont enregistrés puis analysés en vue de déterminer leur qualité par rapport aux critères suivants :

- Le comportement vis-à-vis des problèmes d'échelle.
- La minimisation du nombre de messages échangés et le nombre d'opération de réparation effectuées pour accomplir une mission.
- L'obtention d'un ordonnancement minimisant le temps de réalisation de la mission et le nombre de contraintes non respectés.

Ainsi, à l'aide du module "Générateur de scénarii", nous avons généré aléatoirement plus de 200 scénarii différents. Nous avons varier le nombre d'agent-robots entre 2 et 10, le nombre de contrainte de précédence entre 10 et 60, le nombre des nouvelles tâches entre 1 et 20. Le nombre moyen de tâches à exécuter a été fixé à 12 et 16 tâches par robot.

5.3.1 Nombre de message

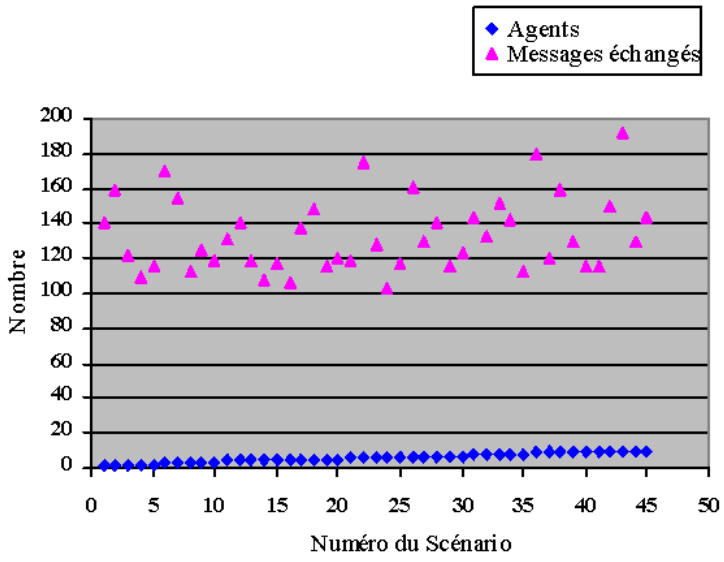
La Figure (Fig.5.2) illustre le rapport entre les paramètres des scénarii et le nombre de messages échangés pendant l'exécution des scénarii. À partir des résultats obtenus (Figure V.2.b et Figure V.2.c), nous pouvons remarquer que le nombre de messages échangés évolue quasi-linéairement en fonction :

- Du nombre de contraintes de précédence entre tâches ($p \sim 0,9$: coefficient de corrélation), ce paramètre représente le degré de couplage entre les plans de tâches des agent-robots.
- Du nombre de nouvelles tâches ($p \sim 0,95$), qui représente le degré de la dynamique du système.

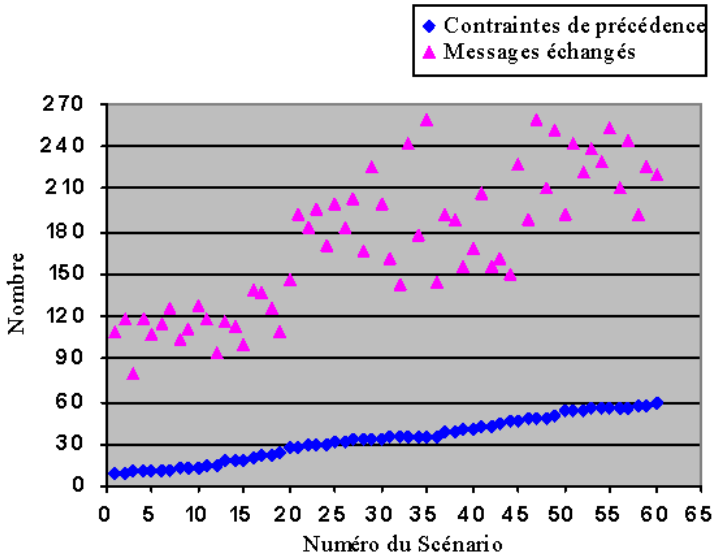
Le coefficient de détermination p^2 (le carré de la corrélation) est égal 0,81 et 0,9. Ce coefficient de détermination proche de 1 signifie que nous pouvons faire confiance à

cette dépendance linéaire entre ces deux paramètres et le nombre de messages échangés.

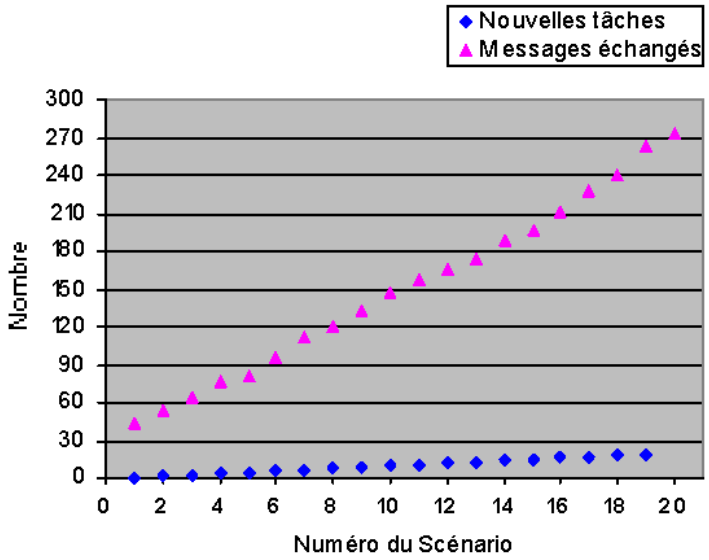
D'autre part, nous avons constaté que l'évolution du nombre de messages est indépendante du nombre d'agent-robots ($p \sim 0,2$) (Figure V.2.a). On peut conclure donc, que l'algorithmique proposée est bien adaptée au dimensionnement à grande échelle ("scalability" en anglais). En effet, l'agent-robot perturbé ne poursuit pas la négociation avec un autre agent-robot lorsqu'à un instant donné celui-ci ne se trouve plus affecté par la perturbation. En outre, un agent-robot perturbé ne sollicite la négociation qu'avec les agent-robots dont les plans sont couplés à son plan de tâches.



(a) Rapport entre le nombre d'agents et le nombre de messages



(b) Rapport entre le nombre de contraintes de précédence et le nombre de messages



(c) Rapport entre le nombre des nouvelles tâches et le nombre de messages

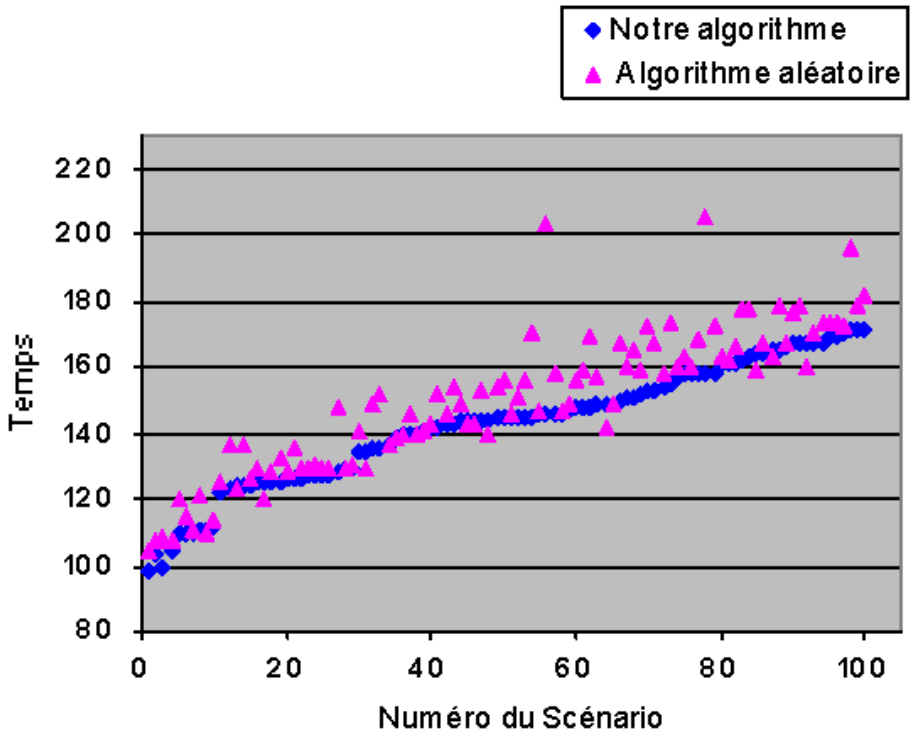
FIG. 5.2 – Rapport entre les paramètres des scénarii et le nombre de messages

5.3.2 Allocation de tâches

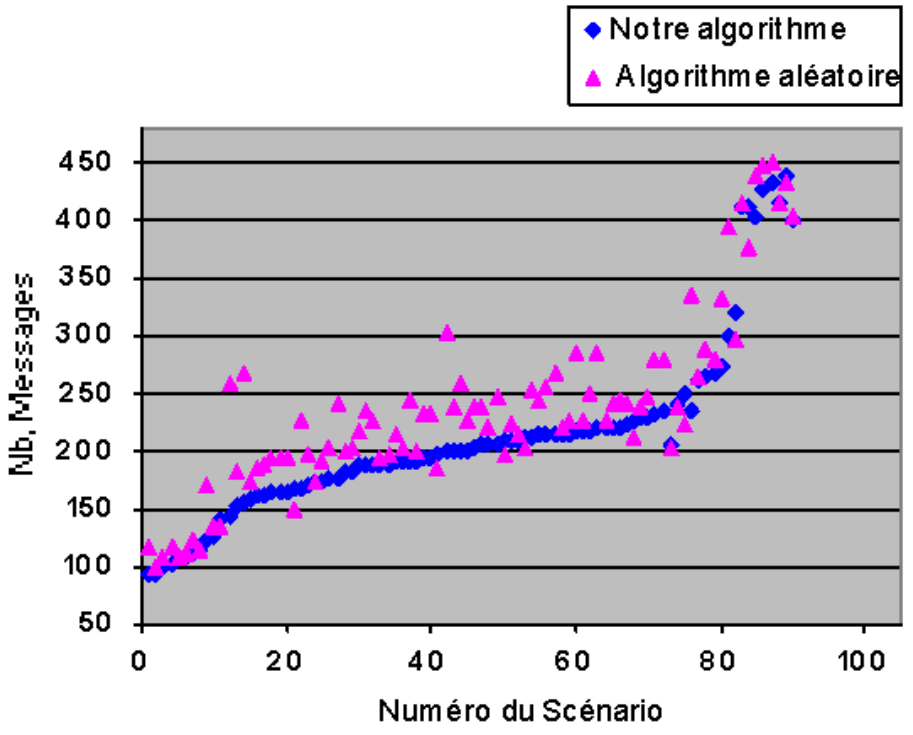
Nous analysons le comportement de l'algorithme d'allocation de tâches, décrit dans la *Section 6.3* du chapitre 4. Nous faisons une comparaison des performances de cet algorithme d'allocation par rapport à l'algorithme d'allocation aléatoire. Nous étudions notamment les rapports suivants :

- Le nombre de messages échangés pendant l'exécution d'un scénario.
- Le nombre d'opérations de réparation effectuées durant l'exécution d'un scénario.
- Le temps global d'exécution d'un scénario.

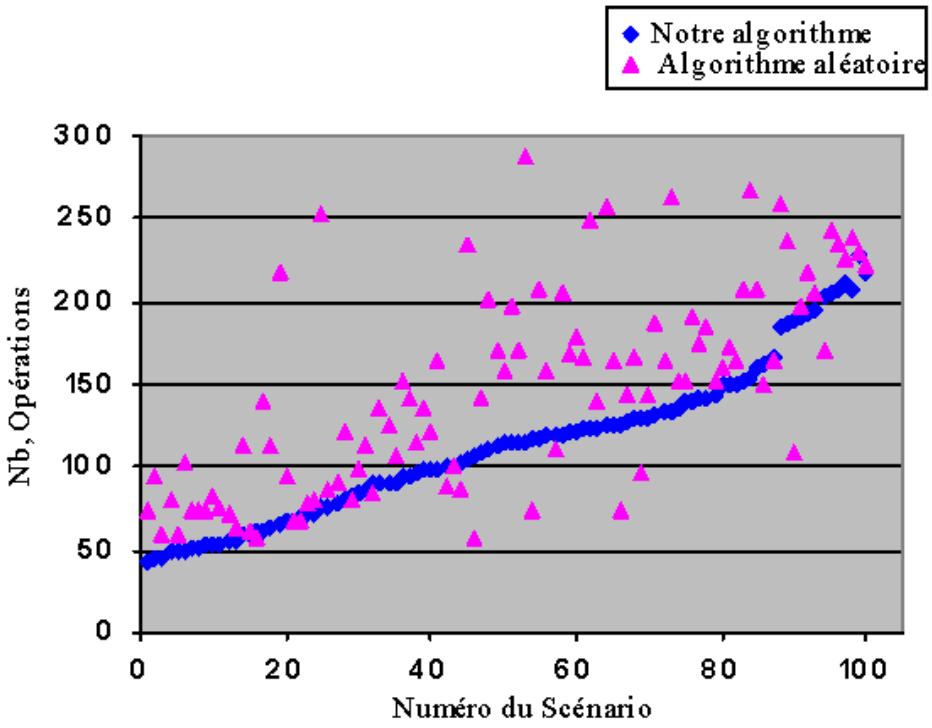
La Figure (Fig.5.3) illustre les résultats obtenus pour 100 scénarii créés aléatoirement et exécutés deux fois : dans la première exécution, nous avons mis en œuvre l'algorithme d'allocation proposé, et dans la deuxième, c'est l'allocation aléatoire qui est mise en œuvre. Puisque les agents n'ont qu'une vue partiale du système globalement dynamique, il est tout à fait normal que dans certains scénarii, l'allocation aléatoire peut conduire à un nombre de messages échangés plus petit que celui échangé par l'algorithme proposé, mais ce dernier est meilleur dans la plupart des scénarii.



(a) Temps de réalisation des scénarii



(b) Nombre de messages échangés



(c) Nombre d'opérations de réparation exécutées

FIG. 5.3 – Performance de l'algorithme d'allocation proposé

Dans les 100 scénarii, le nombre de tâches en-ligne (qui arrivent durant l'exécution du scénario) est fixé à 8 et 12 et le nombre moyen de tâches initiales est égal à 12 par robot. Les 60 premiers scénarii concernent 6 agents-robots avec un nombre de contraintes de précédence autour de 12, 34 et 53. Les scénarii restants concernent 10 agent-robots et un nombre de contraintes de précédence autour de 18, 34, 54 et 75.

Les résultats précédents montrent clairement que le nombre de messages échangés et le nombre d'opérations de réparation exécutées sont fortement liées au nombre de contraintes de précédence entre les tâches. Nous pouvons aussi bien observer les améliorations en nombre de messages échangés, en nombre d'opérations de réparation exécutées et en temps d'exécution qui apportent la mise en œuvre de notre algorithme d'allocation par rapport à celui de l'allocation aléatoire.

5.3.3 Réordonnancement de tâches

Pour mesurer les performances de la méthode d'ordonnancement proposée, nous avons réalisé une comparaison avec les ordonnancements qui se contentent à garantir

seulement l'exécution cohérente des scénarii, c.-à-d. respecter les contraintes de précédence. Un tel ordonnancement est l'ordonnancement obtenu par l'application du quatrième algorithme de la méthode proposé (la propagation de contraintes), qui permet le respect des contraintes de précédence entre tâches. Ainsi, nous avons exécuté 100 scénarii créés aléatoirement en mettant en œuvre les algorithmes d'ordonnancement proposés. Ensuite, les ordonnancements garantissant l'exécution cohérente des scénarii sont calculés hors-ligne. Nous avons étudié le rapport entre les temps d'exécution et les nombres de contraintes respectées dans les deux cas.

La Figure suivante montre les résultats obtenus pour les 100 scénarii avec les mêmes paramètres ci-haut. Les contraintes minimisées représentent la différence entre le nombre de contraintes d'échéance et de précédence dépassées avant et après le réordonnancement.

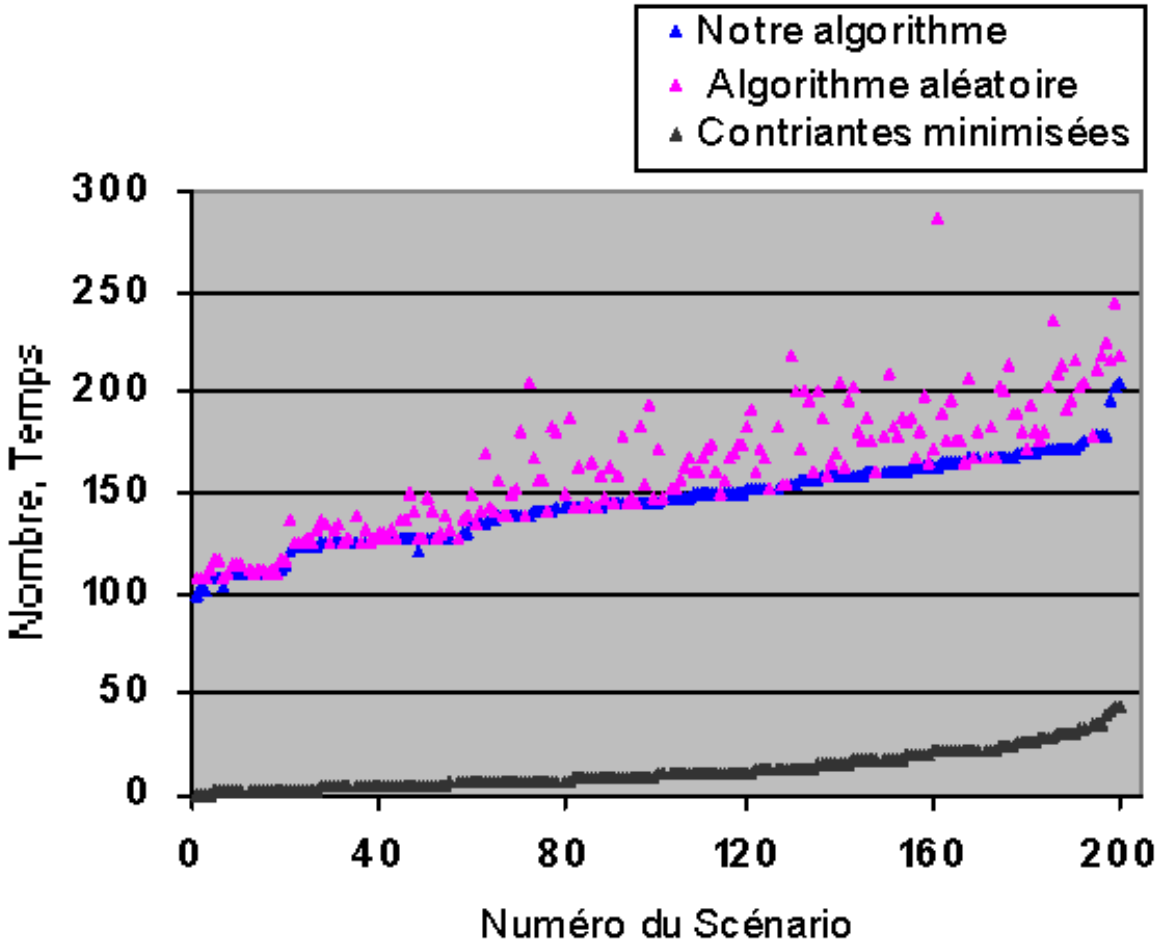


FIG. 5.4 – Performance de l'algorithmique d'ordonnancement proposée

Ces résultats montrent que les ordonnancements produits par la méthode d'ordonnancement proposée satisfaits bien nos objectifs initiaux (minimisation du temps global de réalisation et minimisation du nombre de contraintes dépassées).

En observant la figure V.4, nous pouvons remarquer facilement que le nombre de contraintes dépassées et le temps global de réalisation sont liés au nombre de contraintes de précédence.

Les résultats obtenus montrent que le temps de réponse pour l'établissement d'un nouvel ordonnancement est borné quelque soit la stratégie de résolution mise en œuvre dans la recherche de solutions. Pour les scénarii qui arrivent à mettre en œuvre toutes stratégies de résolution (de l'ordonnancement local, à la propagation de contraintes), le temps de réponse est inférieur à 1 second.

5.4 Conclusion

Dans ce chapitre, nous avons exposé l'implémentation et la validation des modèles de coopération et de l'algorithmique associée proposés dans ce mémoire. Pour ce faire, un environnement de test a été développé et mis en œuvre à l'aide d'une plate-forme multi-agents bien connue, appelée " JADE ", ce qui permettra d'aboutir à un système répondant aux normes courantes. Cet environnement nous a permet de simuler l'exécution distribuée d'une mission multi-robots dynamique à l'aide des agents autonomes.

Les résultats issus de l'exécution de nombreux scénarii que nous avons menées, ont montré que l'approche proposée pour le contrôle des système multi-robots par auto-ordonnancement (ou encore par auto-organisation) de tâches basée sur la coopération d'agents permet d'obtenir un comportement d'agent compatible avec la situation dans laquelle se trouve l'organisation, c'est à dire un comportement adéquat, adaptatif et performant en termes de temps de réalisation et d'agent-robots affectés, tout en étant de complexité limitée. Les performances temporelles obtenues expliquent l'intérêt de l'utilisation des méthodes heuristiques ainsi que la négociation inter-agents pour écourter le temps de recherche de solution. Les résultats obtenus permettent d'envisager d'étendre l'approche pour le traitement d'autres types de contraintes

Nos objectifs étaient de proposer un modèle d'auto-ordonnancement permettant l'accomplissement robuste d'une mission multi-robots, la minimisation du temps global de réalisation et la minimisation du nombre de contraintes non respectées. Ce chapitre nous a montré que les ordonnancements produits par l'algorithmique proposée satisfaits bien nos objectifs initiaux. En plus, cette algorithmique permet un équilibrage de la charge de réalisation d'une mission entre les robots de l'organisation, grâce à l'algorithme d'allocation de tâches.

Conclusion générale et Perspectives

AU terme de ce manuscrit, nous nous proposons de faire un récapitulatif de notre travail, d'analyser globalement les résultats obtenus et enfin de dresser des perspectives qui permettront d'une part, de finaliser les objectifs que nous nous étions fixés au début de ce mémoire et d'autre part, d'envisager de nouveaux axes de recherche comme une suite logique de ce travail.

Dans ce mémoire, nous avons étudié et développé une approche méthodologique et algorithmique pour l'ordonnancement distribué et dynamique de tâches soumises à des contraintes temporelles et de ressources, dans le contexte de la robotique distribuée.

L'approche proposée se base sur le paradigme multi-agents, mettant en œuvre un modèle de coopération pour l'ordonnancement dynamique de tâches, négociant en fonction des informations limitées dont les agents disposent, afin de s'adapter aux changement de l'environnement dans lequel ils évoluent, c'est à dire établir dynamiquement de nouveaux ordonnancements de tâches (plans de tâches) lors de l'exécution d'une mission. Ainsi, nous avons exposé un modèle générique d'organisation multi-agents. Ce modèle privilégie à la fois la distribution de la décision, l'autonomie et la réactivité des agents ainsi que la coordination décentralisée. Il s'agit d'une architecture hybride, s'appuyant sur des capacités délibératives pour raisonner dans des situations complexes et des capacités réactives pour respecter des échéances. Nous avons défini les comportements des agents par des plans comportementaux locaux et des protocoles de négociation qui ont pour objectif de réaliser un partage de connaissances ou une allocation de tâches entre les agents. La solution au problème d'ordonnancement consiste d'abord en un traitement local (au niveau de l'agent perturbé) puis en cas d'échec, passe par une résolution coordonnée qui met en œuvre des négociations entre agents, permettant de réaliser une coopération informationnelle et/ou une coopération par allocation de tâches. Deux critères sont pris en compte dans la recherche de solutions : le nombre de tâche de type prédécesseur affectées par une perturbation et le retard moyen induit au niveau de toute l'organisation. Par ailleurs, la recherche d'une solution coordonnée peut passer par la propagation des contraintes de l'agent perturbé à un autre afin d'assurer

la cohérence globale de réalisation d'une mission. Pour garantir une réactivité de l'organisation qui doit être compatible avec sa dynamique, nous avons proposé et développé un ensemble d'heuristiques associé à des mécanismes de négociation en vue de réduire le temps de recherche de solution.

Une bonne partie de ce travail de magistère a été consacrée à la validation des modèles proposés, pour laquelle une application de simulation multi-agents à été développée. L'analyse statistique des résultats de l'exécution d'un grand nombre de scénarii générés aléatoirement par un générateur aléatoire de scénarii développé, montre que des agents logiciels employant les algorithmes proposés sont capables d'ordonnancer dynamiquement et sans visibilité globale de l'état du système, des tâches soumises à des contraintes de précédence et d'échéance. En effet, l'algorithme d'allocation de tâches trouve effectivement des allocations équilibrant la charge de l'ensemble de robots.

Les résultats obtenus de l'exécution des scénarii sont dans la plupart des cas meilleurs si les agents adoptent les heuristiques proposées. Nous avons également étudié et validé expérimentalement la complexité algorithmique du modèle proposé. Cette complexité concerne les coûts calculatoires des algorithmes associés aux agents, ainsi que le nombre de messages échangés entre agents, dans les processus de négociation. Le nombre de messages échangés évolue quasi-linéairement en fonction du nombre de contraintes de précédence et du nombre de nouvelles tâche, et indépendamment du nombre d'agents.

Compte tenu des différents points que nous avons abordés au cours de ce mémoire, nous proposons dans un cadre de travaux futurs d'aborder d'autres points :

- Comparaison avec des solutions globalement optimales, obtenues par un Solveur optimal. La solution optimale, représente un ordonnancement de toutes les tâches, dont le temps d'exécution de bout en bout est minimal. Elle ne peut être établie que lorsque le système est entièrement observable, c'est-à-dire en traitant l'ensemble des tâches et leurs contraintes de manière centralisée.
- Il reste à valider expérimentalement la prise en compte d'autres contraintes : contraintes multiples de précédence entre tâches (la tâche d'un agent est liée à plusieurs tâche d'autres agents).
- Lever certaines hypothèses retenues dans le cadre de l'algorithmique présentée, par exemple la nécessité d'un seul robot à la fois pour réaliser une tâche. Dans un MRS, une tâche peut nécessiter plusieurs robots en même temps (transport d'objets lourds, etc).

Annexe A

A.1 Exemples de Scénario.xml

```
<?xml version="1.0"?> <!--
#######################################################################
        ###         Informations of the mission        ###
                    ==========================
# Creating configuration test data
# Start time: 13 11 2007 7:39:27 at Wahid
# # Parameters:
#   Number of robots 4
# Utilisation rate of robots        Number of tasks per robot
#   robot_1 : 0.25              ==>     12
#   robot_2 : 0.25              ==>     12
#   robot_3: 0.25               ==>     12
#   robot_4 : 0.25              ==>     12
# Number of tasks of the mission            48
# Execution time interval                   [16, 24]
# Probability of precedence intra plans     0.01
# ==> number of precedence intra plans      1
# Probability of precedence inter plans     0.2
#   ==> number of precedence inter plans    21
# # number of on-line tasks                 7
#######################################################################
### -->

<agent:missiondata>
  <agent:robot name="robot_1">
    <agent:task name="task_1" start="0" length="16" deadline="62">
      <agent:successor task_Name="task_4" robot_Name="robot_3"/>
    </agent:task>
    <agent:task name="task_2" start="17" length="22" deadline="43">
      <agent:successor task_Name="task_3" robot_Name="robot_2"/>
    </agent:task>
```

```
  <agent:task name="task_3" start="40" length="19" deadline="0">
    <agent:predecessor task_Name="task_1" robot_Name="robot_3"/>
  </agent:task>
  <agent:task name="task_4" start="60" length="23" deadline="106">
    <agent:successor task_Name="task_6" robot_Name="robot_2"/>
  </agent:task>
  <agent:task name="task_5" start="84" length="20" deadline="171">
    <agent:successor task_Name="task_9" robot_Name="robot_2"/>
  </agent:task>
  <agent:task name="task_6" start="105" length="21" deadline="190">
    <agent:successor task_Name="task_10" robot_Name="robot_3"/>
  </agent:task>
  <agent:task name="task_7" start="127" length="16" deadline="208">
    <agent:successor task_Name="task_11" robot_Name="robot_3"/>
  </agent:task>
  <agent:task name="task_8" start="144" length="22" deadline="206">
    <agent:successor task_Name="task_11" robot_Name="robot_4"/>
  </agent:task>
  <agent:task name="task_9" start="167" length="21" deadline="0">
    <agent:predecessor task_Name="task_2" robot_Name="robot_2"/>
  </agent:task>
  <agent:task name="task_10" start="189" length="23" deadline="227">
    <agent:successor task_Name="task_12" robot_Name="robot_2"/>
  </agent:task>
  <agent:task name="task_11" start="213" length="16" deadline="0">
    <agent:predecessor task_Name="task_1" robot_Name="robot_2"/>
  </agent:task>
  <agent:task name="task_12" start="230" length="16" deadline="0">
    <agent:predecessor task_Name="task_1" robot_Name="robot_4"/>
  </agent:task>
</agent:robot>
<agent:robot name="robot_2">
  <agent:task name="task_1" start="0" length="21" deadline="212">
    <agent:successor task_Name="task_11" robot_Name="robot_1"/>
  </agent:task>
  <agent:task name="task_2" start="22" length="21" deadline="166">
    <agent:successor task_Name="task_9" robot_Name="robot_1"/>
  </agent:task>
  <agent:task name="task_3" start="44" length="20" deadline="0">
    <agent:predecessor task_Name="task_2" robot_Name="robot_1"/>
  </agent:task>
  <agent:task name="task_4" start="65" length="17" deadline="148">
    <agent:successor task_Name="task_8" robot_Name="robot_2"/>
  </agent:task>
  <agent:task name="task_5" start="83" length="23" deadline="225">
    <agent:successor task_Name="task_12" robot_Name="robot_3"/>
  </agent:task>
```

```
  <agent:task name="task_6" start="107" length="23" deadline="0">
    <agent:predecessor task_Name="task_4" robot_Name="robot_1"/>
  </agent:task>
  <agent:task name="task_7" start="131" length="17" deadline="149">
    <agent:successor task_Name="task_8" robot_Name="robot_4"/>
  </agent:task>
  <agent:task name="task_8" start="149" length="22" deadline="0">
    <agent:predecessor task_Name="task_4" robot_Name="robot_2"/>
  </agent:task>
  <agent:task name="task_9" start="172" length="16" deadline="0">
    <agent:predecessor task_Name="task_5" robot_Name="robot_1"/>
  </agent:task>
  <agent:task name="task_10" start="189" length="19" deadline="0">
    <agent:predecessor task_Name="task_7" robot_Name="robot_4"/>
  </agent:task>
  <agent:task name="task_11" start="209" length="18" deadline="0">
    <agent:predecessor task_Name="task_2" robot_Name="robot_3"/>
  </agent:task>
  <agent:task name="task_12" start="228" length="23" deadline="0">
    <agent:predecessor task_Name="task_10" robot_Name="robot_1"/>
  </agent:task>
</agent:robot>
<agent:robot name="robot_3">
  <agent:task name="task_1" start="0" length="18" deadline="39">
    <agent:successor task_Name="task_3" robot_Name="robot_1"/>
  </agent:task>
  <agent:task name="task_2" start="19" length="22" deadline="208">
    <agent:successor task_Name="task_11" robot_Name="robot_2"/>
  </agent:task>
  <agent:task name="task_3" start="42" length="20" deadline="82">
    <agent:successor task_Name="task_5" robot_Name="robot_4"/>
  </agent:task>
  <agent:task name="task_4" start="63" length="19" deadline="0">
    <agent:predecessor task_Name="task_1" robot_Name="robot_1"/>
  </agent:task>
  <agent:task name="task_5" start="83" length="19" deadline="0">
  </agent:task>
  <agent:task name="task_6" start="103" length="22" deadline="167">
    <agent:successor task_Name="task_9" robot_Name="robot_4"/>
  </agent:task>
  <agent:task name="task_7" start="126" length="22" deadline="0">
    <agent:predecessor task_Name="task_4" robot_Name="robot_4"/>
  </agent:task>
  <agent:task name="task_8" start="149" length="22" deadline="188">
    <agent:successor task_Name="task_10" robot_Name="robot_4"/>
  </agent:task>
  <agent:task name="task_9" start="172" length="18" deadline="0">
```

```
    <agent:predecessor task_Name="task_3" robot_Name="robot_4"/>
  </agent:task>
  <agent:task name="task_10" start="191" length="17" deadline="0">
    <agent:predecessor task_Name="task_6" robot_Name="robot_1"/>
  </agent:task>
  <agent:task name="task_11" start="209" length="16" deadline="0">
    <agent:predecessor task_Name="task_7" robot_Name="robot_1"/>
  </agent:task>
  <agent:task name="task_12" start="226" length="17" deadline="0">
    <agent:predecessor task_Name="task_5" robot_Name="robot_2"/>
  </agent:task>
</agent:robot>
<agent:robot name="robot_4">
  <agent:task name="task_1" start="0" length="19" deadline="229">
    <agent:successor task_Name="task_12" robot_Name="robot_1"/>
  </agent:task>
  <agent:task name="task_2" start="20" length="21" deadline="0">
  </agent:task>
  <agent:task name="task_3" start="42" length="16" deadline="171">
    <agent:successor task_Name="task_9" robot_Name="robot_3"/>
  </agent:task>
  <agent:task name="task_4" start="59" length="23" deadline="125">
    <agent:successor task_Name="task_7" robot_Name="robot_3"/>
  </agent:task>
  <agent:task name="task_5" start="83" length="23" deadline="0">
    <agent:predecessor task_Name="task_3" robot_Name="robot_3"/>
  </agent:task>
  <agent:task name="task_6" start="107" length="21" deadline="0">
  </agent:task>
  <agent:task name="task_7" start="129" length="20" deadline="188">
    <agent:successor task_Name="task_10" robot_Name="robot_2"/>
  </agent:task>
  <agent:task name="task_8" start="150" length="17" deadline="0">
    <agent:predecessor task_Name="task_7" robot_Name="robot_2"/>
  </agent:task>
  <agent:task name="task_9" start="168" length="20" deadline="0">
    <agent:predecessor task_Name="task_6" robot_Name="robot_3"/>
  </agent:task>
  <agent:task name="task_10" start="189" length="17" deadline="0">
    <agent:predecessor task_Name="task_8" robot_Name="robot_3"/>
  </agent:task>
  <agent:task name="task_11" start="207" length="21" deadline="0">
    <agent:predecessor task_Name="task_8" robot_Name="robot_1"/>
  </agent:task>
  <agent:task name="task_12" start="229" length="20" deadline="0">
  </agent:task>
</agent:robot>
```

```
  <agent:basestation name="base_station">
    <agent:newtask name="task_1" arrive="5" length="19" deadline="25"/>
    <agent:newtask name="task_2" arrive="25" length="22" deadline="48"/>
    <agent:newtask name="task_3" arrive="30" length="22" deadline="53"/>
    <agent:newtask name="task_4" arrive="70" length="23" deadline="94"/>
    <agent:newtask name="task_5" arrive="90" length="17" deadline="108"/>
    <agent:newtask name="task_6" arrive="135" length="21" deadline="157"/>
    <agent:newtask name="task_7" arrive="185" length="23" deadline="209"/>
  </agent:basestation>
</agent:missiondata>
```

Bibliographie

[1] L. Adouane, *Architectures de contrôle comportementales et réactivés pour la coopération d'un groupe de robots mobiles*, Ph.D. thesis, U.F.R. des sciences et techniques, Université de Franche-Comté, Avril 2005.

[2] Foundation For Intelligent Physical Agents, *Fipa 07 specification part2 - agent communication language*, (1997).

[3] ———, *Sl content language specification*, (2001).

[4] ———, *Fipa abstract architecture specification*, (2002).

[5] S. Aknine, S. Pinson, , and M. Shakun, *An extended multi-agent negotiation protocol*, International Journal on Autonomous Agents and Multi-Agent Systems (2004).

[6] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert, *Multi robot cooperation in the martha project*, IEEE Robotics and Automation Magazine **5** (1997), no. 1, 36–45.

[7] J.S. Albus, C.R. McLean, A.J. Barbera, and M.L. Fitzgerald, *Hierarchical for control for robots and teleoperators*, Proceedings of IEEE International Workshop on Intelligent Control, vol. 1, Troy, New York USA, IEEE computer society, August 1985, pp. 39–49.

[8] R. C. Arkin, *Behavior-based robotics*, (1998).

[9] T. Balch and R. Arkin, *Communication in reactive multiagent robotic systems*, Journal of Autonomous Robots **1** (1994), no. 1, 27.52.

[10] E. Beaudry, Y. Brosseau, C. Cote, C. Raievsky, D. Letourneau, F. Kabanza, and F. Michaud, *Reactive planning in a motivated behavioral architecture*, (2005).

[11] F. Bellifemine, A. Poggi, and G. Rimassa, *Jade : a fipa2000 compliant agent development environment*, Proceedings the Fifth International Conference on Autonomous Agents (Montreal, Quebec, Canada), 2001, pp. 216–217.

[12] R.A. Brooks, *A robust layered control system for a mobile robot*, IEEE Journal of Robotics and Automation **2** (1986), no. 1, 14–23,.

[13] B.L. Brumitt and A. Stentz, *Grammps : A generalized mission planner for multiple mobile robots in unstructured environments*, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-98),, May 1998, pp. 1564–1571.

[14] R. Burkhart, *The swarm multi-agent simulation system*, September 1994, (OOPSLA), The Object Engine.

[15] Y. U. Cao, A. Fukunaga, and A. Kahng, *Cooperative mobile robotics : Antecedents and directions*, Autonomous Robots **4** (1997), no. 1, 7–27.

[16] S.L. CHENEVIER, *Planning, plan repair and execution control with time and resource management*, Ph.D. thesis, Institut National Polytechnique, Université de Toulouse, Juin 2004.

[17] A.D. Adelardo de MEDEIROS, *Controle d'exécution pour robots mobiles autonomes architecture, spécificationet, validation*, Ph.D. thesis, Laboratoire d'analyse et d'Architecture des Systèmes du CNRS, Université Paul Sabatier de Toulouse, Février 1997.

[18] M. de Weerdt, Y. Zhang, and T. Klos, *Distributed task allocation in social networks*, Proceedings of the 6th international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS'07), May 2007.

[19] M.B. Dias, R. Zlot, N. Kalra, and A. Stentz, *Market-based multirobot coordination : A survey and analysis*, Proceedings of the IEEE SpecialIssueonMulti-RobotCoordination **94** (2006), no. 7, 1257–1270.

[20] E.D. Durfee, V.R. Lesser, and D. Corkill, *Cooperation through communication in a distributed problem solving network*, Distributed Artificial Intelligence (1987).

[21] A. Farinelli, L. Iocchi, and D. Nardi, *Multirobot systems : A classification focused on coordination*, In IEEE Transaction on System, Man, and Cybernetics Part B **34** (2004), no. 5, 2015–2028.

[22] T. Finin, R. Fritzson, D. McKay, and R. McEntire, *Kqml as an agent communication language*, Proceedings of the Third International Conference on Information Management (CIKM), ACM, November 1994.

[23] B.P. Gerkey and M.J. Mataric, *A formal analysis and taxonomy of task allocation in multi-robot systems*, In International Journal of Robotics Research **23** (2004), no. 9, 939–954.

[24] M. Ghallab and H. Laruelle, *Representation and control in $i_X t_E t$, a temporal planner*, Proceedings of AIPS-94, 1994, pp. 61–67.

[25] O. Gutknecht and J. Ferber, *The madkit agent platform architecture*, Revised Papers from the International Workshop on Infrastructure for Multi-Agent Systems (2001), 48–55.

[26] M.J. Huber and E.H. Durfee, *eciding when to commit to action during observation-based coordination*, Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), 1995, pp. 163–170.

[27] T.L. Huntsberger, A.T. Ollennu, H. Aghazarian, P.S. Schenker, P. Pirjanian, and H.D. Nayar, *Distributed control of multi-robot systems engaged in tightly coupled tasks*, Autonomous Robots **17** (2004), 79–92.

[28] T.L. Huntsberger, P. Pirjanian, A.T. Ollennu, H. Das, H. Aghazarian, S.S. Joshi A.J. Ganino, M.S. Garrett, and P.S. Schenker, *Campout : A control architecture for tightly coupled coordination for multi-robot systems for planetary surface exploration*, In IEEE Transactions on Systems, Man, and Cybernetics **33** (2003), 555–559.

[29] j. Ferber, *Les systèmes multi-agents. vers une intelligence collective*, intereditions ed., 1995.

[30] N.R. Jennings, K. Sycara, and M. Wooldridge, *A roadmap of agent research and development*, Autonomous Agents and Multi-Agent Systems **1** (1998), no. 1, 7–38.

[31] E.G. Jones, M.B. Dias, and A. Stentz, *Learning-enhanced market-based task allocation for disaster response*, Tech. Report CMU-RI-TR-06-48, Robotics Institute, Carnegie Mellon University, 2001.

[32] B. Jung and G.S. Sukhatme, *A region-based approach for cooperative multi-target tracking in a structured environment*, In the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (2002), 2764–2769.

[33] L.P. Kaebling, M.L. Kaebling, and A.W. Moore, *Reinforcement learning : A survey*, Journal of Artificial Intelligence Research **4** (1996), 237–285.

[34] P Lopez, *Conception d'un système coopératif en gestion de production*, G. de Terssac & E. Friedberg (Eds.), Coopération et conception. (1996), 171–186, Toulouse : Octarès.

[35] B. Lussier, R. Chatila, F. Ingrand, M.O. Killijian, and D. Powell, *On fault tolerance and robustness in autonomous systems*, Tech. report, LAAS-CNRS, 2004.

[36] M. J. Mataric, *Situated robotics*, Tech. report, Invited contribution to the Encyclopedia of Cognitive Science, 2002.

[37] M.J. Mataric, *Behavior-based control : Examples from navigation, learning, and group behavior*, Journal of Experimental and Theoretical Artificial Intelligence **9** (1997), no. 2-3, 323–336, special issue on Software Architectures for Physical Agents.

[38] A. Meystel, *Intelligent control in robotics*, Journal of Robotic Systems **5** (1988), 269–308.

[39] N. Muscettola, G.A. Dorais, C. Fry, R. Levinson, and C. Plaunt, *Idea : Planning at the core of autonomous reactive agents*, In AIPS 2002 Works hop on On-line Planning and Scheduling (2002).

[40] I.A. Nesnas, A. Wright, M. Bajracharya, R. Simmons, and T. Estlin, *Claraty : An architecture for reusable robotic software*, Proceedings of SPIE Aerosense Conference, Orlando, Florida, 2003, pp. 121–132.

[41] M. C. Neves and E. Oliveira, *A control architecture for an autonomous mobile robot*, Proceedings of the first international conference on autonomous agents, 1997, pp. 193–200.

[42] N.J. Nilsson, *A mobile automaton : an aplication of artificial intelligence technique*, Proceedings International Joint Conference Artificial Intelligence, Washington, 1969, pp. 509–520.

[43] H. Nwana, *Software agents : An overview*, Knowledge Engineering Review, Cambridge University **11** (1996), no. 3, 1–40.

[44] H. Nwana, D. Ndumu, L. Lee, and J. Collis, *Zeus : A tool-kit for building distributed multi-agent systems*, Applied Artificial Intelligence Journal, **13** (1999), no. 1, 129–186.

[45] L.E. Parker, *Alliance : an architecture for fault tolerant multi-robot cooperation*, IEEE Transactions on Robotics and Automation **14** (1998), no. 2, 220–240.

[46] S. Fleury M. Ghallab R. Alami, R. Chatila and F. Ingrand, *An architecture for autonomy*, The International Journal of Robotics Research **17** (1998), no. 4, 315–337, Special Issue on Integrated Architectures for Robot Control and Programming.

[47] J. Rosenblatt, *Damn : A distributed architecture for mobile navigation*, proceedings of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents, Menlo Park, CA, AAAI, 1995.

[48] S. Sariel and T. Balch, *Efficient bids on task allocation for multi robot exploration*, AAAI, 2006.

[49] A. Shafer, A. Stentz, and C. Thorpe, *An architecture for sensor fusion in a mobile robot*, proceedings of International conference on robotics and automation, San Fransisco, California, April 1986, pp. 2002–2011.

[50] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes, *Coordination for multi-robot exploration and mapping*, Proceedings of the National Conference on Artificial Intelligence (AAAI-2000), 2000.

[51] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith, *First results in the coordination of heterogeneous robots for large-scale assembly*, Proceedings of the International Symposium on Experimental Robotics (ISER), December 2000.

[52] AHSRA Web Site, *Advanced cruise-assist highway system research association*, (2005).

[53] NASA JPL Web Site, *Robot work crew web page*.

[54] R.G. Smith, *The contract net protocol : high-level communication and control in a distributed problem solver*, IEEE Transactions on Computers, **C-28** (1980), no. 12, 1104–1113.

[55] Z. Toukal, *Contrôle/commande distribué d'une organisation d'agents robots*, Ph.D. thesis, U.F.R de Sciences et Technologie, Université Paris Val de Marne, France, Février 2000.

[56] M.H. Verrons, *Genca : un modèle général de négociation de contrats entre agents*, Ph.D. thesis, Laboratoire d'Informatique Fondamentale de Lille - UMR 8022, Université des Sciences et Technologies de Lille, Novembre 2004.

[57] G.E. Vieira, J.W. Herrmann, and E. Lin, *Rescheduling manufacturing systems : a framework of strategies, policies and methods*, Journal of Scheduling **6** (2003), no. 1, 39–62.

[58] L. Vig and J.A. Adams, *Multi-robot coalition formation*, IEEE Transactions On Robotics **22** (2006), no. 4.

[59] B. B. Wergerand and M.J Mataric, *Broadcast of local eligibility for multi-target observation*, Proceedings, 5th International Symposium on Distributed Autonomous Robotic Systems (DARS) **4** (2000), 347–356.