Planification de tâches pour un robot mobile autonome

par

Éric Beaudry

Mémoire présenté au Département d'informatique
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, août 2006

SOMMAIRE

Le présent mémoire propose une solution de planification de tâches pour un robot mobile et autonome appelé à participer au *AAAI Robot Challenge*, défi où un robot mobile doit participer à une conférence scientifique. Après une revue de diverses techniques de planification en intelligence artificielle, ainsi que l'évaluation de plusieurs planificateurs, l'algorithme de notre planificateur ConfPlan est élaboré sur le concept de planification HTN. ConfPlan intègre plusieurs stratégies permettant de gérer efficacement des contraintes de temps et de ressources (batteries). Ce planificateur est intégré dans l'architecture décisionnelle MBA. Plusieurs solutions d'intégration sont proposées. Enfin, le mémoire présente les résultats d'expérimentations du planificateur avec le robot Spartacus dans le contexte du *AAAI Robot Challenge*.

REMERCIEMENTS

Je veux tout d'abord remercier Froduald Kabanza et François Michaud pour avoir accepté de diriger mes travaux de recherche. Tout au long de mes études de maîtrise, leur grande disponibilité à superviser mes travaux témoigne de leur confiance en la réussite de mon projet de recherche. Leur enthousiasme et leur leadership dans les divers projets reliés à mes études ont été d'une grande et inspirante source de motivation.

Je tiens aussi à souligner l'importante contribution des équipes des laboratoires LABORIUS et PLANIART. Mes collègues étudiants et professionnels de recherche ont été d'une aide précieuse pour l'avancement de plusieurs tâches critiques dans mon projet.

Je tiens également à remercier ma famille qui m'a toujours soutenu tout au long de mes études.

En terminant, j'aimerais remercier le Conseil de la recherche en sciences naturelles et génie du Canada (CRSNG) pour son soutien financier à travers le programme de bourses d'études supérieures.

TABLE DES MATIÈRES

SOMMAIRE **ii**

REMERCIEMENTS **iii**

TABLE DES MATIÈRES **iv**

LISTE DES TABLEAUX **ix**

LISTE DES FIGURES **x**

LISTE DES ALGORITHMES **xiii**

INTRODUCTION **1**

CHAPITRE 1 — Notions de robotique **5**

1.1 Composantes d'un robot mobile 5

1.1.1 Capteurs 5

1.1.2 Actionneurs 6

1.1.3 Modules logiciels 7

1.2 Boucle de contrôle 8

1.3 Architectures décisionnelles 9
1.3.1 Architecture délibérative 9
1.3.2 Architecture comportementale 9
1.3.3 Architecture hybride 11

CHAPITRE 2 — Domaines d'applications en robotique **13**

2.1 Livraison de colis (« Goto Room ») 13
2.2 *AAAI Robot Challenge* 14

CHAPITRE 3 — Séquenceur de mission **16**

3.1 Machines à états finis 16
3.1.1 Exemple pour livraison de colis 16
3.1.2 Exemple pour un robot conférencier 17
3.1.3 Limitations 19
3.2 Planification automatique 20

CHAPITRE 4 — Planification de tâches **21**

4.1 Hypothèses courantes 21
4.2 Planification classique 22
4.3 Algorithmes de planification déterministe 26
4.3.1 Recherche à chaînage avant dans l'espace d'états 26
4.3.2 Recherche dans l'espace de plans 28
4.3.3 Recherche dans l'espace d'un graphe de planification 28
4.4 Stratégies de contrôle de recherche 28

4.4.1 Planification HTN 29
4.5 Planification heuristique 31
4.6 Planification concurrente 32
4.7 Planification avec métriques 32
4.8 Planification non déterministe 33

CHAPITRE 5 — Planification de chemin 35

5.1 Carte de l'environnement 35
5.2 Algorithmes de planification de chemin 36

CHAPITRE 6 — Évaluation de planificateurs 37

6.1 Planificateurs évalués 38
6.2 Tests et résultats 38
6.3 Limitations liées au langage PDDL 40

CHAPITRE 7 — ConfPlan : un planificateur pour robot mobile autonome 43

7.1 Contraintes temporelles 45
7.2 Gestion des priorités 46
7.3 Gestion des ressources 49
7.4 Optimisation et solution en tout temps 49
7.5 Intervalles de flexibilités temporelles 51
7.6 Implémentation 52
7.7 Spécification du domaine de la conférence scientifique 54
7.7.1 Tâches du domaine 55

7.7.2 Contrainte globale 57
7.7.3 Critère d'optimisation 57
7.8 Résultats 58

CHAPITRE 8 — Intégration de ConfPlan dans une architecture décisionnelle 62

8.1 Intégration du planificateur dans MBA 64
8.1.1 Réaction aux événements 64
8.1.2 Planification et exécution simultanées 66
8.1.3 Mise à jour de l'état projeté 66
8.1.4 Validation du plan courant 67
8.1.5 Compromis entre fiabilité et efficience 68
8.1.6 Détection d'opportunisme 69
8.2 Adaptations au domaine de la conférence 69
8.2.1 Monde continu versus représentation discrète 70
8.2.2 Points de repère inconnus sur la carte 72
8.3 Discussion 74

CHAPITRE 9 — Expérimentations avec le robot Spartacus 75

9.1 AAAI Robot Challenge 2005 76
9.1.1 Démonstration 77
9.1.2 Difficultés rencontrées 78
9.2 Conférence simulée avec le robot 80
9.2.1 Violation de contraintes temporelles 83

9.2.2 Détection d'opportunisme 83
9.2.3 Simplification de missions 86
9.2.4 Planification avec des endroits inconnus 87
9.2.5 Performances de planification 88

CHAPITRE 10 — Travaux futurs et améliorations à apporter 90

10.1 Localisation active 90
10.2 Planification concurrente 92
10.3 Modèle probabiliste du temps 93
10.4 Planification de tâches passives 93
10.5 Courbes de décharge des batteries 94
10.6 Approche générique 95

CONCLUSION 96

ANNEXE A — Code PDDL 2.1 pour le domaine de la conférence simulée 98

A.1 PDDL 2.1 niveau 2 - avec numériques 98
A.2 PDDL 2.1 niveau 5 - valeurs numériques et temporelles 102

ANNEXE B — Expériences au 5^{e} étage 106

BIBLIOGRAPHIE 107

LISTE DES TABLEAUX

7.1 Tableau des temps (sec) de planification des planificateurs évalués. 61

9.1 Temps de planification en milliseconde. 89

LISTE DES FIGURES

1.1 Capteurs typiques d'un robot 6

1.2 Perceptions des quelques capteurs 6

1.3 Boucle de contrôle 8

1.4 Exemple d'architecture comportementale 10

1.5 Exemples de comportements 11

1.6 Architecture hybride 12

2.1 Domaine de livraison de colis 13

2.2 Plan de la conférence simulée 15

3.1 Machine à états finis pour un robot livreur 17

3.2 Machine à états finis pour un robot conférencier 18

3.3 Séquenceur de mission avec un planificateur de tâches. 20

4.1 Modèle de planification 22

4.2 Recherche à chaînage avant 27

4.3 Exemple de spécification d'un réseau de tâches en HTN 31

5.1 Exemple de chemin dans une grille d'occupation 36

6.1 Comparaison des planificateurs 42

7.1 Exemples de propagation des contraintes de temps 46

7.2 Exemple de plan sous forme textuelle. 52

7.3 Exemple de plan sous forme de diagramme de Gantt. 52

7.4 Convergence vers solution optimale. 53

7.5 Tâches pour le domaine de la conférence scientifique. 55

7.6 Comparaison des planificateurs 60

8.1 Architecture MBA 63

8.2 ConfPlan avec action courante 67

8.3 Robot des positions ambigues 70

9.1 Le robot Spartacus 76

9.2 Notre équipe au AAAI Robot Challenge 77

9.3 Carte pour le AAAI Robot Challenge au Westin Convention Center . . . 78

9.4 Positions estimées du robot au AAAI Robot Challenge 80

9.5 Vitesse estimée du robot au AAAI Robot Challenge 81

9.6 Carte du 5^e étage avec 11 points de repère 81

9.7 Trace du robot lors d'une expérience au 5^e étage. 82

9.8 Plans pour l'expérience de violation de contraintes temporelles. 84

9.9 Trace d'exécution pour l'expérience de violation de contraintes temporelles. 84

9.10 Plans pour l'expérience de détection d'opportunisme. 85

9.11 Trace d'exécution pour l'expérience de détection d'opportunisme. 85

9.12 Plans pour l'expérience de la simplification d'une mission. 86

9.13 Trace d'exécution pour l'expérience de la simplification d'une mission. . . 86

9.14 Plans pour l'expérience avec un endroit inconnu. 87

9.15 Trace d'exécution pour l'expérience de la simplification d'une mission. . . 88

B.1 Carte du 5^e étage avec table de distances 106

LISTE DES ALGORITHMES

1 Planificateur ConfPlan 44
2 Calcul des intervalles de flexibilités temporelles 51
3 Boucle de planification réactive 65

INTRODUCTION

Depuis fort longtemps, l'humain rêve de créer des machines intelligentes capables d'effectuer des tâches à sa place. Ainsi, les humains auraient plus de temps à consacrer pour leurs loisirs, ou prendraient moins de risques pour effectuer des tâches dangereuses. Or créer une machine pouvant réaliser des tâches que seuls les humains sont normalement capables de faire n'est pas aussi simple qu'on pourrait le penser. En effet, sans toujours y penser, les tâches les plus élémentaires de la vie quotidienne d'un humain peuvent devenir extrêmement complexes lorsqu'on les analyse de plus près. Par exemple, participer à un congrès ou à une conférence peut devenir un véritable casse-tête. Avant tout, il faut consulter son agenda afin de s'assurer de sa disponibilité. En présence de conflits d'horaire, il est alors nécessaire de négocier des arrangements avec les personnes concernées. Une fois cette étape franchie, il faut s'inscrire et payer les frais d'inscription. Par la suite, il faut se rendre sur les lieux de l'événement en combinant divers moyens de transport comme l'automobile, l'autobus et peut-être même l'avion. Une fois arrivé sur place, il faut repérer le kiosque d'enregistrement et s'y présenter afin d'obtenir son porte-nom ainsi que tous les autres documents pertinents, comme le programme de l'événement. De plus, tout au long du congrès ou de la conférence, il faut se rendre dans les bonnes salles, et ce, aux bons moments. Bien que ces tâches semblent simples pour nous les humains, cela est loin d'être aussi évident pour un robot.

Afin d'être autonome, un robot mobile doit posséder de nombreuses capacités. Premièrement, il doit être capable de percevoir son environnement et de se localiser dans celui-ci. Pour ce faire, un robot possède des capteurs, comme des sonars et un dispositif à balayage laser servant à mesurer des distances entre lui-même et les obstacles à proximité.

Une fois localisé dans son environnement, le robot doit être capable de se déplacer d'un point à l'autre en trouvant des chemins efficaces et sécuritaires afin d'éviter des collisions avec les obstacles. De plus, un robot est souvent appelé à communiquer avec les gens ou d'autres agents situés à proximité. Cela peut être fait de différentes façons, comme par communication vocale ou depuis une interface graphique.

En plus de pouvoir percevoir globalement son environnement, un robot doit souvent être capable d'identifier des objets, de reconnaître des personnes, de lire des indications, et même de repérer des symboles graphiques. Ces opérations sont effectuées en analysant les images acquises par la ou les caméras installées sur le robot. Après avoir identifié et localisé un objet, on peut imaginer que le robot ait ensuite à manipuler cet objet avec son bras-robot.

Bien que beaucoup d'autres capacités de perception et d'action pourraient être ajoutées à cette liste, l'important est d'avoir à l'esprit que la robotique mobile est un domaine hautement multidisciplinaire qui fait l'objet de beaucoup de recherches dans des disciplines très diversifiées. Pour cette raison, dans le présent mémoire, nous faisons abstraction de la plupart de ces capacités robotiques et nous considérons avoir accès à celles-ci puisqu'ils découlent de d'autres travaux, indépendamment de ceux présentés ici.

Enfin, une autre capacité robotique, tout aussi importante que celles énumérées précédemment, est la capacité pour un robot de prendre lui-même ses propres décisions pour réaliser et coordonner des missions complexes. Cette capacité est d'une grande importance puisque, pour beaucoup de tâches robotiques, il peut exister de nombreuses façons de les réaliser. Par raisonnement, le robot doit sélectionner les meilleures actions à effectuer pour réussir adéquatement sa mission.

De plus, dans certaines missions robotiques, il est impératif de bien séquencer les tâches afin de respecter diverses contraintes permettant au robot d'avoir un comportement cohérent et efficace. Par exemple, pour des tâches de livraison de colis, un certain ordre d'exécution doit être respecté : il faut aller à l'endroit de cueillette avant de saisir l'objet et ensuite se rendre à l'endroit de livraison avant de le déposer. Il peut également arriver qu'un robot ait à exécuter des tâches à l'intérieur d'une certaine fenêtre de temps.

Au fil des ans, plusieurs solutions ont été proposées afin de donner aux robots la capacité de coordonner leurs missions. L'une d'entre elles est l'utilisation de techniques de planification dans le domaine de l'intelligence artificielle. Dans le présent mémoire, nous nous concentrons quasi exclusivement sur ce sujet. Nous étudions plus précisément la planification et la coordination de missions pour des robots mobiles et intelligents évoluant dans des environnements dynamiques et incertains. Nous proposons un module de planification réactive spécialement adapté à la robotique mobile, et nous accordons une attention particulière à son intégration dans une architecture robotique. La solution globale proposée répond à trois objectifs principaux : améliorer l'autonomie des robots en permettant une plus grande flexibilité dans l'exécution de missions complexes, faciliter l'attribution de missions aux robots et augmenter l'efficacité des robots dans l'accomplissement de leurs tâches.

Comme banc d'essai, nous avons expérimenté la solution de planification proposée dans le contexte du *AAAI Robot Challenge*, un défi dans lequel un robot est appelé à participer à une conférence scientifique. Pour réussir cet ambitieux défi, un large éventail de solutions doivent être intégrées afin de répondre à différentes facettes du problème. Entre autres, au cours de cet événement, le robot doit être en mesure de planifier lui-même sa mission et s'adapter au fur et à mesure que la conférence progresse.

Les travaux présentés dans ce mémoire répondent à cette problématique précise, soit de donner des capacités de planification à Spartacus, un robot conçu par le laboratoire LABORIUS. En premier lieu, nous présentons ConfPlan, un nouvel algorithme de planification mieux adapté à cette problématique. Dans une seconde étape, nous élaborons un domaine de planification, destiné à ce nouveau planificateur, pour le contexte du *AAAI Robot Challenge*. Enfin, nous discutons comment le planificateur et la spécification du domaine ont été modifiés afin d'être intégré dans MBA, une architecture robotique conçue par notre équipe.

Ce mémoire est organisé de la façon suivante. Le premier chapitre introduit des notions de base en robotique mobile. Le chapitre 2 décrit les domaines d'applications robotiques de références pour les chapitres suivants. Le chapitre 3 donne des exemples de fonction-

nement d'un séquenceur de mission. Le chapitre 4 introduit la planification de tâches et explique les principales approches. Le chapitre 5 explique brièvement le rôle du planificateur de chemin. Le chapitre 6 détaille l'évaluation de planificateurs existants. Le chapitre 7 décrit la conception et l'implémentation du planificateur ConfPlan. Le chapitre 8 présente les solutions utilisées pour intégrer ConfPlan dans l'architecture décisionnelle MBA. Le chapitre 9 explique les résultats d'expérimentations réalisées avec le robot Spartacus. Le chapitre 10 ouvre la porte à des idées à explorer comme travaux futurs.

CHAPITRE 1

Notions de robotique

Avant d'entrer dans le cœur du sujet, il est important d'avoir une idée générale du fonctionnement d'un robot mobile afin de bien comprendre les interactions entre les différents modules auxquels nous faisons référence. Pour une couverture plus approfondie sur le sujet, les volumes [3, 34] sont recommandés.

1.1 Composantes d'un robot mobile

À la base, un robot mobile est constitué de composantes matérielles et logicielles. Parmi les composantes matérielles, on retrouve une plateforme mobile à laquelle sont rattachées toutes les autres composantes comme les capteurs, les actionneurs et une source d'énergie (batteries).

1.1.1 Capteurs

Les capteurs ont pour fonction d'acquérir des données provenant de l'environnement. Les capteurs typiquement installés sur un robot mobile (voir figure 1.1) sont des sonars à ultrasons, un capteur laser de proximité, des encodeurs de roues (odomètres), une ou deux caméras optiques et des microphones. Les types d'informations perçues ainsi

que leur précision varient beaucoup d'un capteur à l'autre. Par exemple, la figure 1.2 montre qu'un capteur laser de proximité (c) permet de mieux percevoir les contours de l'environnement (a) que les sonars (b) puisque le capteur il offre une meilleure résolution angulaire et une meilleure précision sur la distance.

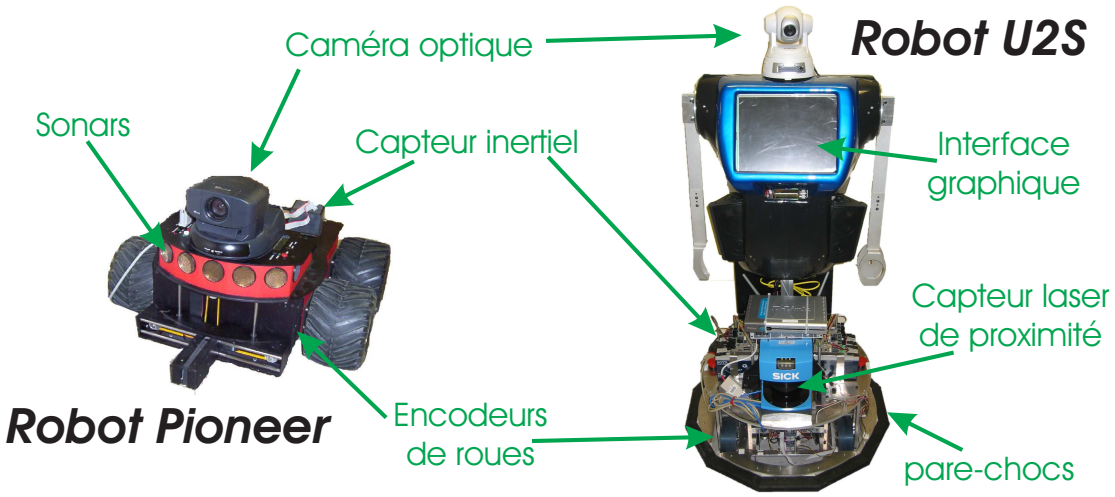
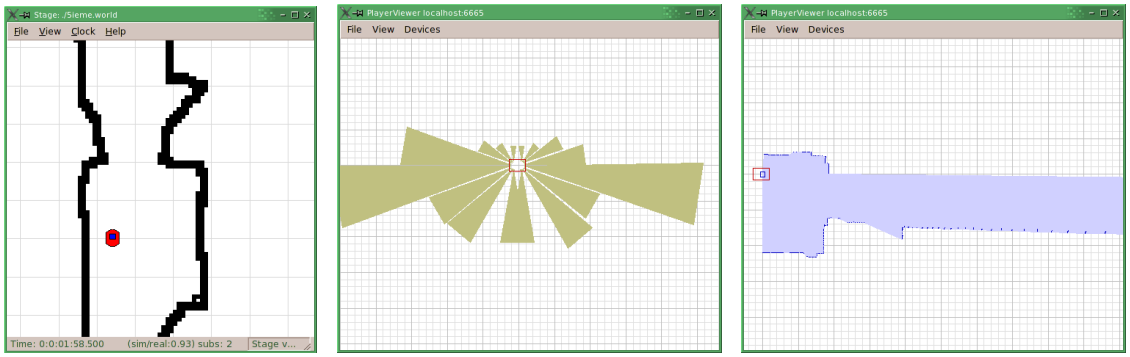


Figure 1.1 – Capteurs typiques d'un robot.



(a) Simulateur Stage (b) Vue des sonars (c) Vue du laser

Figure 1.2 – Perceptions des quelques capteurs.

1.1.2 Actionneurs

Pour bouger à l'intérieur de son environnement et interagir avec celui-ci, un robot est équipé d'actionneurs. Par exemple, un robot est muni d'un ou de plusieurs moteurs pouvant faire tourner ses roues afin d'effectuer des déplacements. Généralement, les roues du robot sont contrôlées par deux commandes motrices, soit une vitesse d'avancement et

un taux de rotation. Habituellement, ces commandes s'expriment en mètres par seconde (m/s) et en degrés de rotation par seconde (deg/s).

1.1.3 Modules logiciels

Afin de faire fonctionner un robot mobile, plusieurs modules logiciels sont mis à contribution. Ces modules peuvent servir à interpréter les données perçues par les capteurs afin d'en extraire des informations, ou à traiter des commandes de haut niveau pour générer d'autres commandes à un niveau inférieur. Les modules les plus fréquemment utilisés sont les modules de localisation, de navigation, de vision, d'audio et de séquencement d'activités du robot.

Localisation

L'une des fonctions les plus importantes pour un robot est celle d'être capable de se localiser dans son environnement. À partir des données fournies par les capteurs, le module de localisation estime la position courante du robot. Typiquement, cette position est exprimée par un triplet (X, Y, θ) représentant une position et une orientation sur un plan à deux dimensions. La localisation peut se faire à l'aide de techniques basées sur la théorie des processus de décisions markoviennes [18], à l'aide de techniques d'échantillonnage de Monte Carlo (filtres de particules) [40] ou avec d'autres méthodes.

Vision

En analysant les images captées par les caméras, on peut extraire une multitude d'informations. Par exemple, à l'aide d'un algorithme de segmentation [21], on peut reconnaître des objets de couleur en plus d'estimer leur position relative (angle) par rapport à la vue de la caméra. À l'aide de techniques de vision tridimensionnelle [41], il est aussi possible d'estimer certaines distances dans l'environnement. On peut aussi reconnaître des symboles, des caractères et lire des messages [30], comme des affiches dans un corridor, des signaux de direction, ou des badges de conférence.

Navigation

Un module de navigation est responsable de déplacer un robot de sa position courante vers une destination désirée de façon sécuritaire et efficace. En plus d'inclure des fonctions de perception de l'environnement et de localisation, le module de navigation a aussi la responsabilité de trouver un chemin reliant la position d'origine et de destination, formé d'une liste de points intermédiaires à atteindre, et de guider le robot à travers ce chemin. Pour trouver efficacement un chemin, on utilise un planificateur de chemin (voir chapitre 5), comme les algorithmes A* [23], D* ou Anytime Dynamic A* [28].

1.2 Boucle de contrôle

Un robot mobile est commandé par une boucle de contrôle, comme illustré à la figure 1.3. De façon itérative, cette boucle fait une lecture des données reçues par les capteurs, les interprète, calcule les commandes motrices et les envoie aux actionneurs. Typiquement, cette boucle est exécutée environ dix (10) fois par seconde ; la fréquence peut varier selon les types de capteurs et d'actionneurs utilisés. La boucle de contrôle n'est pas unique ; selon l'architecture utilisée, elle peut être décomposée en plusieurs sous-boucles de contrôle agencées de manières différentes.

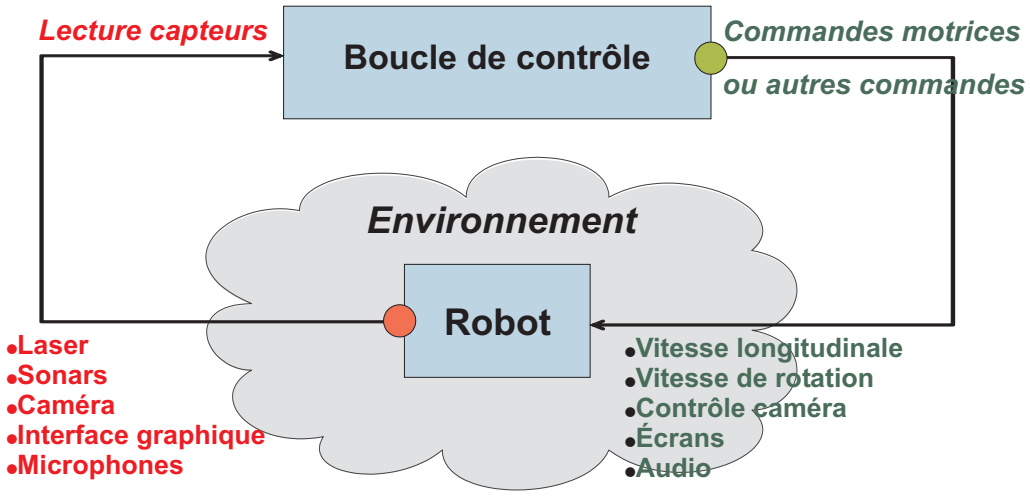


Figure 1.3 – Boucle de contrôle.

1.3 Architectures décisionnelles

Un robot est donc composé d'un ensemble de modules, chacun étant responsable d'une ou de plusieurs capacités. L'un des premiers défis à résoudre est de déterminer comment relier efficacement les différents modules. Pour ce faire, il faut élaborer une architecture décisionnelle qui dicte les responsabilités de chacun des modules et comment l'information circule entre ces derniers. Depuis les débuts de la robotique, beaucoup d'architectures ont été proposées. Elles peuvent être généralement classées en trois grandes catégories : délibérative, comportementale et hybride.

1.3.1 Architecture délibérative

Les architectures délibératives [34] sont les premières à avoir été proposées. Comme son nom l'indique, les architectures de ce type sont basées sur des processus complètement planifiés. Par exemple, afin d'exécuter un déplacement, un robot basé sur ce type d'architecture calcule un plan complet, lui disant d'avancer de x mètres, ensuite de tourner de y degrés, et ainsi de suite. Lorsqu'un changement dans l'environnement est perçu, l'exécution est suspendue et un nouveau plan est généré.

Ce type d'architecture souffre de plusieurs lacunes importantes. Premièrement, puisque les capteurs sont imprécis et que l'environnement est dynamique et partiellement observable, il est très difficile de tout prévoir à l'avance. Pour ces raisons, il n'est pas d'une très grande utilité de tout planifier à l'avance, puisque les plans seront constamment à refaire. Un autre problème avec ce type d'architecture est que la génération de plans précis demande beaucoup de ressources (temps de calcul et mémoire).

1.3.2 Architecture comportementale

L'architecture comportementale, proposée par Brooks [10], est inspirée par le comportement des insectes [3]. L'idée générale est de développer plusieurs petits modules simples et indépendants les uns des autres et, une fois regroupés, un comportement plus intel-

ligent émerge sans qu'il ait été spécifiquement programmé. Ce type d'architecture est complètement à l'opposé des architectures délibératives et ne fait aucune place à des processus raisonnés.

Un exemple d'architecture comportementale est illustré à la figure 1.4. Celle-ci est composée de plusieurs modules simples appelés **comportements** et d'un module d'arbitration. Dans sa conception, chaque comportement se limite à une seule fonctionnalité pour le contrôle du robot. Les comportements sont tous indépendants les uns des autres. Par exemple, on peut avoir des comportements pour l'évitement d'obstacles, le suivi de chemin, le suivi d'objets de couleur ou la manipulation d'objets. Les comportements sont exécutés parallèlement à une certaine fréquence (habituellement dix fois par seconde). Lors d'une itération, chaque comportement calcule une ou plusieurs commandes motrices qui sont envoyées à un module d'arbitration. Ce dernier fusionne l'ensemble des commandes reçues et calcule les commandes finales devant être envoyées à chacun des actionneurs du robot.

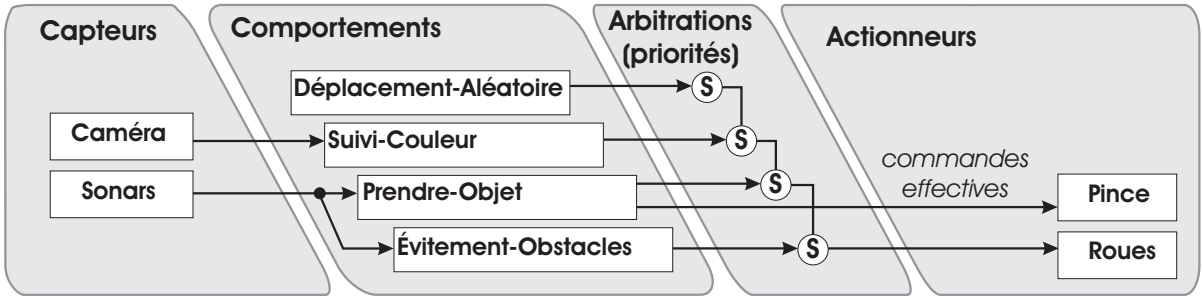
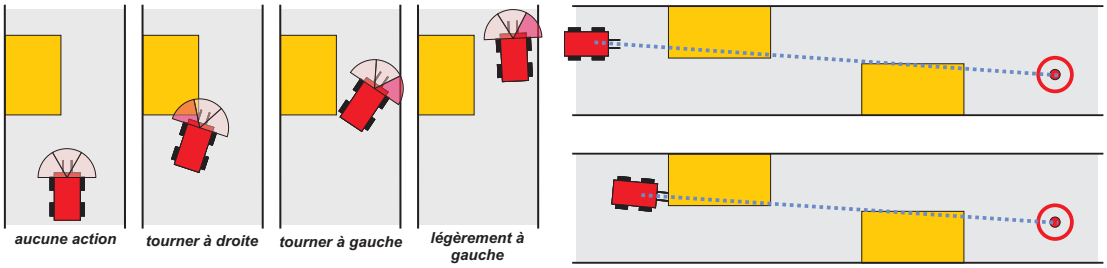


Figure 1.4 – Exemple d'architecture comportementale.

En guise d'exemple, examinons l'intégration d'un comportement d'évitement d'obstacles et d'un comportement de suivi de chemin. En présence d'un obstacle sur un côté (voir figure 1.5(a)), le comportement d'évitement d'obstacles ne fait que tourner vers le côté opposé. Indépendamment, le comportement de suivi de chemin (souvent appelé «Goto» en anglais) aligne et dirige en ligne droite le robot vers la cible courante. Comme nous pouvons l'apercevoir à la figure 1.5(b), ce comportement ignore la présence d'obstacles.

Afin de diriger le robot vers la cible sans collision, les capacités de ces deux comportements doivent être fusionnées par un mécanisme d'arbitration. L'approche classique consiste



(a) Évitement d'obstacles (b) Suivi de chemin

Figure 1.5 – Exemples de comportements.

à utiliser une arbitration par priorité. Dans le cas présent, l'évitement d'obstacles a priorité sur le suivi de chemin. Lorsqu'aucun obstacle n'est détecté, le module d'évitement d'obstacles ne génère aucune commande et laisse le plein contrôle au module de suivi de chemin. Par contre, quand un obstacle est détecté, le module d'évitement substitue la commande du module de suivi de chemin par une commande de rotation dans le sens opposé à l'obstacle perçu. Lorsque l'obstacle est complètement évité et sorti du champ de vision des capteurs, le comportement de suivi de chemin reprend le plein contrôle du robot.

En contrepartie, les architectures comportementales ont de la difficulté à réaliser des tâches structurées, puisqu'elles ne contiennent aucun processus délibératif. En effet, les tâches complexes requièrent la capacité du robot à prédire les conséquences futures de ses actions afin de sélectionner celles qui conviennent le mieux pour la réalisation de ses activités. En d'autres mots, ces tâches complexes ont besoin d'être planifiées.

1.3.3 Architecture hybride

Les limitations des deux types d'architecture précédentes justifient l'émergence récente des architectures hybrides [34], tentant de combiner les avantages des architectures délibératives et comportementales. Elles sont généralement décomposées en plusieurs niveaux. Dans la partie supérieure, on place les modules de type délibératif. Dans la partie inférieure, on retrouve les modules de type comportemental. La figure 1.6 illustre un exemple d'architecture hybride où un séquenceur s'occupe des capacités délibératives.

C'est ce type d'architecture qui est utilisé pour les travaux de ce mémoire. Le chapitre 3 explique plus en détail le fonctionnement d'un séquenceur.

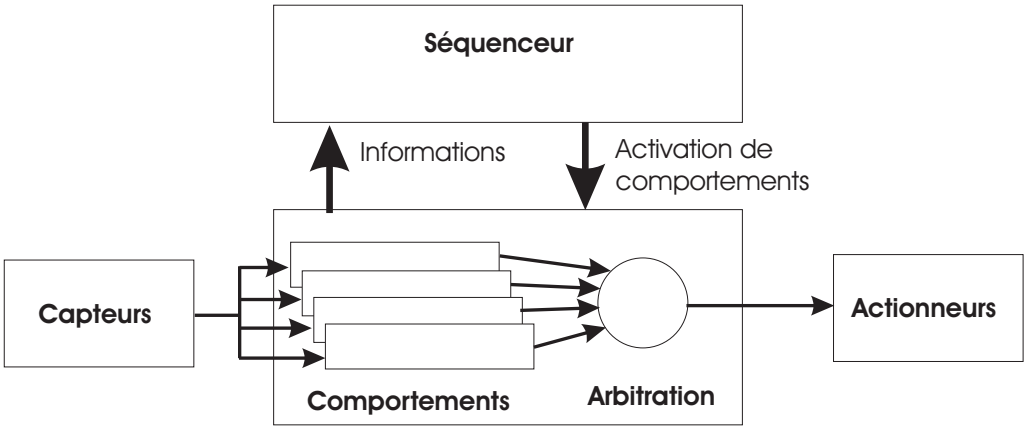


Figure 1.6 – Architecture hybride.

CHAPITRE 2

Domaines d'applications en robotique

Deux domaines d'applications robotiques sont utilisés pour les travaux de ce mémoire et sont décrits dans ce chapitre. Le domaine de livraison de colis sert principalement de référence pour les chapitres suivants et celui de la conférence simulée sert à mieux définir le type d'applications ciblées pour les travaux présentés.

2.1 Livraison de colis (« Goto Room »)

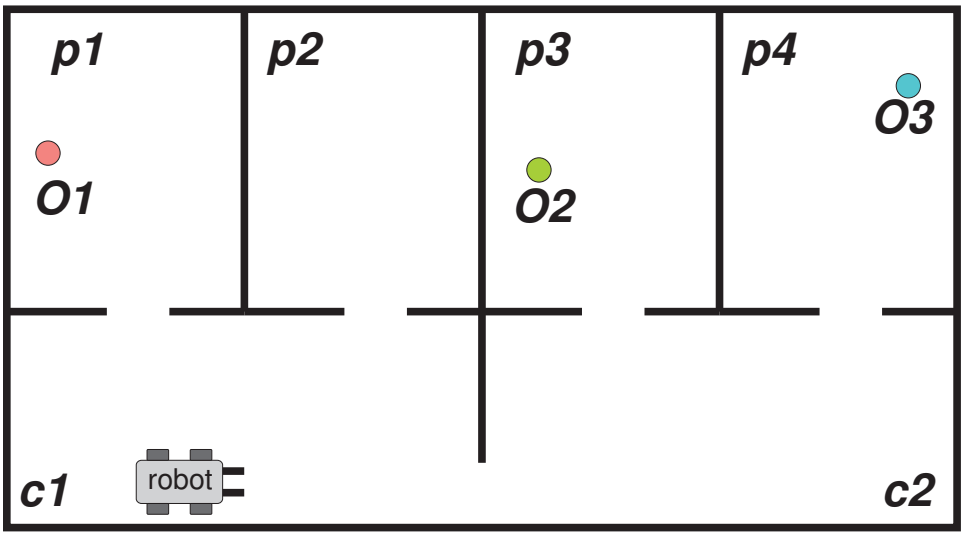


Figure 2.1 – Domaine de livraison de colis.

Dans ce domaine, le robot a pour mission de livrer des objets d'une pièce à une autre. De plus, le robot ne peut saisir et transporter qu'un seul objet à la fois. Dans l'exemple illustré à la figure 2.1, on peut demander au robot de déplacer le bloc *o1* dans la pièce *p2*, le bloc *o2* dans la pièce *p4* et le bloc *o3* dans la pièce *p1*. Une solution à ce problème serait le plan suivant : *{AllerÀ(c2), AllerÀ(p3), Prendre(o2), AllerÀ(c2), AllerÀ(p4), Deposer(o2), Prendre(o3), AllerÀ(c2), AllerÀ(c1), AllerÀ(p1), Deposer(o3), Prendre(o1), AllerÀ(c1), AllerÀ(p2), Deposer(o1)}*. Les tâches *AllerÀ*, *Prendre*, *Déposer* sont considérées comme des combinaisons de comportements primitifs. Une fois isolées, elles ne requièrent aucune capacité délibérative et peuvent être réalisées simplement par l'activation de comportements.

2.2 *AAAI Robot Challenge*

Initié en 1999, le *AAAI Robot Challenge* est un défi où un robot est appelé à participer à une conférence scientifique de manière autonome [31]. Comme son nom l'indique, cet événement est organisé par l'*American Association for Artificial Intelligence* (AAAI) et a lieu pendant la *National Conference on Artificial Intelligence*. Comme l'idée de faire participer un robot à une conférence semble être de la science-fiction, cet événement devient donc un défi très audacieux. De nombreux problèmes de différentes natures doivent être résolus et plusieurs modules décisionnels doivent être intégrés ensemble pour mener à bien ce projet.

Les participants au *AAAI Robot Challenge* ont une certaine liberté dans la façon de réaliser ce défi puisqu'il n'y a pas de structure rigide imposée. Généralement, on propose aux participants de réaliser un certain nombre de tâches qui sont typiques d'une conférence. Le robot est entre autres invité à s'enregistrer à l'endroit approprié, à effectuer des tâches bénévoles comme de transporter des objets, de livrer des messages et de surveiller des salles. Le robot peut aussi servir d'agent d'information et donner des indications aux autres invités. À la fin du défi, on suggère que le robot donne une courte présentation dans laquelle il peut expliquer brièvement son fonctionnement interne, dire en quoi il est

original et enfin, résumer son expérience à la conférence.

Ce défi représente un intéressant banc d'essai pour un module de planification autonome qui permettrait à un robot participant de planifier et de coordonner ses activités lors d'une conférence. En effet, le planificateur peut conférer au robot une capacité d'anticipation. Par exemple, l'heure de présentation du robot peut être fixée au moment de l'enregistrement et une tâche de surveillance d'une salle peut avoir lieu entre deux sessions de présentations. Ainsi, le robot peut accomplir un maximum de tâches bénévoles en tenant compte de son autonomie énergétique limitée et de ses tâches obligatoires.

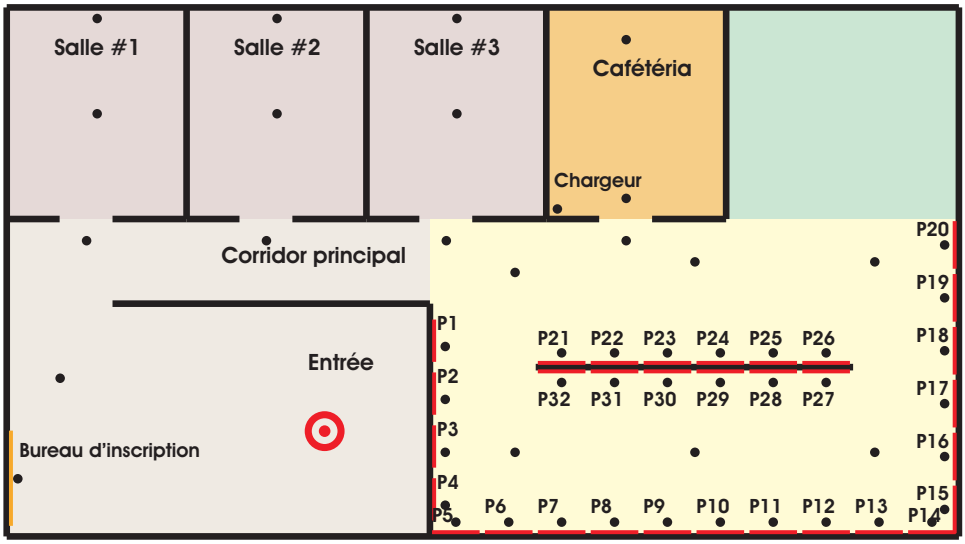


Figure 2.2 – Plan de la conférence simulée.

Durant la préparation pour participer au *AAAI Robot Challenge* de 2005, un environnement de conférence simulée a été développé. Il est illustré à la figure 2.2. Le robot simulé peut se voir confier les tâches suivantes :

- s'enregistrer au bureau d'inscription ;
- assister à des présentations ;
- donner une ou plusieurs présentations ;
- installer et retirer des affiches ;
- prendre en photo des affiches ;
- livrer des messages d'un endroit à un autre ;
- surveiller des salles.

CHAPITRE 3

Séquenceur de mission

Dans une architecture robotique hybride telle que décrite à la section 1.3.3), plusieurs stratégies peuvent être utilisées pour séquencer les étapes d'une mission.

3.1 Machines à états finis

Une méthode classique pour la coordination de missions est l'utilisation de machines à états finis. Une **machine à états finis** est un automate dans lequel on retrouve des états et des transitions. Chaque **état** représente un **mode** précis du système dans lequel un certain nombre de modules sont configurés et activés. Dans le cas d'une architecture de type hybride, chaque état est associé à une liste des comportements à activer, et ce, avec des paramètres prédéterminés. Les **transitions** de l'automate permettent au système de changer d'état (mode) lorsqu'un événement particulier survient, comme la terminaison d'une tâche ou un signal d'horloge.

3.1.1 Exemple pour livraison de colis

Une machine à états finis simple est illustrée à la figure 3.1. Cette dernière permet de coordonner un robot devant livrer l'objet *o1* de l'endroit *p1* à l'endroit *p2* (voir figure 2.1).

Initialement, le système se retrouve dans l'état *Début* et fait une transition immédiate à l'état *AllezÀ(p1)*. Cet état configure le robot de manière à ce qu'il se déplace vers le point *p1*, c'est-à-dire que ce mode active le comportement d'évitement d'obstacles ainsi que le comportement de suivi de chemin ayant pour cible le point *p1*. De façon continue, le séquenceur de mission vérifie la condition d'arrivé à *p1*, et lorsque le robot y arrive, le séquenceur fait une transition à l'état *Prendre(o1)*. Cet état active le comportement de prise d'objet et désactive celui de suivi de chemin.

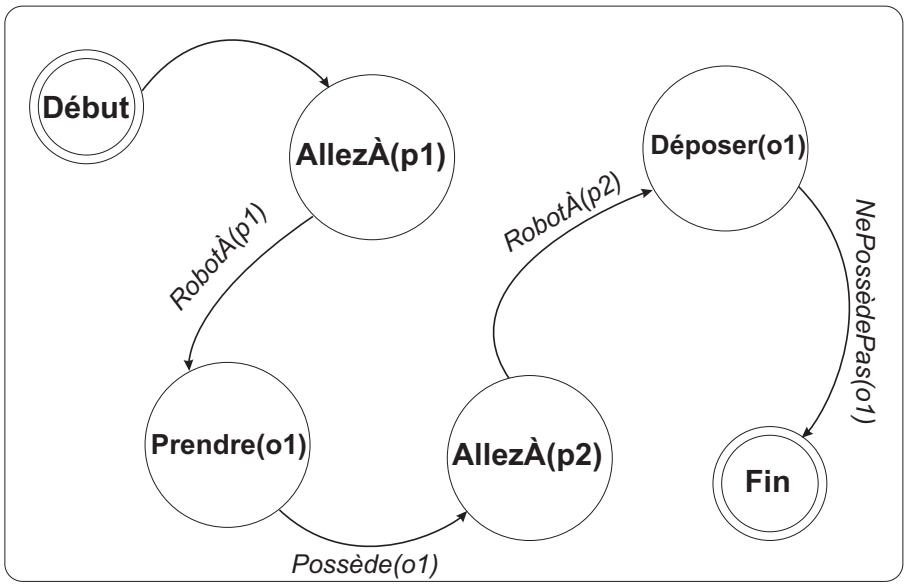


Figure 3.1 – Machine à états finis pour un robot livreur.

3.1.2 Exemple pour un robot conférencier

Pour le contrôle de haut niveau d'un robot conférencier devant présenter une affiche et faire un exposé oral, on pourrait utiliser la machine à états finis présentée à la figure 3.2. À 9 heures, le robot commence par son inscription en activant des comportements comme la recherche du kiosque d'information. Dès qu'il a reçu son porte-nom, il se considère comme étant maintenant inscrit à la conférence et bascule dans le mode *aller au panneau* dans lequel il active le module de navigation pour se rendre à son kiosque attitré. Une fois sur place, si son affiche n'est pas déjà installée, il l'installe. Ensuite, il bascule en mode d'attente de visiteurs. Dans ce mode, il active un comportement exploitant la caméra

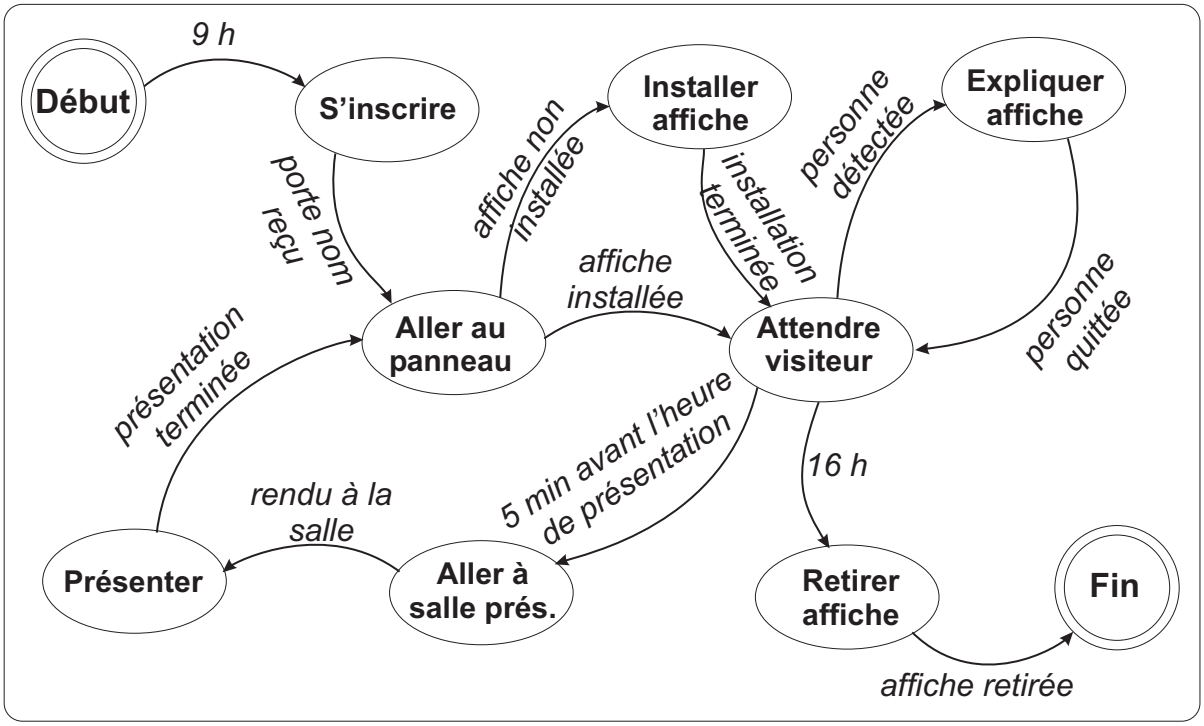


Figure 3.2 – Machine à états finis pour un robot conférencier.

pour détecter des visages humains et un autre qui analyse les sons captés par ses microphones. Lorsqu'un visage est détecté face au robot ou qu'une phrase clé comme « Peux-tu m'expliquer ton affiche ? » est entendue, le robot passe alors au mode d'explication de l'affiche. Dans ce mode, il peut simplement démarrer une vidéo ou un diaporama sur l'écran du robot. Lorsqu'il détecte que la personne a quitté, il ne fait que revenir dans le mode d'attente. À partir de l'état *attendre visiteur*, il y a aussi une transition spéciale qui s'effectue sur une condition temporelle : 5 minutes avant l'heure à laquelle le robot doit faire son exposé. Cette transition amène le robot à se déplacer vers la salle de conférence et une fois arrivé, il passe dans un mode de présentation qui est similaire à la présentation de l'affiche. Après son exposé, le robot retourne au panneau d'affichage et se remet en mode d'attente. Enfin, à 16 heures (la fin de la conférence), le robot retire son affiche et termine ainsi sa mission.

3.1.3 Limitations

Les limitations des machines à états finis découlent du fait qu'elles sont programmées manuellement. En plus de demander beaucoup de temps, cette opération pose plusieurs problèmes.

- **Flexibilité.** Les machines à états finis manquent de flexibilité. Plus on ajoute des tâches à la mission du robot, plus la machine à états finis devient complexe. Avec un grand nombre d'états et de transitions, il devient difficile de les modifier efficacement pour traiter l'ensemble des possibilités.
- **Erreurs.** Un autre problème est le risque d'erreurs. Puisque les machines à états finis sont conçues à la pièce, il est possible que des erreurs de conception s'y glissent. De plus, il devient très difficile de prévoir tous les cas possibles.
- **Efficacité.** Puisqu'elles doivent être conçues à l'avance, les machines à états finis ne sont pas toujours efficaces. Lors de leur conception, un ordre entre les tâches peut être préétabli en fonction des durées anticipées des tâches. Or, si certaines tâches progressent plus rapidement que prévu, il peut être nécessaire de changer l'ordre d'exécution de certaines tâches afin de les terminer plus rapidement.
- **Mission dynamique.** Dans de nombreux cas, un robot doit accepter des missions dynamiques, c'est-à-dire des missions spécifiées en cours d'exécution contrairement à des missions totalement déterminées à l'avance. Pour cela, il n'est pas possible d'utiliser des machines à états finis préprogrammées. Il faut plutôt les générer dynamiquement, c'est-à-dire les planifier.
- **Facilité à coder une mission.** Pour être utile, un robot doit offrir une interface de spécification de mission simple. L'utilisation de machines à états finis peut devenir une opération complexe, longue et coûteuse. Une interface où l'usager n'aurait qu'à spécifier des tâches dans un langage de haut niveau serait une solution beaucoup plus convenable. Par exemple, les tâches de la figure 3.2 devraient découler automatiquement d'un but donné au robot lui demandant de présenter une affiche et de faire une présentation à une heure donnée.

3.2 Planification automatique

Une alternative à l'utilisation de machines à états finis, permettant de pallier tous les problèmes précédents, pour codifier une mission robotique, est l'utilisation d'un module de planification. Au lieu de décrire le séquencement complet du robot, l'usager n'a qu'à fournir les tâches de haut niveau à effectuer, comme de faire une présentation à une certaine heure. Avec des connaissances sur les tâches elles-mêmes, le module de planification raisonne, détermine les tâches sous-jacentes et trouve une façon optimale de les agencer. La figure 3.3 illustre une nouvelle représentation de l'architecture décisionnelle hybride qui inclut un planificateur de tâches. Ce dernier a accès à un modèle de l'environnement en plus des informations extraites de l'environnement. À partir de la mission courante du système, le planificateur produit un plan qui est envoyé à un module d'exécution. Ce dernier se charge d'envoyer les bonnes activations de comportements selon l'étape courante du plan en cours d'exécution. Le module d'exécution a aussi la responsabilité de vérifier la validité du plan courant. Lors d'une situation imprévue, certaines actions peuvent devenir impossibles à réaliser tel qu'elles ont été planifiées. Par exemple, si l'action courante est d'entrer dans une salle et que la porte est fermée, le module d'exécution doit interrompre le plan courant et demander au planificateur d'en générer un nouveau afin d'obtenir un plan conforme à la nouvelle configuration de l'environnement. Le chapitre 4 introduit plus en détail le fonctionnement d'un planificateur de tâches.

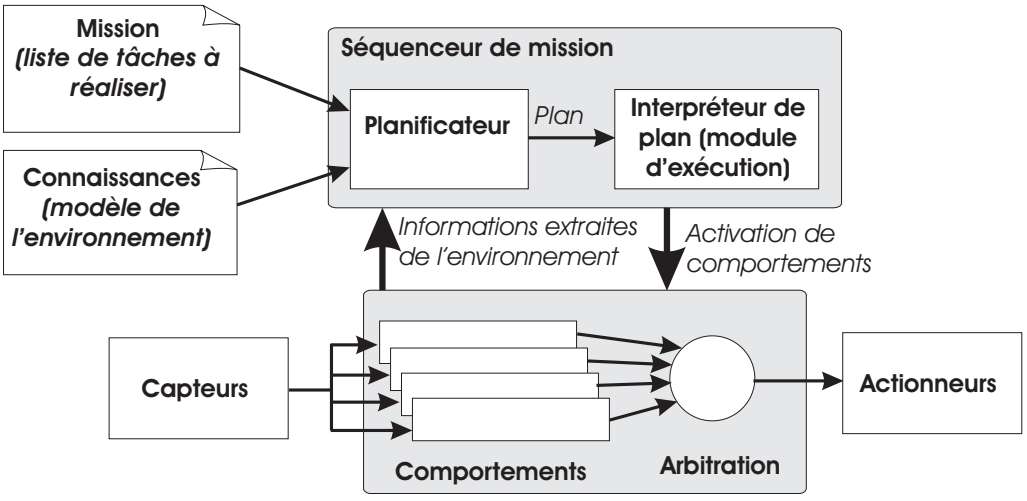


Figure 3.3 – Séquenceur de mission avec un planificateur de tâches.

CHAPITRE 4

Planification de tâches

Le problème de planification en intelligence artificielle consiste à déterminer quelles sont les actions à exécuter, et ce, dans quel ordre, pour accomplir un but donné. Comme illustré à la figure 4.1, un planificateur reçoit généralement en entrée un domaine, soit le modèle des actions et du monde (environnement)[1], un état initial représentant l'état actuel du système et un but à atteindre. En sortie, le planificateur génère un plan. Ce plan peut prendre différentes formes, la plus commune étant une séquence d'actions ou de tâches primitives, les deux notions étant considérées équivalentes ici. Intuitivement, une action ou une tâche primitive est une activité que le robot sait déjà comment accomplir de manière innée, par l'agencement d'un ou plusieurs modules comportementaux. Dans ce chapitre, nous introduisons les techniques de planification les plus répandues.

4.1 Hypothèses courantes

Avant de modéliser un domaine d'application pour un planificateur, on fait généralement certaines hypothèses afin de simplifier le problème [36]. Les plus courantes sont les suivantes :

[1] Dans la littérature, le terme monde (*World*) est généralement employé en planification en intelligence artificielle pour désigner une simplification (un modèle) de ce qu'on appelle environnement en robotique mobile.

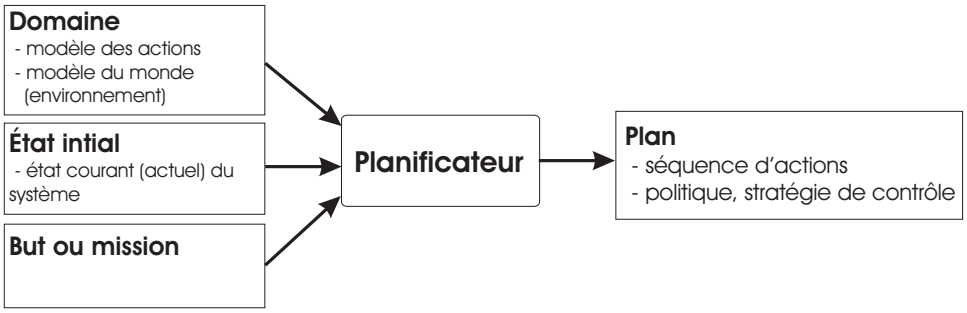


Figure 4.1 – Modèle de planification.

- **H1 Observabilité totale.** À tout instant, tout ce qui est requis de connaître sur l'environnement est connu. Par exemple, dans le scénario de livraison de colis, les positions du robot et des boîtes sont connues à tout instant.
- **H2 Déterministe.** Le résultat d'une action est unique et constant. En d'autres mots, on présume que l'exécution se déroule dans un monde parfait où les actions ne peuvent pas échouer et que leurs effets sont totalement prédéterminés.
- **H3 Monde statique.** Il n'y a pas d'événements externes qui modifient le monde (l'environnement). Par exemple, dans le domaine de la livraison de colis, seul le robot pour lequel on planifie ses actions peut déplacer les blocs.
- **H4 Plans séquentiels.** Les plans sont des séquences d'actions où chaque action s'exécute l'une à la suite de l'autre. Il n'y a pas d'actions simultanées.
- **H5 Durée implicite.** Les actions n'ont pas de durée, elles sont considérées comme étant instantanées au moment de la planification.

En fonction des combinaisons d'hypothèses retenues, plusieurs familles d'algorithmes de planification sont obtenues : classique, déterministe, non déterministe, probabiliste, ainsi de suite. Les frontières entre ces familles ne sont pas toujours nettes et certaines familles se chevauchent.

4.2 Planification classique

La planification classique est le type de planification la plus contraignante, car elle utilise toutes les hypothèses précédemment énumérées. Dans la littérature, la planification clas-

sique est souvent associée à STRIPS (*Standford Research Institue Problem Solver*) [15], un des premiers planificateurs à être développés (1972).

Domaine de planification

Un **domaine** en planification classique est la modélisation d'une application donnée pour un planificateur. Intuitivement, un domaine modélise toutes les transitions (ou actions) que peut effectuer le système pour lequel on planifie. Dans notre cas, le système pour lequel on planifie est un robot.

Formellement, un domaine de planification est un triplet $D = (\Sigma, M, O)$ où Σ est l'alphabet des symboles du domaine, M est la représentation du monde (relations rigides) et O est un ensemble d'opérateurs. Chaque **symbole** $\sigma \in \Sigma$ représente un objet de l'application. Par exemple, dans le monde de la livraison de colis (figure 2.1), $p1$ est un symbole représentant la pièce en haut à gauche, et $o1$ représente l'objet numéro un.

États et relations rigides

Un **état** est une configuration du domaine. La façon classique pour représenter un état est l'utilisation de la logique du premier ordre. On utilise des prédicats et des symboles pour exprimer des propriétés et des faits sur des objets ou des relations entre divers objets. Dans le domaine de la livraison de colis, on peut avoir le prédicat *ObjetDans(o,x)* exprimant qu'un objet o se situe dans une pièce x. Un état s est alors un ensemble fini de prédicats. Par exemple, l'état courant du domaine illustré à la figure 2.1 est : *s={RobotDans(c1), PinceLibre(), ObjetDans(o1, p1), ObjetDans(o2, p3), ObjetDans(o3, p4)}*. Dans le cas présent, le planificateur travaille avec des états décrits par des prédicats complètement instanciés, c'est-à-dire que les arguments des prédicats ne contiennent pas de variable. Donc, ces prédicats sont des propositions.

Pour des raisons d'efficacité, les **relations rigides** du domaine, c'est-à-dire les relations qui sont permanentes, ne sont pas explicitement représentées dans un état. Par exemple, dans le cas de livraison de colis, un état contient implicitement les relations d'adjacences

des pièces. Ainsi, pour la figure 2.1, l'état inclut implicitement l'ensemble des relations rigides *M={Adjacent(p1, c1), Adjacent(p2, c1), Adjacent(p3, c2), Adjacent(p4, c2), Adjacent(c1, c2), ...}*. Puisque la relation géométrique d'adjacence est symétrique, pour toute relation $Adjacent(i, j)$, on a aussi la relation $Adjacent(j, i)$.

Opérateurs et actions

Un **opérateur** est un patron d'actions représentant une opération possible à l'intérieur d'un domaine donné. Un opérateur *o* est défini par le quadruplet : *o=(nom(o), params(o), precond(o), effets(o))* :

- $nom(o)$ est le nom de l'opérateur.
- $params(o)$ est une liste de variables en paramètres.
- $precond(o)$ est un ensemble de préconditions représentées par des littéraux[2].
- $effets(o)$ est un ensemble d'effets[3] représentés par des littéraux.

Dans le contexte de la livraison de colis, les opérateurs suivants sont définis :

```
AllerÀ(a, b)    ;; déplace le robot d'une pièce a à une pièce b
  precond : {Adjacent(a, b), RobotDans(a)}
  effets  : {!RobotDans(a), RobotDans(b)}

Prendre(o, p)    ;; prendre l'objet o dans la pièce p
  precond : {RobotDans(p), ObjetDans(o, p), PinceLibre()}
  effets  : {!PinceLibre(), RobotPossède(o), !ObjetDans(o, p)}

Déposer(o, p)    ;; déposer l'objet o dans la pièce p
  precond : {RobotDans(p), RobotPossède(o)}
  effets  : {PinceLibre(), !RobotPossède(o), ObjetDans(o, p)}
```

[2]Un littéral est un prédicat ou la négation d'un prédicat. Le symbole « ! » est utilisé pour désigner le connecteur logique de négation.

[3]Les effets négatifs, marqués par la négation d'un prédicat, suppriment un littéral dans l'état successeur.

Une **action** est une instance d'un opérateur où les variables (paramètres) ont été substituées par des symboles du domaine. Par exemple, l'action *AllerÀ(c1, p1)* est une action valide pour le domaine de la livraison de colis où l'on a utilisé la substitution $a = c1, b = p1$, $c1$ et $p1$ étant des lieux bien spécifiques.

Une action a est dite **applicable** dans l'état s si toutes ses préconditions sont satisfaites dans l'état s. Par exemple, supposons l'état *s={RobotDans(c1), PinceLibre(), ObjetDans(o1, p1), ObjetDans(o2, p3)*, et l'action *a=AllerÀ(c1, p1)*. Les préconditions de l'action a sont *precond(a)={Adjacent(c1, p1), RobotDans(c1)}*. Puisque la relation rigide *Adjacent(c1, p1)* $\in M$ est implicitement définie dans l'état s et que le littéral *RobotDans(c1)* est inclus dans l'état s, alors l'action a est applicable dans l'état s.

L'application de l'action a_i dans l'état s_i résulte en un état **successeur** s_{i+1}, obtenu de s_i en supprimant les effets négatifs de a_i et en ajoutant les effets positifs de a_i. Formellement, on peut écrire cette relation à l'aide d'une fonction de transition $\gamma : s_{i+1} = \gamma(a, s_i)$. Intuitivement, si le modèle d'actions (opérateurs) est correct et que l'exécution se déroule comme prévu, le fait qu'une action a_i est applicable à l'état s_i signifie qu'elle mènera à l'état s_{i+1} lors de son exécution.

Problèmes et plans

Un **problème** en planification classique est défini par un triplet $P = (D, s_0, g)$ où D est le domaine, s_0 est l'état initial et g est le but à atteindre. Le **but** est spécifié par un ensemble de littéraux positifs. Par exemple, toujours pour le domaine de livraison de colis illustré à la figure 2.1, l'état initial est *s_0={RobotDans(c1), PinceLibre(), ObjetDans(o1, p1), ObjetDans(o2, p3), ObjetDans(o3, p4)}*, et le but pourrait être *g={ObjetDans(o1, p4)}*.

Une **solution** à un problème de planification classique est appelée **plan** et est une séquence d'actions $\pi = (a_1, a_2, ..., a_n)$. De plus, un plan est **valide** si toutes ses actions sont applicables l'une à la suite de l'autre et mènent au but donné. Dans le cas contraire, si un plan ne mène pas au but ou n'est pas applicable, alors ce plan est **invalide**.

4.3 Algorithmes de planification déterministe

De nombreuses techniques de planification déterministe existent. Dans cette section, les plus connues sont présentées.

4.3.1 Recherche à chaînage avant dans l'espace d'états

Une technique simple pour résoudre un problème de planification déterministe est l'utilisation d'un algorithme de recherche à chaînage avant dans l'espace des états du domaine. La figure 4.2 illustre un exemple d'une telle recherche dans le domaine de la livraison de colis. En partant de l'état initial, il suffit d'appliquer toutes les actions applicables. Chaque action est appliquée et mène à un nouvel état. De façon récursive, cette opération est répétée à partir des nouveaux états trouvés, et ce, jusqu'à ce qu'un état satisfaisant au but donné soit atteint. Dans le cas où aucune solution n'existe, alors tout l'espace d'états atteignable est exploré de façon exhaustive.

Pour parcourir l'espace d'états, l'algorithme A* [23] (recherche heuristique) peut être utilisé. Une fonction heuristique souvent utilisée, estimant le coût restant pour atteindre le but g à partir de l'état s, est la fonction suivante : $h(s) = |g - s|$. Cette fonction compte le nombre de littéraux restant à obtenir pour atteindre le but donné. Bien que cette fonction soit admissible, elle n'est pas très efficace puisque sa valeur est généralement beaucoup plus petite que le coût restant. Pour A*, une fonction heuristique est dite admissible si elle ne surestime pas le coût restant. Avec une telle fonction, A* trouve une solution optimale. Parmi les fonctions heuristiques admissibles, celle qui sous-estime le moins le coût restant est la plus efficace puisqu'elle permet d'obtenir plus rapidement une solution en explorant un plus petit espace d'états. Par exemple, dans l'exemple de la figure 4.2, tous les états ont une valeur d'heuristique égale à un (1) sauf l'état final où la valeur est de zéro (0). Pour cette raison, sur cet exemple-ci, l'algorithme A* se comporte comme une recherche en largeur puisque l'heuristique ne permet pas de trier efficacement les états en fonction de leur proximité avec le but. Cependant, il existe d'autres fonctions heuristiques plus efficaces (voir section 4.5).

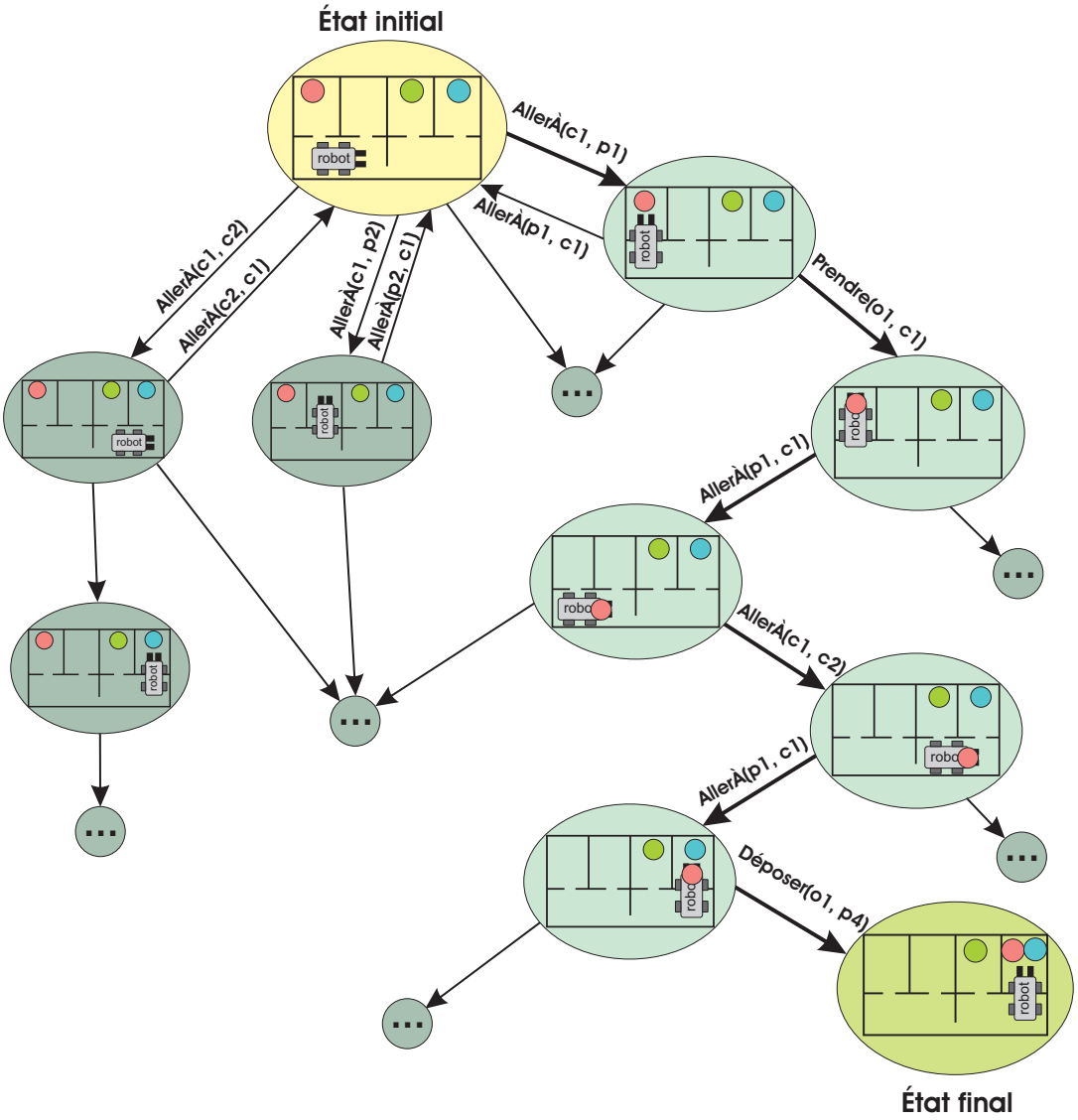


Figure 4.2 – Recherche à chaînage avant.

Sur le même principe que la recherche vers l'avant, on peut également effectuer une recherche à chaînage arrière dans l'espace d'états. Au lieu de partir de l'état initial, la recherche débute du but et cherche toutes les actions permettant d'atteindre ce but. Pour chaque action possible, un état prédécesseur est trouvé. Ce processus se répète jusqu'à ce que l'état initial du problème donné soit trouvé. Théoriquement, une recherche vers l'arrière a la même efficacité qu'une recherche vers l'avant. Par contre, il peut arriver en pratique que sur certains domaines l'une des deux recherches soit meilleure que l'autre.

L'approche proposée dans ce mémoire explore un espace d'états avec un chaînage avant.

4.3.2 Recherche dans l'espace de plans

Une autre approche plus élaborée est de faire une recherche dans l'espace de plans. Les nœuds de cet espace sont des spécifications partielles de plans et les arcs sont des opérations de raffinement pour résoudre un but n'étant pas encore satisfait ou pour résoudre une contrainte. Une solution obtenue par ce type de recherche n'est plus un plan sous forme de séquence d'actions, mais un **plan partiellement ordonné** noté $\pi = (A, \prec, B, L)$ où $A = \{a_1, a_2, ..., a_n\}$ est un ensemble d'actions et $\prec$ est un ensemble d'ordres partiels de la forme $a_i \prec a_j$, indiquant que l'action a_i doit précéder l'action a_j. Pour plus de détails, le lecteur est référé à [36].

4.3.3 Recherche dans l'espace d'un graphe de planification

Une technique encore plus sophistiquée est l'approche GraphPlan [9]. Elle consiste à faire une recherche dans un **graphe de planification**. La construction du graphe de planification se fait niveau par niveau, où chaque niveau n_i contient l'ensemble des prédicats accessibles après l'application d'au moins i actions. Le premier niveau, soit n_0, contient un nœud pour chaque prédicat de l'état initial. Un niveau n_{i+1} est formé par l'union des prédicats du niveau n_i et de tous les prédicats obtenus par l'application de toutes les actions applicables au niveau n_i. Entre deux niveaux consécutifs n_i et n_{i+1}, des arêtes étiquetées d'une action indiquent comment les prédicats d'un niveau n_{i+1} dépendent des prédicats du niveau précédent n_i. Aussi, à l'intérieur d'un même niveau, d'autres arêtes indiquent les prédicats étant mutuellement exclusifs. Enfin, une fois le graphe de planification construit jusqu'à un certain niveau, GraphPlan fait une recherche dans ce dernier pour trouver un plan.

4.4 Stratégies de contrôle de recherche

Afin d'améliorer les performances de recherche de plan, des stratégies de contrôle de recherche peuvent être utilisées. Les deux approches les plus populaires sont l'utilisation

de formules de logique temporelle et l'utilisation de modèles de décomposition de tâches.

Lors d'une recherche dans un espace d'états, beaucoup de branches explorées ne sont pas utiles. Par exemple, dans l'exemple précédent où il faut livrer l'objet $o1$ de la pièce $p1$ à $p4$, jamais il n'est utile d'aller dans la pièce $p2$. Or, durant une recherche vers l'avant, il est fort possible que l'algorithme explore la branche d'états générée par l'action *AllerÀ(c1, p2)*.

En fait, les seules connaissances dont dispose l'algorithme de planification sont les transitions possibles déduites des opérateurs décrivant le domaine. Dans l'exemple précédent, de façon intuitive, un humain saurait que la pièce $p2$ n'a rien à voir avec le problème donné. Ce genre de connaissances sont dites des **connaissances stratégiques** du domaine. Elles n'ont rien à voir avec les heuristiques. Ces dernières ordonnent simplement les états par mérite en terme de proximité à un état final satisfaisant le but. Par contre, une connaissance stratégique suggère des stratégies d'un expert pour trouver efficacement un plan dans un domaine donné. Ainsi, les connaissances stratégiques limitent l'exploration de l'espace d'états en indiquant implicitement les mauvais chemins qu'il faudrait éviter d'explorer ou, à l'inverse, les chemins les plus prometteurs.

Une façon de spécifier ce genre de connaissances est par l'utilisation de **formules de logique temporelle (LTL)** pour décrire l'évolution des états à visiter pour aboutir à une solution. Dans le cas de l'exemple précédent, il serait possible de spécifier qu'il faut aller dans la pièce $p1$, et qu'ensuite, il faut toujours posséder l'objet $p1$ jusqu'à temps d'être dans la pièce $p4$. Cette approche a été exploitée pour la première fois dans le planificateur TLPlan [5].

4.4.1 Planification HTN

Une autre approche pour spécifier des connaissances stratégiques est d'utiliser des réseaux hiérarchiques de tâches, ou en anglais, *Hierarchical Task Network* (HTN) [36]. La **planification HTN** est basée sur la notion de tâches. Au lieu de spécifier un but sous la forme d'une conjonction de clauses, un but est défini sous la forme d'une liste de tâches de haut

niveau. Un planificateur HTN fait alors une recherche dans un espace de décomposition de tâches afin de réduire la mission en une liste de tâches primitives. Pour chaque tâche de haut niveau, un modèle de décomposition dicte comment réduire cette tâche en sous-tâches. Il peut y avoir plusieurs façons de décomposer une tâche. Ces connaissances sur les différentes décompositions possibles relèvent de stratégies habituellement possédées par un expert.

Plus formellement, un domaine de planification HTN est défini par le quadruplet $D = (\Sigma, M, O, N)$ où Σ, M, et O sont, tout comme pour la planification classique, respectivement l'alphabet des symboles du domaine, le monde et l'ensemble des opérateurs. Enfin, N est la définition du réseau de décomposition de tâches, soit un ensemble de méthodes de décomposition. Une **méthode de décomposition** (sous-réseau) $n \in N$ est défini comme suit : $n = (tache(n), precond(n), soustaches(n), \prec)$ où $tache(n)$ est une tâche de haut niveau, $precond(n)$ est un ensemble de préconditions, $soustaches(n)$ est l'ensemble des sous-tâches et $\prec$ est un ensemble d'ordres partiels sur l'ensemble des sous-tâches. Une sous-tâche peut être une tâche primitive ou une autre tâche de haut niveau.

La figure 4.3 montre un exemple de réseau de décomposition de tâches pour le domaine de la livraison de colis. Dans le haut de la figure, il y a une méthode de décomposition pour la tâche de haut niveau *LivrerObjet(o, orig, dest)* qui consiste à livrer l'objet *o* de à l'endroit *orig* à l'endroit *dest*. Elle se décompose en les quatre sous-tâches suivantes : *SeRendreÀ(orig)*, *Prendre(o, orig)*, *SeRendreÀdest* et *Déposer(o, dest)*. La flèche horizontale, qui traverse les quatre flèches liant la tâche de haut niveau aux tâches de niveau inférieur, indique un ordre chronologique entre les sous-tâches. Enfin, les deux autres méthodes de décomposition décrivent comment une tâche *SeRendreÀ* peut être décomposée en une suite de tâches primitives *AllerÀ* donnant un chemin d'endroits adjacents.

La famille des planificateurs SHOP (*Simple Hierarchical Ordered Planner*), soit JSHOP, SHOP [35] et SHOP2 [37] sont des exemples de planificateurs basés sur la planification HTN. Ils combinent une exploration à chaînage avant dans un espace d'états (section 4.3.1) avec une décomposition de tâches. Notre algorithme décrit plus loin est inspiré de SHOP2.

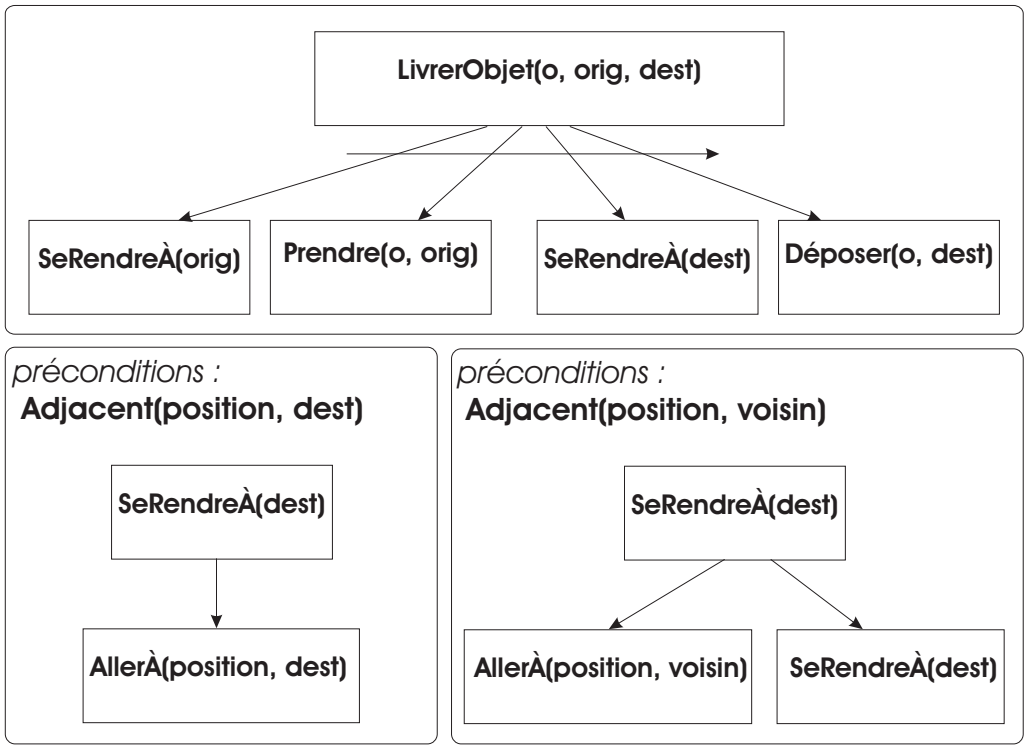


Figure 4.3 – Exemple de spécification d'un réseau de tâches en HTN.

4.5 Planification heuristique

Nous avons déjà abordé le rôle des heuristiques en mentionnant A* pour l'exploration d'un espace d'états avec chaînage avant. Les techniques les plus avancées vont au-delà de l'approche A* de base.

À la section 4.3.1, il est expliqué qu'il est possible d'utiliser comme fonction heuristique le compte de prédicats manquants dans un état pour atteindre le but. Une meilleure approche consiste à utiliser une version allégée des graphes de planification (*relaxed Graph-Plan*) afin d'estimer le nombre d'actions requises pour atteindre le but. En d'autres mots, au cours de la recherche, pour chaque état visité, un graphe de planification simplifié est construit temporairement, mais sans prendre en compte les prédicats mutuellement exclusifs et les effets négatifs. Lorsqu'un niveau contient tous les prédicats du but recherché, le nombre d'actions minimales requises pour générer ces prédicats est compté. Puisque les prédicats mutuellement exclusifs et les effets négatifs sont ignorés, cette fonction heuristique ne surestime jamais le coût restant ; elle est donc admissible. Cette heuristique

produit des valeurs plus près du coût restant que l'heuristique classique comptant les prédicats manquants dans l'état. Elle est donc plus efficace et permet de réduire considérablement le nombre d'états visités au cours de la recherche. Grâce à cette technique, le planificateur Fast-Forward (FF) [25] a obtenu la meilleure performance dans la catégorie planification classique (STRIPS)[4] à la 3^e compétition IPC (*International Planning Competition*) lors de la conférence ICAPS (*International Conference on Autmated Planning and Scheduling*) en 2002.

4.6 Planification concurrente

Parfois, il peut être nécessaire de générer des plans dans lesquels plusieurs actions peuvent s'exécuter simultanément. Il s'agit de planification concurrente. Pour ce faire, l'hypothèse **H4** doit être levée et il faut créer un nouveau modèle de planification permettant de gérer des actions avec des durées variables.

Une technique pour résoudre ce type de problème consiste à ajouter une variable temporelle directement dans la représentation interne des états et d'ajouter une durée dans la description des opérateurs [4]. De plus, les préconditions et les effets sont temporels : on peut les spécifier pour le début, pour la fin ou pour toute la durée de l'action. Le planificateur SAPA [13] utilise cette approche afin de planifier des actions simultanées.

4.7 Planification avec métriques

Pour beaucoup de domaines, certaines caractéristiques de l'environnement, comme des ressources, doivent être modélisées à l'aide de variables numériques. Par exemple, le niveau des batteries d'un robot peut se représenter à l'aide d'une variable réelle. Le temps peut aussi être considéré comme une ressource particulière ne pouvant qu'être incrémentée. La spécification des opérateurs peut définir des préconditions sur ces variables numériques ainsi que des effets pouvant les incrémenter ou les décrémenter. Par exemple,

[4]http ://planning.cis.strath.ac.uk/competition/

un opérateur de déplacement pour un robot mobile peut avoir la précondition d'avoir suffisamment d'énergie et l'effet de décrémenter la variable du niveau d'énergie, tandis qu'un opérateur de recharge peut avoir comme effet d'incrémenter cette même variable.

En planification classique, l'efficacité d'un plan se mesure en terme du nombre d'actions qu'il contient ; un plan est optimal s'il n'existe pas d'autres plans avec moins d'actions. En d'autres mots, les actions ont toutes le même coût. La planification avec métriques va plus loin : elle permet d'avoir des actions avec des coûts variables. Deux approches équivalentes peuvent être utilisées pour estimer le coût d'un plan : additionner les coûts individuels des actions, ou évaluer une expression mathématique basée sur les valeurs numériques de l'état final. On peut par exemple demander à un planificateur de trouver un plan minimisant un certain critère comme l'énergie consommée, la durée totale ou même une combinaison linéaire de ces deux critères.

Intégrer la gestion de métriques dans un planificateur à chaînage avant est plutôt simple : il ne s'agit que d'ajouter des variables numériques dans la représentation des états. Par contre, pour être efficace, le planificateur doit tenir compte des ressources lors de son raisonnement. Une technique efficace consiste à étendre le concept de la planification heuristique basée sur les graphes de planification en considérant les effets numériques comme suit : les incrémentations sont considérées comme des effets positifs et les décrémentations sont ignorées comme ce fut le cas pour les effets négatifs. Basé sur cette approche, le planificateur Metric-FF [24], qui est une extension métrique à FF [25], a terminé en première position pour les problèmes de type numérique à la 3^e compétition IPC. Les planificateurs LPG [20] et SAPA [13] exploitent aussi cette même approche.

4.8 Planification non déterministe

La planification non déterministe consiste à générer des plans conditionnels prévoyants des alternatives pour des cas non déterministes, c'est-à-dire connus à l'avance, mais avec un certain degré d'incertitude. La planification classique fait l'hypothèse que le domaine est complètement déterministe. En d'autres mots, on présume que les plans s'exécuteront

dans un monde idéal où l'échec n'existe pas. Par contre, la réalité est tout autre et des situations imprévues peuvent survenir. Par exemple, dans le domaine de la livraison de colis, il est possible que le robot puisse se tromper de pièce suite à l'exécution d'une action de déplacement. Ainsi, le résultat de l'exécution d'une action n'a plus qu'un seul état successeur, mais un ensemble d'états successeurs possibles où chacun peut avoir une certaine probabilité.

Ce type de planification est d'une complexité de plusieurs ordres de grandeur supérieurs à la planification déterministe. Comme notre approche n'est pas basée cette méthode, ce type de planification n'est pas exploré dans ce mémoire. En fait, nous expliquons comment, avec des techniques simples [22], il est possible d'utiliser un planificateur déterministe pour générer et exécuter des plans dans des environnements dynamiques et non déterministes. En d'autres mots, au lieu de faire face au non-déterminisme durant la planification, le robot l'ignore et planifie pour le meilleur des mondes. Cependant, le robot surveille le déroulement de l'exécution pour détecter les situations divergentes du monde parfait. Dans ce cas, le planificateur corrige le plan et le robot continue à nouveau.

CHAPITRE 5

Planification de chemin

À la section 1.1.3, nous avons introduit le module de navigation. Dans ce module, on a un planificateur de chemin, un planificateur différent du planificateur de tâches. Lorsque le planificateur de tâches génère une action de déplacement du type *AllerÀ*, le planificateur de chemin est appelé pour trouver un chemin permettant de déplacer le robot de sa position courante jusqu'à la destination, et ce, de façon optimale et sécuritaire.

5.1 Carte de l'environnement

Afin de trouver un chemin, un planificateur de chemin a besoin d'une carte de l'environnement dans lequel le robot évolue. Habituellement, cette carte est représentée à l'aide d'une **grille d'occupation** obtenue en discrétisant l'environnement en cellules. Une cellule est soit libre ou soit occupée par un ou des obstacles, comme dans l'exemple de la figure 5.1. À partir d'une cellule, un robot peut atteindre une cellule voisine, si elle est libre. Il existe deux définitions de relation de voisinage : 4 voisins et 8 voisins. Dans la première, le robot peut se déplacer dans quatre directions cardinales, soit nord, sud, est et ouest ; dans la deuxième les quatre directions obliques sont aussi permises.

Le modèle de la grille d'occupation peut être amélioré en y ajoutant certains attributs. On peut par exemple attribuer des coûts aux cellules ou même des probabilités de présence

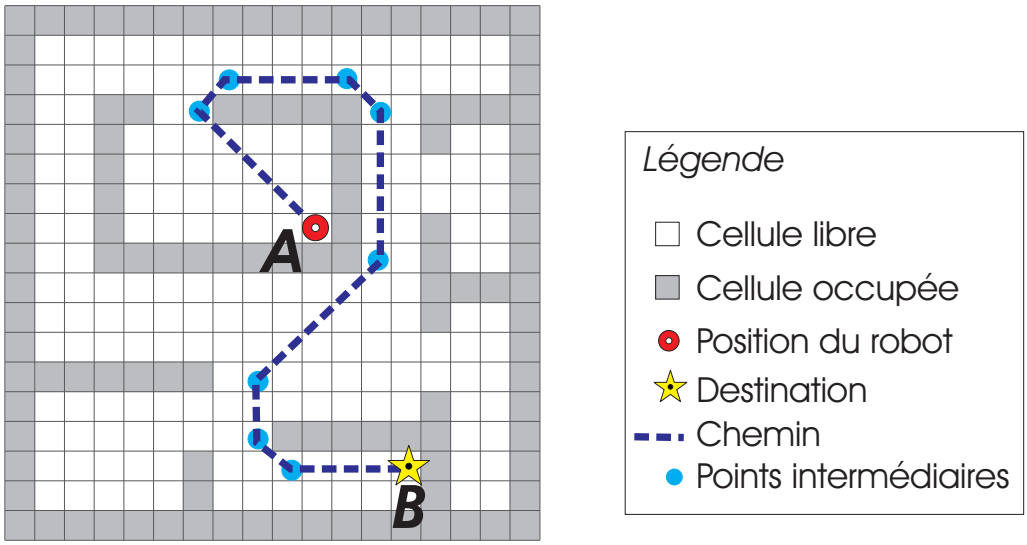


Figure 5.1 – Exemple de chemin dans une grille d'occupation.

d'obstacles. D'autres représentations de cartes peuvent aussi être utilisées.

5.2 Algorithmes de planification de chemin

L'algorithme A* [23] peut être utilisé pour trouver le plus court chemin afin de se rendre à destination. Comme fonction heuristique, on utilise la distance euclidienne (aussi appelée distance à vol d'oiseau) dans le cas d'une relation à 8 voisins ou la distance de Manhattan dans le cas d'une relation à 4 voisins. Une solution de chemin prend la forme d'une liste de cellules voisines, comme celles sous les traits à la figure 5.1. Souvent, on simplifie la polyligne formée par le chemin en conservant un nombre minimal de points intermédiaires. Le navigateur utilise ces points pour configurer le comportement de suivi de chemin.

En présence d'obstacles dynamiques, on peut utiliser un algorithme comme D* ou Anytime Dynamic A* [28]. Ces algorithmes ont la capacité de réutiliser les informations des recherches précédentes afin de réduire les efforts nécessaires pour les prochaines recherches. Il existe aussi d'autres techniques, basées sur les processus décisionnels de Markov (MDP) [26], permettant de trouver des chemins fiables et optimaux en considérant l'incertitude de l'environnement liée à l'imprécision des capteurs et à des probabilités de présences d'obstacles dans l'environnement.

CHAPITRE 6

Évaluation de planificateurs

Le domaine d'application utilisé pour faire l'évaluation de différents planificateurs est celui de la conférence simulée telle que présentée à la section 2.2. Ici, seule la génération de plans statiques est évaluée. L'aspect dynamique du domaine n'est aucunement considéré puisque le premier objectif est de trouver un bon algorithme de planification.

Pour ce domaine, des tâches d'actions et de déplacements de haut niveau doivent être planifiées. Le planificateur n'a pas à générer d'itinéraires complets en planifiant des points intermédiaires : un planificateur de chemin, utilisé en complémentarité avec le planificateur de tâches lors de l'exécution des missions, réalise ce travail. Ainsi, contrairement au domaine de la livraison de colis, une action *AllerÀ(a, b)* est valide même si les endroits a et b ne sont pas adjacents. À l'exécution, le planificateur de chemin a la responsabilité de trouver le chemin optimal pour se rendre du point a au point b.

Les types de tâches à planifier sont :

- Livrer un message d'un point A à un point B.
- Faire une présentation à un lieu P, à une période de temps donnée.
- Présenter une affiche à un lieu P. Cette tâche consiste à installer une affiche à l'intérieur d'une fenêtre de temps avant le début de la session d'affichage et à la retirer après la fin de la session.
- Photographier une affiche à un stand donné lors d'une session d'affichage donnée.

6.1 Planificateurs évalués

Le choix des planificateurs évalués est basé sur plusieurs critères. En premier lieu, les planificateurs étudiés doivent permettre de gérer le temps. Si l'option temporelle n'est pas disponible, il est également possible d'utiliser un planificateur numérique en définissant nous-mêmes une variable temporelle dans la représentation de l'état. En second lieu, nous avons retenu les planificateurs réputés pour leurs performances. Comme indicateur de performance, nous nous sommes basés sur les résultats des compétitions IPC (International Planning Competition), ayant lieu lors des conférences annuelles ICAPS. Enfin, le dernier critère est la disponibilité des planificateurs, puisque certains auteurs préfèrent ne pas distribuer le code source ou binaire de leur planificateur.

Les planificateurs évalués sont Metric-FF [24], SAPA [13], LPG [20] et SHOP2 [37]. Les deux premiers sont des planificateurs compatibles avec le langage PDDL version 2.1 niveau 2 (Problem Definition Description Language) [19] alors que SAPA supporte le niveau 5 [19]. Le niveau 2 prend en charge les problèmes numériques alors que le niveau 5 supporte en plus des actions ayant des durées variables. Quant à SHOP2, ce dernier utilise son propre formalisme de définition de problème.

6.2 Tests et résultats

Une spécification du domaine a été codée pour chaque type de planificateur. Les codes PDDL utilisés pour le domaine de la conférence simulée sont disponibles à l'annexe A.

Cinq séries de tests ont été générées aléatoirement. Les quatre premières séries comportent des tâches d'un seul type et la dernière mélange plusieurs types à la fois. Les types de tâches sont : 1)livrer un message, 2)présenter une affiche, 3)réaliser une présentation et 4)photographier une affiche. Pour chaque série, nous avons généré des tests de différentes tailles, allant d'une (1) à dix (10) tâches. Pour chaque série et pour chaque taille de problème, dix (10) tests ont été générés, faisant un grand total de 500 tests. Enfin, pour chacun de ces tests, des fichiers de tests équivalents ont été générés pour chacun des

planificateurs évalués.

Lors des premières expérimentations, nous avons remarqué qu'un bon nombre de planificateurs ne parvenaient pas à résoudre efficacement des problèmes nécessitant au moins une recharge des batteries. Pour cette raison, nous avons pris soin de ne pas générer de tels problèmes dans les séries de tests aléatoires.

Pour quantifier les performances des planificateurs, nous avons mesuré les temps requis pour planifier chaque test. Certains tests nécessitant trop de temps pour leur résolution, nous avons donc limité le temps d'exécution des planificateurs à 5 minutes pour chaque test. Lorsqu'un planificateur est interrompu après ce délai maximum, c'est ce temps qui est utilisé lors des analyses. Le choix de cette durée est raisonnable puisqu'en robotique mobile, on s'attend à avoir des délais de réponses nettement sous la barre de la minute.

La figure 6.1 (page 42) montre les résultats des planificateurs pour chaque séries de tests. L'axe des x représente la taille des problèmes et l'axe des y représente la durée moyenne de planification pour chaque planificateur.

De façon générale, les performances des planificateurs varient beaucoup en fonction du type de tâches à planifier. Par exemple, le planificateur LPG démontre d'excellents résultats pour la livraison de messages, alors qu'il obtient parmi les pires résultats pour tous les autres cas.

- **Metric-FF.** Le planificateur Metric-FF obtient de bons résultats dans la livraison de colis. Ceci s'explique bien puisque ce type de tâche ne possède pas de contraintes de temps. Dans ce cas, l'heuristique basée sur les graphes de planifications simplifiés [25] est très efficace.

  Les problèmes avec lesquels Metric-FF a le plus de difficulté sont ceux contenant des tâches soumises à des contraintes de temps. Le temps étant représenté à l'aide d'une variable numérique, cette dernière est prise en compte lors du calcul de la valeur heuristique pour chaque état. Lors de ce calcul, puisque Metric-FF ignore les effets négatifs et les actions mutuellement exclusives dans la génération du graphe de planification, Metric-FF n'arrive pas à départager l'ordre des actions à l'aide des valeurs d'heuris-

tique. Ainsi, le planificateur doit explorer un grand nombre d'états avant de trouver une solution valide.

- **LPG.** Puisque le planificateur LPG utilise une stratégie semblable à Metric-FF, les mêmes hypothèses peuvent être retenues pour expliquer ses performances. De plus, utilisant des méthodes stochastiques, LPG ne retourne pas nécessairement la même solution pour deux résolutions d'un même problème. Cela pourrait causer des oscillations non désirées si ce planificateur était intégré dans un robot.
- **SAPA.** SAPA est un planificateur efficace pour la planification concurrente. Par contre, tout comme Metric-FF et LPG, le planificateur SAPA est mal adapté pour l'ordonnancement de tâches et la planification sous contraintes de temps (« scheduling »).
- **SHOP2.** Premièrement, la façon dont SHOP2 détecte et gère les relations rigides du domaine peut probablement expliquer pourquoi SHOP2 n'a pas très bien performé. Par exemple, en utilisant la commande *show state*, on s'aperçoit que SHOP2 semble conserver la volumineuse table des distances dans tous les états générés. Puisque la copie de ces données d'un état à l'autre nécessite beaucoup de temps, cela explique en partie les faibles performances de SHOP2.

  Quant aux tests avec des tâches avec contraintes de temps, puisque SHOP2 n'analyse pas ces contraintes pour ajouter des ordres partiels sur les tâches, le planificateur doit donc essayer plusieurs façons d'ordonnancer les tâches avant de trouver une solution. Par exemple, si on demande à SHOP2 de planifier n tâches de présentation, alors, dans le pire cas, il devra essayer $n!$ permutations pour trouver la solution. Si SHOP2 pouvait analyser les contraintes de temps, il créerait un ordre total sur la mission et il pourrait ainsi tester qu'une seule permutation de tâches pour trouver la solution.

6.3 Limitations liées au langage PDDL

Pendant l'évaluation des planificateurs, certaines difficultés ont été observées au niveau du langage PDDL. Entre autres, ce langage est mal approprié pour la spécification de

missions sous forme de tâches puisque le but doit être encodé sous une expression logique basée sur des symboles et des prédicats. Par exemple, pour spécifier la tâche de photographier l'affiche A entre 9h30 et 10h30, il n'est pas possible de simplement écrire *Photographier(A, 32400, 36000)*[1] comme cela est possible de faire avec le planificateur SHOP2. Il n'est pas possible de le faire puisque le langage PDDL ne permet pas l'usage de valeurs numériques dans l'expression du but. Il faut plutôt encoder le but à l'aide d'un prédicat comme *PhotoPrise(A, I)*, déclarer un symbole I de type intervalle dans la définition du problème et l'associer à un intervalle de temps à l'aide de fonctions numériques comme : *(= (Debut I) 32400)* et *(= (Fin I) 36000)*. Cette façon de procéder est moins intuitive et plus complexe à traiter pour le planificateur.

De plus, il n'est pas possible de spécifier des contraintes facilement. Par exemple, pour deux tâches de livraison de messages à effectuer, il n'est pas facile de spécifier qu'une livraison doit être traitée avant l'autre. Pour ce faire, il faut aller jouer au niveau des opérateurs et ajouter des préconditions supplémentaires et dépendantes de ce cas spécial. Par contre, avec une approche basée sur le concept de tâches comme HTN, il est facile d'imposer cette contrainte en faisant simplement ajouter un ordre partiel entre les deux tâches, et ce, sans toucher à la spécification des opérateurs.

Les résultats des planificateurs évalués montrent qu'ils sont mal adaptés au contexte particulier du *AAAI Robot Challenge*. De plus, en raison des difficultés rencontrées avec l'usage du langage PDDL dans ce domaine précis, il est préférable d'envisager une approche HTN pour l'élaboration d'une solution de planification pour un robot dans le contexte du *AAAI Robot Challenge*.

[1] Les nombres 32400 et 36000 expriment les temps en secondes depuis minuit.

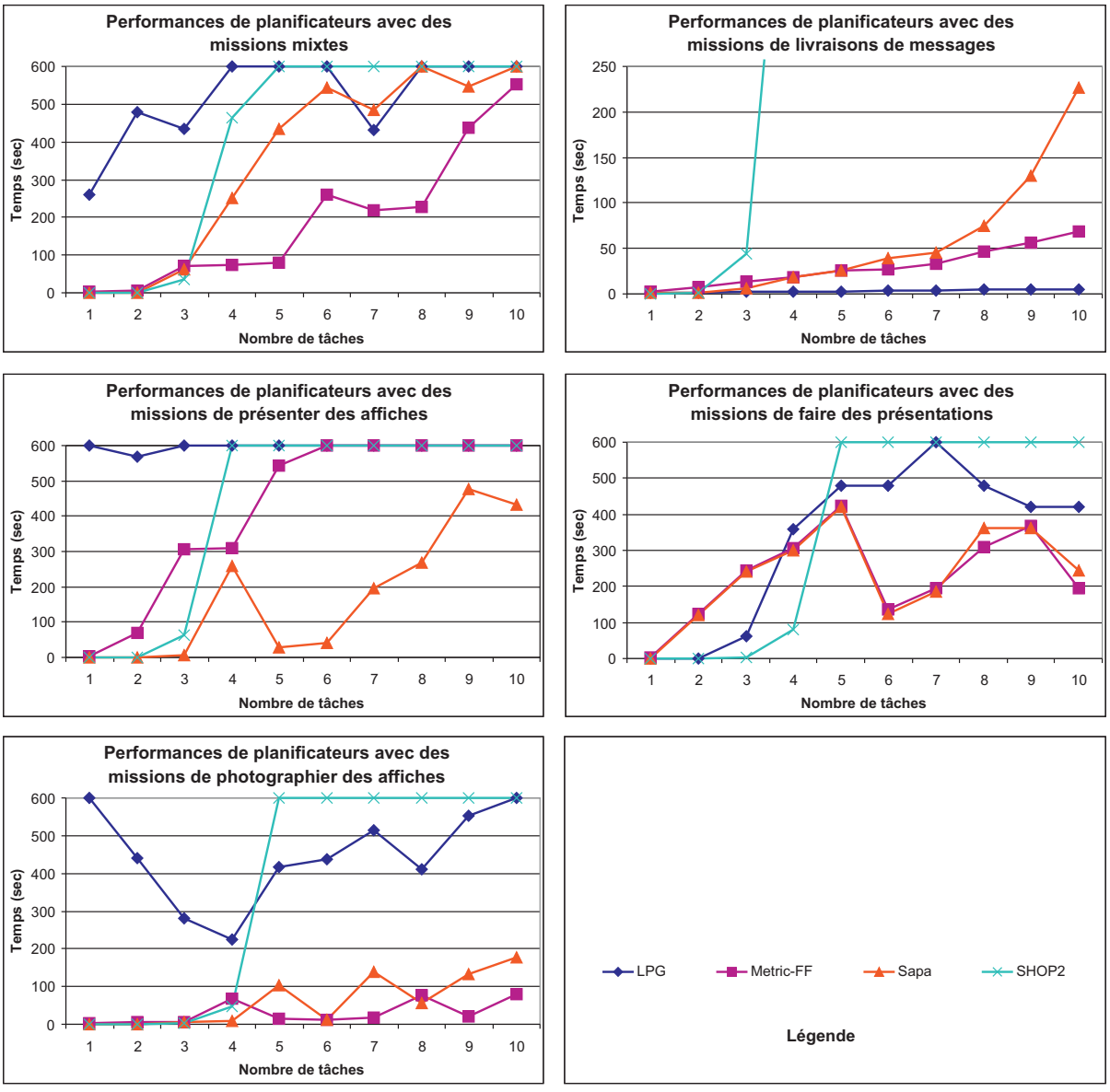


Figure 6.1 – Comparaison des planificateurs.

CHAPITRE 7

ConfPlan : un planificateur pour robot mobile autonome

Afin de rendre un robot plus autonome en lui permettant de planifier et de coordonner lui-même ses actions, ConfPlan [7], un nouveau planificateur, a été développé. Ce dernier est spécialement conçu pour gérer des missions du type du *AAAI Robot Challenge*. ConfPlan est capable de planifier des missions contenant des tâches avec des contraintes de temps rigides ou souples, de gérer l'énergie du robot en décidant des moments appropriés pour la recharge des batteries, et de gérer des contraintes entre les tâches (ex. : on doit être inscrit à la conférence avant d'assister à des présentations). L'algorithme 1 montre le pseudo-code de ConfPlan.

ConfPlan est bâti sur le principe de planification HTN (*Hierarchical Task Network* - réseaux hiérarchiques de tâches) [36] auquel des extensions ont été intégrées pour répondre à des besoins particuliers de certaines applications robotiques. Plusieurs raisons motivent l'utilisation d'une approche HTN. Comme cela est expliqué à la fin du chapitre 6, la spécification de missions à l'aide de tâches est naturelle pour des applications robotiques, et la spécification des réseaux de tâches est intuitive à créer. De plus, puisqu'un algorithme HTN fonctionne avec une recherche à chaînage avant, il est simple d'y intégrer la gestion de métriques comme présentée à la section 4.7.

Algorithme 1 Planificateur ConfPlan

```
1.  ConfPlan(Tâches T, Ordres O, État s) : Plan
2.    Forall t ∈ T
3.      If Planifier({t}, {}, s) == ÉCHEC alors T -= {t}
4.    Forall t1, t2 ∈ T && t1 != t2
5.      If Planifier({t1, t2}, O, s) == ÉCHEC
6.        T -= minimum_priorité{t1, t2}
7.    Plan P = ÉCHEC
8.    While(P == ÉCHEC)
9.      ContraintesGlobalesÉchoués = { }
10.     P = Planifier(T + TS, O, s)
11.     If P == ÉCHEC
12.       If ∃x ∈ ContraintesGlobalesÉchouées && x.nb < x.MaxTentative
13.         TS += {x.solution} ; x.nb++
14.       Else
15.         T -= minimum_priorité(T)
16.         TS = {}
17.   Return P

18. Planifier(Tâches T, Ordres O, État s) : Plan
19.   while ∃t ∈ T && t n'est pas une tâche primitive
20.     T -= {t}
21.     Choisir m, une méthode de décomposition de la tâche t
22.       T += sous_tâches(m(t))
23.       Ajuster O à l'aide de : ordres(m(t))
24.   Forall t1, t2 ∈ T && t1 != t2
25.     If t1.maxend < t2.minbegin
26.       O += (t1, t2)
27.   P = Ordonnancer(T, O, s)
28.   Return Meilleur(P) ∀ choix de m en ligne 21

29. Ordonnancer(TâchesPrimitives T, Ordres O, État s) : Plan
30.   Plan P = ÉCHEC
31.   Forall T' = (Ta1, Ta2, ..., Tan) ∈ Permutations(T, O)
32.     If((P !=ÉCHEC && TempsCourant>=DuréeMin) || TempsCourant>=DuréeMax)
33.       Return P
34.     s' = s
35.     Forall t ∈ T'
36.       If t.estApplicableEn(s') == ÉCHEC
37.         Continuer ligne 31
38.       s' = t.appliquer(s')
39.       If TesterContraintesGlobales(s') ÉCHEC
40.         Continuer ligne 31
41.   If Coût(T') < Coût(P) P = T
42.   Return P

43. TesterContraintesGlobales(État s) : Booléen
44.   Forall x ∈ Domaine.ContraintesGlobales
45.     If x.tester(s) == ÉCHEC
46.       ContraintesGlobalesEchoues += {x}
47.       Return ÉCHEC
48.   Return SUCCÈS
```

Contrairement à l'algorithme HTN original, dans ConfPlan, les préconditions au niveau des méthodes de décomposition de tâches ne dépendent pas de l'état courant. Les tâches peuvent ainsi être toutes décomposées avant même de commencer leur ordonnancement, permettant ainsi à simplifier l'implémentation de l'algorithme. Ceci correspond aux lignes 19 à 23 de l'algorithme 1.

7.1 Contraintes temporelles

Dans beaucoup d'applications robotiques, comme pour le *AAAI Robot Challenge*, certaines tâches doivent être exécutées à l'intérieur d'intervalles de temps précis. L'algorithme HTN original n'est pas conçu pour gérer des contraintes de temps puisque le modèle des actions ne prévoit pas de durée. Au chapitre précédent, nous avons expliqué comment il est possible de gérer des contraintes de temps en ajoutant une variable numérique *current-time* dans la représentation des états. Puisque cette solution n'est pas suffisamment efficace, une amélioration à l'algorithme HTN original a été apportée à ConfPlan.

Elle consiste à intégrer une variable temporelle *time* directement dans la représentation interne des états de ConfPlan. Le planificateur a lui-même accès à cette variable, contrairement à l'approche ad hoc qui consiste à ajouter cette variable dans la spécification du domaine et où seuls les préconditions et les effets pouvaient y faire référence. De plus, dans la définition des tâches, nous avons ajouté les attributs *minbegin*, *maxbegin*, *minend* et *maxend* permettant de définir des bornes temporelles sur le commencement et de la terminaison des tâches. Dans les méthodes de décompositions de tâches, ces attributs peuvent être hérités ou recalculés lors de la création des sous-tâches.

La figure 7.1 illustre un exemple de décomposition de la tâche *Présenter-Affiche(10h00, 12h00, Stand27)*. Cette tâche indique qu'il faut présenter une affiche entre 10h00 et 12h00 au stand numéro 27. Au niveau de la tâche *Présenter-Affiche*, seul l'attribut *maxend* est spécifié puisqu'il faut retirer l'affiche au maximum une heure après la fin de la session de présentation. La tâche de haut niveau se décompose en deux sous-tâches : *Installer-Affiche*

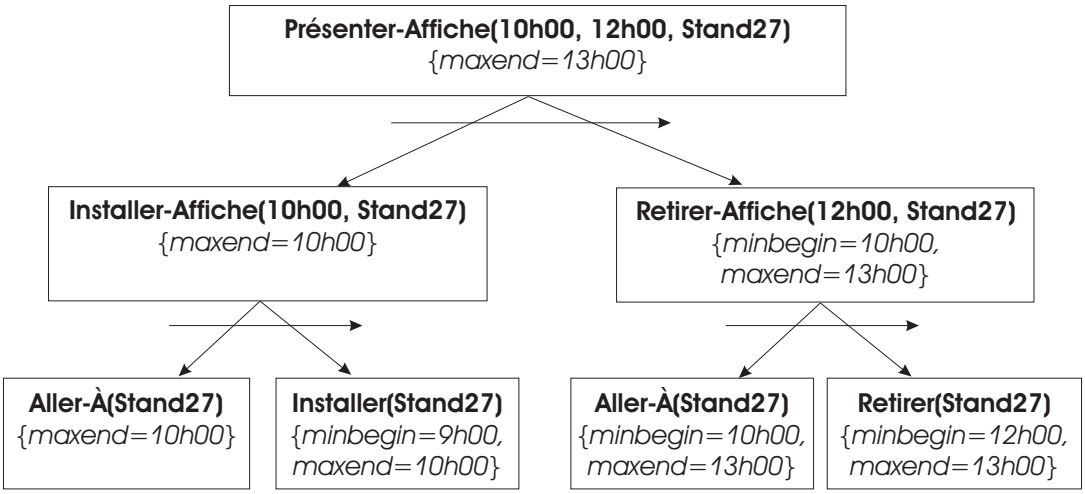


Figure 7.1 – Exemple de propagation des contraintes de temps.

et *Retirer-Affiche*. Puisque l'affiche doit être installée pour 10h00, l'attribut $maxend$ de la tâche *Installer-Affiche* est fixé à 10h00. De façon similaire, des bornes temporelles sont fixées pour les autres sous-tâches.

Lors de la planification, les bornes temporelles peuvent être analysées pour déduire de nouveaux ordres partiels sur les tâches de la mission. Pour deux tâches t_i et t_j, si $t_i.maxend \leq t_j.minbegin$, l'ordre partiel $t_i < t_j$ est ajouté à l'ensemble des ordres partiels O (voir lignes 24 à 26 de l'algorithme 1). Les nouveaux ordres partiels permettent de réduire l'espace de recherche et ainsi d'accélérer la génération des plans.

7.2 Gestion des priorités

Généralement, il est souhaité qu'un robot accomplisse le maximum de tâches qui lui sont soumises. Or, il peut arriver qu'il soit impossible de toutes les réaliser pour diverses raisons comme des conflits de contraintes temporelles ou tout simplement parce que la mission est trop ambitieuse pour le temps disponible. Dans de tels cas, il faut être en mesure de réaliser partiellement la mission en éliminant les tâches impossibles à réaliser ou celles moins prioritaires.

La planification classique et la planification HTN ne sont pas conçues pour gérer des

priorités de tâches. En planification classique, le but étant traité comme un ensemble indissociable de clauses, les algorithmes de planification retourne un plan en cas de succès et un échec lorsqu'il n'existe pas de solution. L'algorithme de ConfPlan a été adapté pour gérer des missions impossibles, c'est-à-dire des missions pour lesquelles il n'existe pas de plan solution. L'approche implémentée est plutôt simple et consiste à planifier pour un sous-ensemble de la mission lorsque cette dernière résulte à un échec. L'approche proposée se divise en trois étapes.

(1) Élimination des tâches non réalisables

Dans une première passe, la réalisation de chaque tâche de la mission est évaluée en la planifiant individuellement. Si cette tentative échoue, la tâche est immédiatement retirée de la mission (voir les lignes 2 et 3 de l'algorithme 1).

(2) Détection de tâches mutuellement exclusives

Dans une seconde passe, la détection des tâches mutuellement exclusives est effectuée. Cette étape vise particulièrement les tâches dont les contraintes de temps sont incompatibles. Par exemple, demander au robot de faire une présentation en même temps que de surveiller une salle, représente un tel conflit. Il peut aussi arriver que deux tâches soient en conflit même s'il n'y a pas d'intersection dans les intervalles de temps de ces deux tâches. Par exemple, si le robot doit aller à un endroit pour une certaine heure et aller à un autre 5 minutes plus tard, ces deux tâches sont en conflit s'il faut plus de 5 minutes au robot pour parcourir la distance séparant ces deux salles.

Pour détecter les tâches mutuellement exclusives, l'algorithme tente de générer un plan pour chaque paire de tâches de la mission. Lorsqu'il n'existe aucun plan, il est donc prouvé que les deux tâches sont mutuellement exclusives. Dans ce cas, la tâche ayant la plus faible priorité est retirée de la mission (voir les lignes 4 à 6 de l'algorithme 1).

Seule la détection de paires de tâches mutuellement exclusives est effectuée par l'algorithme. Ce prétraitement pourrait être étendu à des groupes de plus de deux tâches. Par

exemple, il est possible que les paires de tâches (t_1, t_2), (t_1, t_3) et (t_2, t_3) soient toutes réalisables mais que le triplet (t_1, t_2, t_3) ne le soit pas. Cette problématique n'est pas prise en compte pour plusieurs raisons. Tout d'abord, le premier objectif de ce prétraitement est de détecter les tâches ayant des contraintes temporelles en conflit. Ainsi, détecter uniquement les paires de tâches mutuellement exclusives est généralement amplement suffisant. De plus, tester toutes les combinaisons de tâches possibles augmente très rapidement en fonction de la taille des sous-groupes de tâches. Pour une mission de n tâches, vérifier les tâches mutuellement exclusives par groupe de k tâches requiert $\binom{n}{k}$ tests.

(3) Réduction de la mission au besoin

Après les deux premières passes, l'algorithme planifie l'ensemble de la mission réduite. Si aucune solution n'existe, alors la tâche ayant la plus faible priorité est retirée (voir ligne 15) et une replanification survient. Si nécessaire, cette stratégie est répétée jusqu'à ce qu'un plan valide soit trouvé. Dans le pire cas, il ne reste plus aucune tâche à planifier et un plan vide est retourné. Cette étape permet de garantir que le planificateur retourne toujours une solution, peu importe la mission qui lui est donnée.

Les priorités des tâches sont attribuées lors de la spécification du domaine. Dans l'implémentation actuelle de ConfPlan, le niveau de priorité d'une tâche est attribué selon son type. Par exemple, dans le cas du domaine de la conférence scientifique, faire une présentation a une priorité plus élevée que d'assister à une présentation.

Le coût de la gestion des priorités est faible. Dans la première passe de réduction de la mission, il faut faire autant de tests que de tâches. Ces tests consistent à générer plusieurs fois des plans pour une seule tâche. Dans la deuxième passe, pour n tâches, il faut planifier $n(n-1)/2 \approx \Theta(n^2)$ missions d'uniquement deux tâches. Puisque des missions d'une ou deux tâches sont des missions très faciles à planifier, ces opérations ne sont pas trop coûteuses. De plus, lors de ces tests, puisque seule l'existence d'une solution est vérifiée, la phase d'optimisation n'est pas réalisée.

7.3 Gestion des ressources

Puisqu'un robot mobile a une autonomie d'énergie limitée, il faut être en mesure de gérer cette ressource en planifiant la recharge des batteries au besoin. Comme cela a été vu à la section 4.7, plusieurs planificateurs sont conçus pour gérer des contraintes de ressources. Par contre, l'algorithme HTN original n'est pas conçu pour gérer efficacement des ressources.

Pour gérer les ressources dans ConfPlan, une approche similaire à IxTeT [27] a été intégrée. Dans la spécification du domaine, des **contraintes globales** (préconditions permanentes) sont définies et elles sont systématiquement vérifiées après l'application de chaque action (voir ligne 39). De plus, ces contraintes peuvent être associées à des tâches (solutions) permettant de les résoudre en cas de violation (voir ligne 13). Par exemple, pour la gestion des batteries, une contrainte c est mise sur la variable d'état représentant le niveau d'énergie en indiquant qu'elle doit toujours être supérieure à zéro. Lorsqu'aucun plan n'existe pour une mission donnée et que cette contrainte d'énergie a été violée au moins une fois lors de la planification, une tâche de recharge est ajoutée à la mission pour résoudre la contrainte violée et l'algorithme replanifie. Si cela n'est pas suffisant, une deuxième tâche de recharge est ajoutée avant de relancer la planification. Dans la spécification du domaine, un nombre maximal d'ajouts de ces tâches est fixé afin d'éviter que le planificateur n'en ajoute indéfiniment. Lorsque ce nombre maximal est atteint, le planificateur retourne un échec et la procédure de gestion de priorités élimine une tâche avant de relancer la planification.

7.4 Optimisation et solution en tout temps

ConfPlan possède la capacité de planifier une solution en tout temps. Cette capacité est mieux connue sous l'expression anglaise *anytime planning* [43, 29]. L'idée est simple : plus le planificateur a de temps, plus il peut améliorer la qualité de sa solution. Cette capacité est d'une grande importance dans les systèmes fonctionnant en temps réel, comme un

robot, puisqu'elle permet de garantir des délais de réponses.

ConfPlan implémente cette capacité de façon semblable à SHOP2. Pour ce faire, un temps minimal et un temps maximal de planification sont configurés. Le planificateur passe donc un minimum de temps à optimiser sa solution à moins qu'il ait exploré la totalité de l'espace de recherche. À l'opposé, lorsque le planificateur dépasse la limite maximale de temps permise, il considère la mission courante comme étant impossible à planifier.

L'optimisation réalisée par ConfPlan se fait au niveau de l'ordonnancement des tâches (voir lignes 29 à 42). Lorsqu'une mission est décomposée en tâches primitives, il peut exister une multitude de solutions. L'optimiseur explore un grand ensemble de permutations de tâches respectant les ordres partiels (ligne 31). Durant cette exploration, le planificateur valide les différentes séquences de tâches primitives en testant les préconditions (ligne 36) et en appliquant les effets (ligne 38). Lorsqu'il est possible d'appliquer toute la séquence de tâches primitives à partir de l'état courant, cette séquence forme un plan valide. Pour chaque plan valide trouvé, son coût est estimé à l'aide d'une fonction spécifiée dans le domaine. Enfin, le plan ayant un coût minimal est conservé (ligne 41). Par exemple, pour la livraison de colis, une fonction de coût pourrait être la distance totale parcourue par le robot.

L'optimisation peut aussi se faire en testant plusieurs méthodes de décomposition de tâche au niveau de la planification. Dans la procédure $Planifier$, la ligne 21, où l'on fait un choix d'une méthode de décomposition, est un point où il peut y avoir un retour en arrière (*backtracking point*). Ainsi, après avoir évalué plusieurs ordonnancements dans la procédure $Ordonnancer$, le planificateur peut alors changer de plan en choisissant d'autres méthodes de décomposition.

Algorithme 2 Calcul des intervalles de flexibilités temporelles

```
01. CalculIntervallesTemps(Plan P = (t_1, t_2, ..., t_n), État s_0)
02.   For i = 1 to n
03.     s_i = t_i.appliquer(s_{i-1})

04.   time = s_0.time
05.   For i = 1 to n
06.     If t_i.mintime < time)
07.       t_i.mintime = time
08.     If t_i.minend < t_i.minbegin + t_i.durée)
09.       t_i.minend = t_i.minbegin + t_i.durée
10.     time = t_i.minend

11.   time = t_n.maxbegin
12.   For i = n - 1 to 1
13.     If t_i.maxend > time
14.       t_i.maxend = time
15.     If t_i.maxbegin > t_i.maxend + t_i.durée
16.       t_i.maxbegin = t_i.maxend + t_i.durée
17.     time = t_i.maxbegin
```

7.5 Intervalles de flexibilités temporelles

Après la génération de plan, ConfPlan calcul des intervalles de flexibilités temporelles à l'aide de l'algorithme 2. Un intervalle de flexibilité temporelle spécifie une fenêtre de temps à l'intérieur de la quelle une action peut être accomplie. Dans une première passe (voir lignes 4 à 10), pour chacune des tâches, en partant du début du plan, les valeurs de leurs attributs *minbegin* et *minend*, représentant respectivement les temps minimaux estimés pour le commencement et la terminaison, sont ajustées. Dans une seconde passe (voir lignes 11 à 17), en partant de la fin du plan, les valeurs des attributs *maxend* et *maxbegin*, représentant respectivement les temps maximums pour la terminaison et le commencement, sont ajustés.

Ces intervalles sont utiles pour la validation de plan lors de l'exécution. De plus, le calcul de ces intervalles permet de représenter les plans sous forme de diagramme de Gantt. Par exemple, le plan sous forme textuelle[1] à la figure 7.2 peut être représenté graphiquement à l'aide du diagramme de Gantt à la figure 7.3.

[1]Chaque ligne i a la forme suivante : Action(paramètres) MaxInterval=[minbegin, maxend] MinInterval=[maxbegin, minend]. De plus, les valeurs des temps sont exprimées en secondes.

```
#. Action(params)  MaxInterval=[MinBegin, MaxEnd] MinInterval=[MaxBegin, MinEnd]
============================================================
0. AllerA(FrontRoom_3)  MaxInterval=[29700,32752] MinInterval=[32176, 30276]
1. DemanderMessage(FrontRoom_3,m1)  MaxInterval=[30276,32842] MinInterval=[32752, 30366]
2. AllerA(CoffeeRoom)  MaxInterval=[30366,33018] MinInterval=[32842, 30542]
3. DonnerMessage(CoffeeRoom,m1)  MaxInterval=[30542,33108] MinInterval=[33018, 30632]
4. AllerA(WP_3)  MaxInterval=[30632,33172] MinInterval=[33108, 30696]
5. DemanderMessage(WP_3,m0)  MaxInterval=[30696,33262] MinInterval=[33172, 30786]
6. AllerA(P31)  MaxInterval=[30786,33598] MinInterval=[33262, 31122]
7. PrendreMessage(P31,m0)  MaxInterval=[31122,33688] MinInterval=[33598, 31212]
8. AllerA(StandRoom_2)  MaxInterval=[31212,34200] MinInterval=[33688, 31724]
9. FairePres(StandRoom_2, 34200, 1200)  MaxInterval=[34200,35400] MinInterval=[34200, 35400]
10. AllerA(P1)  MaxInterval=[35400,43080] MinInterval=[42696, 35784]
11. Photographier(P1)  MaxInterval=[39600,43200] MinInterval=[43080, 39900]
```

Figure 7.2 – Exemple de plan sous forme textuelle.

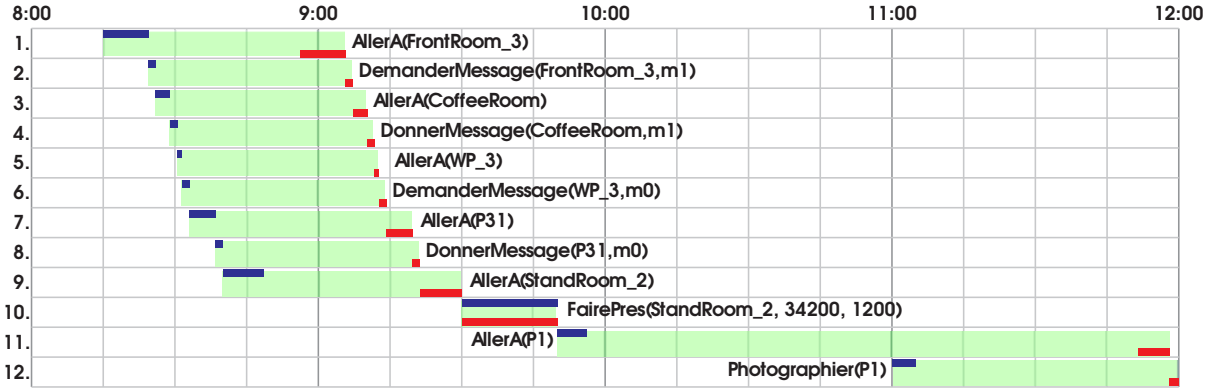


Figure 7.3 – Exemple de plan sous forme de diagramme de Gantt.

7.6 Implémentation

Le planificateur ConfPlan est implémenté entièrement en C++. Les principales classes de ConfPlan sont **State** (état), **World** (relations rigides du monde), **Task** (une tâche), **TasksSet** (un ensemble de tâches partiellement ordonnées) et **Plan** (une séquence de tâches primitives).

Puisque la version actuelle de ConfPlan ne supporte pas de langage de spécification de domaine comme PDDL (*Problem Definition Description Language*) [14], la spécification du domaine doit être entièrement encodée dans les classes **State** et **World** qui sont déclarées et définies dans les fichiers sources *domaine.h* et *domaine.cpp*. Pour chaque tâche du

domaine, il faut créer une classe héritant de la classe **Task**. Les tâches primitives doivent implémenter les fonctions virtuelles **isApplicable(const State&, const World&)** et **apply(State&, const World&)** afin de spécifier respectivement les préconditions et les effets. Les tâches de haut niveau doivent implémenter la fonction virtuelle **getSubtasks(const World&)** afin de spécifier une méthode de décomposition. Dans la mise en œuvre actuelle, le nombre de méthode de décomposition pour un type de tâche est limité à un.

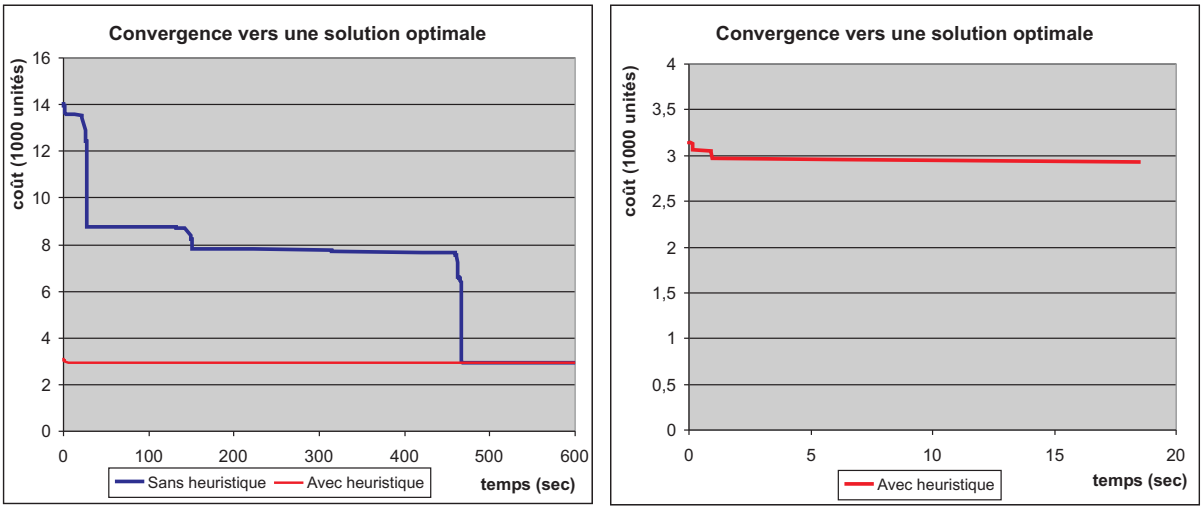


Figure 7.4 – Convergence vers solution optimale.

Comme expliqué à la section 7.4, la procédure *Ordonnancer* de l'algorithme de ConfPlan peut avoir un très grand nombre de solutions à évaluer. Puisqu'il n'est pas toujours possible de toutes les tester afin de retenir la meilleure solution, une attention particulière doit être apportée à l'ordre par lequel les différentes solutions sont évaluées. Nous avons donc intégré une heuristique dans l'ordonnanceur afin de converger le plus rapidement possible vers la solution optimale. L'heuristique utilisée consiste à ordonner les tâches de haut niveau de la mission par ordre croissant de leur coût individuel. L'utilisation de cette heuristique permet de converger plus rapidement vers une meilleure solution. Pour un problème donné, la figure 7.4 illustre l'amélioration apportée par cette heuristique. Sans heuristique, le planificateur commence par générer un premier plan très coûteux et met près de 500 secondes avant d'atteindre une solution proche optimale.

7.7 Spécification du domaine de la conférence scientifique

Pour la spécification du domaine de la conférence scientifique, les classes **World** et **State** sont définies de la façon suivante :

Classe World

- vector<Location> pointsReperes ; *// liste des points de repère*
- double** **distances** *// table des distances entre paires de points de repère*
- double **speed** *// vitesse moyenne du robot*
- Location* **rechargePlace** *// endroit où le robot peut se recharger*
- double **batteryCapacity** *// capacité maximale de la batterie*
- double **energyPerMeter** *// quantité d'énergie consommée par mètre*
- double **energyPerSecond** *// quantité d'énergie consommée par seconde*

Classe State

- Location* **pos** *// position courante du robot (point de repère)*
- int **time** *// temps courant de l'état (exprimé en secondes)*
- bool **registered** *// est-ce que le robot est inscrit à la conférence*
- double **battery** *// niveau de charge de la batterie*

Le monde, défini par la classe **World**, contient tout ce qui est statique dans le domaine, comme la liste des points de repère dans l'environnement. Cette classe contient aussi les spécifications techniques du robot, comme sa vitesse moyenne et ses exigences énergétiques. Elle définit aussi une table de distances, soit une matrice carrée contenant la distance minimale séparant chaque paire de points de repère. Cette matrice est précalculée en faisant un appel à l'algorithme de Dijkstra [12] pour chaque point de repère. Pour de meilleures performances, l'algorithme Floyd-Warshall [16] pourrait être utilisé.

L'état, défini par la classe **State**, contient toutes les informations variables dans le domaine, c'est-à-dire tout ce que le robot peut influencer par ses actions, comme la position du robot, le temps courant, l'état d'inscription à la conférence et le niveau de charge des batteries.

7.7.1 Tâches du domaine

Les tâches de haut niveau pour le domaine de la conférence scientifique sont illustrées à la figure 7.5 avec leurs méthodes de décomposition. Une mission à planifier consiste en un ensemble d'instances de tâches de haut niveau, *LivrerMessage(p1, p2, m1)* étant un exemple d'instance de tâche qui consiste à livrer un message $m1$ de $p1$ à $p2$.

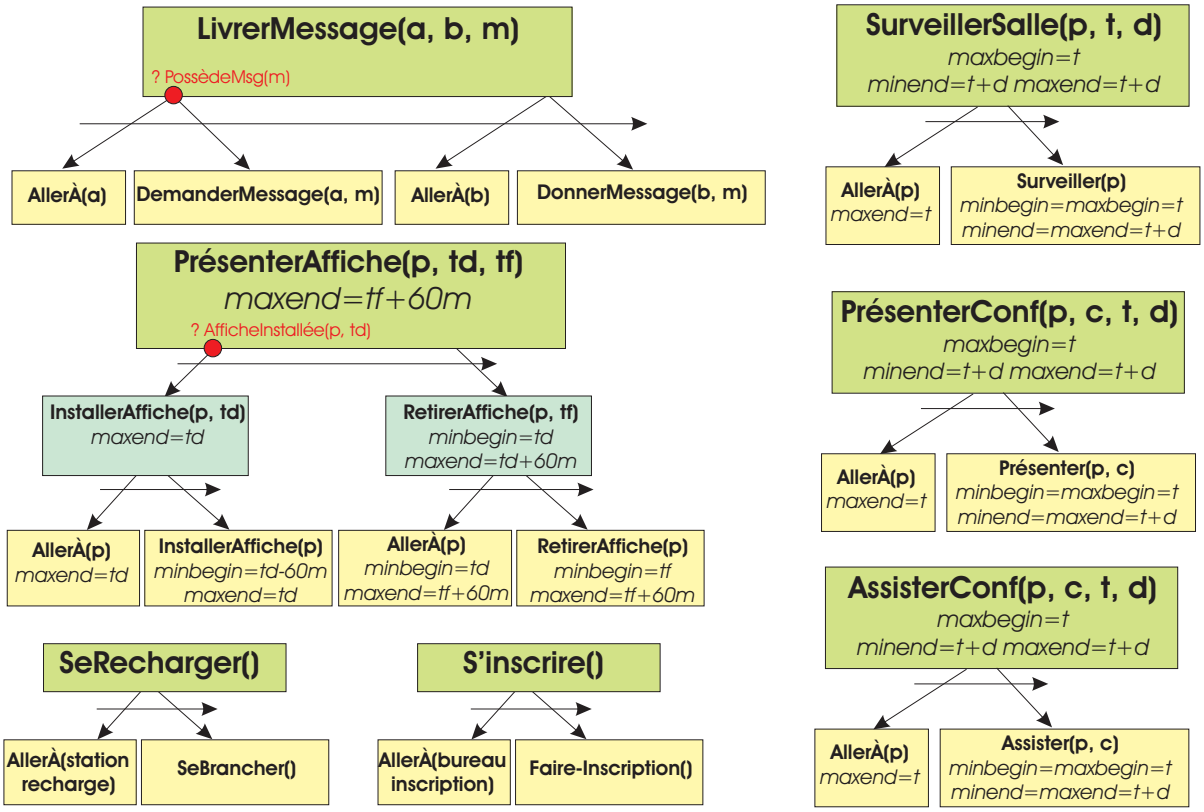


Figure 7.5 – Tâches pour le domaine de la conférence scientifique.

Les spécifications des tâches primitives, c'est-à-dire les feuilles des réseaux de tâches illustrés à la figure 7.5, sont les suivantes[2] :

AllerÀ(dest)

Préconditions :

- (aucune)

Effets :

- $s.pos = dest$
- $s.time \mathrel{+}= w.distances(s.pos, dest) \div w.speed$
- $s.battery \mathrel{-}= w.distances(s.pos, dest) \times w.energyPerMeter$

[2]Les symboles s et w sont des alias faisant référence aux objets *state* (état courant) et *world* (monde).

Assister(p, conf)

Préconditions :

- $s.pos = p$
- $s.time \leq minend$

Effets :

- $s.time = minend$
- $s.battery$ -= $(minend - s.time) \times w.energyPerSec$

DemanderMessage(origine, msg)

Préconditions :

- $s.pos = origine$

Effets :

- $s.time$ += 90
- $s.battery$ -= $90 \times w.energyPerSec$

DonnerMessage(destination, msg)

Préconditions :

- $s.pos = destination$

Effets :

- $s.time$ += 90
- $s.battery$ -= $90 \times w.energyPerSec$

FaireInscription()

Préconditions :

- $s.pos = w.bureauInscription$

Effets :

- $s.registered = true$
- $s.time$ += 300 *(5 min)*
- $s.battery$ -= $300 \times w.energyPerSec$

InstallerAffiche(p)

Préconditions :

- $s.pos = p$

Effets :

- $s.time$ += 120
- $s.battery$ -= $120 \times w.energyPerSec$

Présenter(p, conf)

Préconditions :

- $s.pos = p$
- $s.time \leq minend$

Effets :

- $s.time = minend$
- $s.battery$ -= $(minend - s.time) \times w.energyPerSec$

Surveiller(p, conf)

Préconditions :

- $s.pos = p$
- $s.time \leq minend$

Effets :

- $s.time = minend$
- $s.battery$ -= $(minend - s.time) \times w.energyPerSec$

RetirerAffiche(p)

Préconditions :

- $s.pos = p$

Effets :

- $s.time$ += 120
- $s.battery$ -= $120 \times w.energyPerSec$

SeBrancher()

Préconditions :

- $s.pos = w.rechargePlace$

Effets :

- $s.time$ += 3600 *(1 heure)*
- $s.battery = w.batteryCapacity$

7.7.2 Contrainte globale

Au lieu de spécifier une précondition sur le niveau d'énergie pour chacun des types de tâche primitive, nous avons défini une contrainte globale devant être vérifiée après l'application des effets. Ainsi, il n'est pas nécessaire de spécifier la précondition *s.battery > w.distances(s.pos, dest)× w.energyPerMeter* pour l'action *AllerÀ(dest)*. Il suffit plutôt de définir la contrainte globale *state.battery>0* à laquelle est associée la tâche de haut niveau *SeRecharger(w.rechargePlace)*. Lorsque ConfPlan n'arrive pas à trouver un plan dû à une violation de cette contrainte, il ajoute alors la tâche de recharge dans la mission.

7.7.3 Critère d'optimisation

Dans le contexte de la conférence scientifique, il est souhaitable que le robot réalise le plus rapidement possible ses tâches et qu'il optimise ses déplacements afin de minimiser l'usage de ses ressources énergétiques. Pour spécifier ces critères d'optimisation au planificateur, il faut les transformer en une expression mathématique permettant au planificateur d'évaluer le coût des plans. Le planificateur peut donc choisir un plan qui minimise le coût d'exécution. L'expression choisie pour le présent domaine est une somme pondérée du temps total de la mission, des délais de livraison des messages et de la distance totale parcourue. La fonction $c(p)$ évaluant le coût d'un plan p est définie par l'équation 7.1 où $p.distance$ est la distance totale parcourue en mètres, $p.duree$ est la durée estimée

du plan en secondes, $delaiattente(m)$ est le temps que doit attendre le destinataire du message m, et M est l'ensemble des messages à livrer. Les coefficients 0.15 et 0.2 sont choisis de façon arbitraire. Ils peuvent être ajustés selon les préférences de l'utilisateur final du robot.

$$c(p) = p.distance + 0.15 \times p.duree + 0.2 \times \sum_{m \in M} delaiattente(m) \tag{7.1}$$

7.8 Résultats

Nous avons expérimenté le planificateur ConfPlan avec les mêmes séries de tests que ceux présentés au chapitre 6. Nous avons testé ConfPlan dans deux modes : 1) ConfPlan tente de générer rapidement des plans valides[3] ; 2) ConfPlan utilise l'optimiseur pour une durée pouvant aller jusqu'à 120 secondes[4] (ConfPlan 120). De plus, comme les problèmes sont tous réalisables, aucun temps maximal de planification n'a été spécifié. En d'autres mots, la procédure de simplification de mission n'est jamais employée dans ces tests. Ainsi, ConfPlan prend tout le temps requis pour générer des solutions complètes. Lorsque ConfPlan dépasse le temps minimal de planification et qu'aucune solution n'est trouvée, il continue son travail jusqu'à ce qu'un premier plan valide soit trouvé. Dans de tels cas, la solution retournée n'est donc pas optimale alors que dans tous les autres cas, c'est-à-dire où la durée de planification est inférieure au temps minimum de planification, la solution retournée est garantie d'être optimale. Le tableau 7.1 et la figure 7.6 montrent les résultats. Ici, on ne s'intéresse qu'à la durée de planification. La qualité des plans n'est pas comparée. Une partie de ces résultats a été présentée à la conférence Canadian AI [7] qui s'est tenue à Victoria, C.-B. en mai 2005.

Les résultats obtenus montrent clairement que ConfPlan est très efficace pour ce type de domaine. Par contre, une partie de l'efficacité peut s'expliquer en raison de la conception

[3]Le temps minimal de planification est fixé à une seconde. Une fois une solution trouvée, le planificateur continue à explorer d'autres solutions jusqu'à ce que le temps minimal soit atteint ou que toutes les possibilités aient été explorées.

[4]Le temps minimal de planification est fixé à 120 secondes.

technique de ConfPlan. En effet, puisque l'ensemble du domaine est directement codé dans des classes C++ et compilé à l'intérieur du planificateur lui-même, cela permet d'obtenir un gain d'efficacité non négligeable puisqu'il y a un niveau d'abstraction en moins par rapport aux autres planificateurs. En d'autres mots, le planificateur ConfPlan n'a pas à effectuer les étapes de lecture, d'analyse syntaxique et d'analyse sémantique pour le chargement d'un domaine en mémoire, contrairement aux autres planificateurs évalués. Cependant, puisque l'étape du chargement en mémoire du domaine et du problème est plutôt d'une durée fixe, elle n'est pas très significative pour les problèmes plus difficiles. La principale raison expliquant ces bonnes performances est la façon dont ConfPlan gère les contraintes de temps. En déterminant automatiquement des ordres partiels déduits des contraintes de temps, ConfPlan peut réduire radicalement le nombre de possibilités par rapport aux autres planificateurs n'offrant pas cette capacité.

Figure 7.6 – Comparaison des planificateurs.

Tableau 7.1 – Tableau des temps (sec) de planification des planificateurs évalués.

		Nombre de tâches									
Série	**Planificateur**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Missions mixtes	**ConfPlan**	0,00	0,00	0,00	0,00	0,00	0,02	0,54	0,53	0,61	1,31
	ConfPlan 120	0,00	0,00	0,00	0,00	0,00	0,02	6,29	2,47	36,5	50,1
	LPG	259	480	435	600	600	600	431	600	600	600
	Metric-FF	2,84	6,46	72,3	74,7	80,6	261	218	226	439	552
	Sapa	0,58	1,02	63,1	251	434	543	486	600	547	600
	SHOP2	0,10	0,67	35,2	465	600	600	600	600	600	600
Livraison de messages	**ConfPlan**	0,00	0,00	0,00	0,01	0,66	1,48	1,6	1,52	1,39	1,52
	ConfPlan 120	0,00	0,00	0,00	0,01	0,65	37,8	121	121	121	121
	LPG	0,61	1,11	2,23	2,4	2,95	3,66	4,03	4,8	5,4	5,47
	Metric-FF	3,02	7,63	13	18,8	25,6	27,3	32,9	47,1	56,6	68,4
	Sapa	0,75	1,71	5,62	18,5	26,1	39,4	45,5	75,2	130	227
	SHOP2	0,14	1,38	44,6	600	600	600	600	600	600	600
Présentation d'affiches	**ConfPlan**	0,00	0,00	0,00	0,00	0,00	0,05	0,26	1,5	1,53	1,56
	ConfPlan 120	0,00	0,00	0,00	0,00	0,00	0,05	0,26	17,6	84,2	121
	LPG	600	567	600	600	600	600	600	600	600	600
	Metric-FF	4,50	68,5	307	310	544	600	600	600	600	600
	Sapa	0,56	1,15	5,86	260	28,6	39,5	196	268	476	433
	SHOP2	0,10	1,01	62,2	600	600	600	600	600	600	600
Faire des conférences (temps fixes)	**ConfPlan**	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	ConfPlan 120	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	LPG	0,25	0,37	60,4	360	480	480	600	480	420	420
	Metric-FF	2,87	124	244	305	425	136	193	310	368	194
	Sapa	0,54	120	241	301	421	123	184	363	362	244
	SHOP2	0,09	0,39	4,21	81,3	600	600	600	600	600	600
Prendre en photos des affiches	**ConfPlan**	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,01	0,02	0,57
	ConfPlan 120	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,01	0,03	0,57
	LPG	600	442	282	224	416	438	513	412	551	600
	Metric-FF	1,57	5,34	5,96	68,2	14	12,3	17,1	77,4	20,4	80,8
	Sapa	0,54	0,74	4,83	8,12	102	12,2	139	57,2	133	178
	SHOP2	0,09	0,33	3,31	47	600	600	600	600	600	600

CHAPITRE 8

Intégration de ConfPlan dans une architecture décisionnelle

Le planificateur ConfPlan a été intégré dans MBA (*Motivated Behavioral Architecture*) [6, 32], une architecture hybride à laquelle des modules motivationnels et un conteneur de tâches ont été ajoutés. Cette architecture est représentée la figure 8.1.

Les **motivations** sont responsables du contrôle de haut niveau du robot. Dans l'implémentation actuelle, deux types de motivation sont utilisés : rationnelles et instinctives. Les motivations instinctives sont influencées principalement par des perceptions et des règles simples, tandis que les motivations rationnelles utilisent des processus de raisonnement plus évolués. Par exemple, la motivation instinctive *Survie* ne fait qu'ajouter une tâche de rechargement lorsque la tension des batteries baisse sous un certain seuil, contrairement à la motivation rationnelle *Planificateur* qui ajoute des tâches selon un plan précis.

Les tâches courantes du robot se retrouvent dans le **conteneur de tâches**, aussi appelé *Dynamic Task Workspace*. Chaque motivation peut y ajouter ou enlever des tâches. Par exemple, la motivation *Obéir* est responsable d'ajouter les tâches soumises à partir de l'interface graphique du robot. De plus, certaines motivations sont responsables de la décomposition d'un certain nombre de tâches en vue de leur exécution. Par exemple,

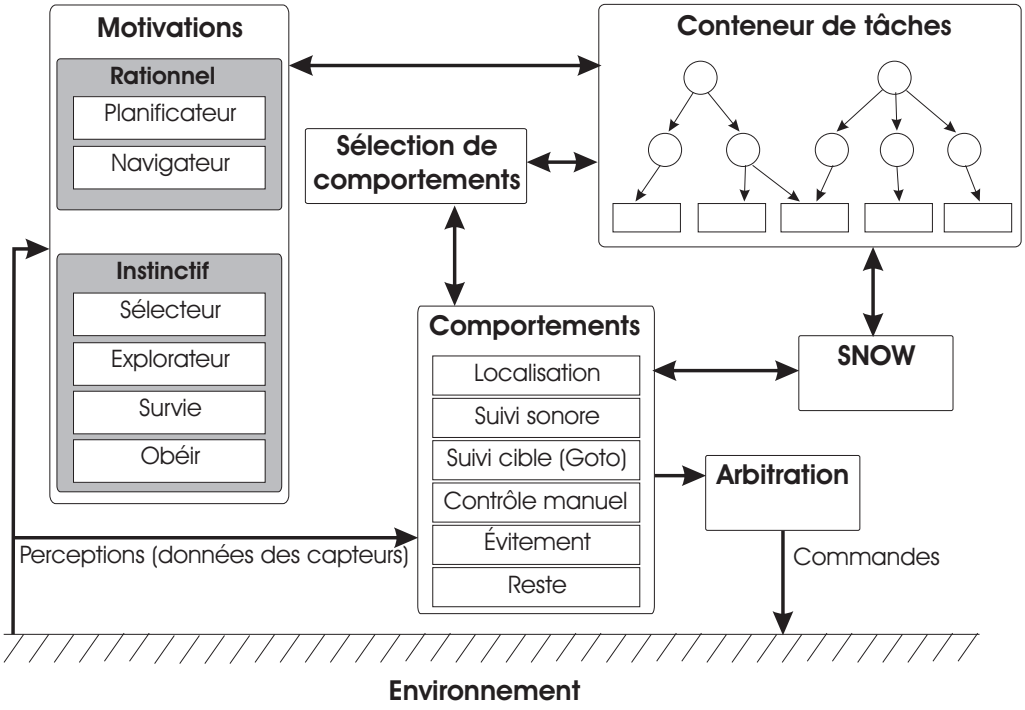


Figure 8.1 – Architecture MBA.

pour une tâche de livraison de message, le *Planificateur* est responsable de générer successivement les tâches *AllerÀ*, *PrendreMessage* et *DonnerMessage*. De façon similaire, dans le cas d'une tâche de déplacement (*AllerÀ(B)*), le *Navigateur* est responsable de générer successivement des tâches *Goto* faisant passer le robot par des cibles intermédiaires afin d'attendre le point B. Pour être admissible à l'exécution, une tâche doit être recommandée par au moins une motivation sans être non désirée par une autre.

Dans MBA, une tâche est définie par un identificateur numérique unique (ID), un nom, une liste de paramètres, les identificateurs des tâches parents, la ou les motivation(s) propriétaire(s) de cette tâche, une liste de recommandations et son état (actif ou complété).

Les motivations peuvent interagir de plusieurs manières avec le conteneur de tâches. En premier lieu, il y a le mode interrogation dans lequel les motivations peuvent interroger le conteneur de tâches. De cette façon, il est entre autres possible de lister les tâches, de chercher les tâches d'un type particulier et d'obtenir les paramètres pour un numéro de tâche. En deuxième lieu, il y a aussi le mode événementiel à l'aide duquel les motivations peuvent aussi s'inscrire à des événements particuliers, comme l'ajout d'un certain type

de tâche ou la modification d'un paramètre précis pour n'importe quelle tâche.

Parmi les tâches de bas niveau pouvant se retrouver dans le conteneur de tâches, un certain nombre d'entre elles peuvent être directement associées à des comportements. Ces dernières sont appelées **tâches comportementales**. Par exemple, une tâche comportementale *Goto* est naturellement associée au comportement du même nom. L'activation des modules comportementaux se fait en sélectionnant les tâches comportementales ayant été recommandées dans le conteneur de tâches.

8.1 Intégration du planificateur dans MBA

La motivation *Planificateur* est l'endroit où le planificateur ConfPlan est intégré dans l'architecture MBA. Cette intégration est réalisée à l'aide d'une boucle réactive, qui est résumée par l'algorithme 3. Cette boucle est divisée en deux parties, l'une pour l'exécution et la surveillance du plan courant, et l'autre pour la planification. Ces deux boucles peuvent s'exécuter simultanément dans deux fils d'exécution (*thread*) distincts.

8.1.1 Réaction aux événements

La première boucle d'exécution (**MBA_Planification**) s'occupe de l'exécution et de la surveillance des plans, et a aussi le rôle de répondre aux événements externes au module de planification. Les principaux événements sont l'ajout, la suppression, la modification et la terminaison de tâches dans le conteneur de tâches. Suite à ces événements, le module de planification ajuste la mission courante (liste *mission*) et indique le besoin de replanifier à l'aide de la variable *besoinreplan*.

Sur un événement de terminaison d'une tâche (lignes 14 à 21), un traitement supplémentaire est nécessaire. En premier lieu, si la tâche complétée est celle que le plan courant recommande pour l'exécution, alors il faut vérifier s'il s'agit de la dernière sous-tâche du plan pour réaliser une tâche de haut niveau de la mission. Si tel est le cas, alors cette dernière doit aussi être marquée complétée. Suite à la terminaison de la tâche courante,

Algorithme 3 Boucle de planification réactive.

```
01. MBA_ExecutionSurveil()
02.   Loop forever
03.     e = waitForEvent(5 sec) ; // après 5 sec, un événement timeout
04.     Switch(e) :
05.       NouvelleTâche(t) :
06.         mission += t
07.         besoinreplan = true
08.       AnnulationTâche(t) :
09.         If(mission.contient(t)) :
10.           mission -= t
11.           besoinreplan = true
12.           If(d.descendantDe(tâchecourante)) :
13.             tâchecourante = <indéfinie>
14.       TerminaisonTâche(t) :
15.         If(t == tâchecourante) :
16.           If(plan.DerniereTacheDescendateDe(t.parent)) :
17.             ConteneurTâches.MarquerTerminé(t.parent)
18.           tâchecourante = plan.prochaineTâche()
19.         If(mission.contient(t)) :
20.           mission -= t
21.           besoinreplan |= plan.resteTâchesDescendantesDe(t)
21.       ÉchecTâche(t) :
22.         If(t == tâchecourante) :
23.           tâchecourante = <indéfinie>
24.           besoinreplan = true
25.       CollecteInformation(info) :
26.         étatcourant.metteàjour(info)
27.       IndiceTerminaison(it) :
28.         étatprojeté = estimerEtat(it)
29.     besoinreplan |= (plan.valider(étatcourant, ...) == ÉCHEC)
30.     If(besoinreplan)
31.       Thread_MBA_Planification.réinitialiser()
32.     ConteneurTâches.recommander(tâchecourante)

33. MBA_Planification()
34.   Loop forever
35.     If(plan.progressePlusViteQuePrevu() &&
36.           !plan.prochaineTache().estPrete(currentTime + tempsPlanification))
37.       besoinreplan = true
38.
39.       If(besoinreplan) :
40.         besoinreplan = false
41.         plan = ConfPlan(mission, étatcourant, tâchecourante, étatprojeté)
42.         tachecourante = plan.prochaineTâche()
43.         Thread_MBA_ExecutionSurveil.sendTimeoutEvent()
```

il faut passer l'exécution à la prochaine tâche dans le plan. En second lieu, si la tâche complétée fait partie de la mission courante, il faut la retirer de la mission afin d'éviter de l'exécuter deux fois (ligne 20).

8.1.2 Planification et exécution simultanées

L'un des premiers défis à résoudre lors de l'intégration d'un planificateur dans une architecture décisionnelle pour un robot mobile est de gérer les situations dans lesquelles un plan doit être généré alors que le robot est déjà en train d'en exécuter un autre et que l'action courante n'est pas terminée. Il faut donc être en mesure de générer des plans tout en tenant compte de la progression de l'action courante. Pour remédier à ce genre de situation, nous avons introduit la notion d'**état projeté**. Lorsque le module de planification recommande une nouvelle tâche primitive (ligne 32), il estime l'état projeté qu'atteindra le système à la fin de cette action en appliquant les effets de cette tâche dans l'état courant du système. De plus, comme la progression réelle de l'action peut diverger avec ce qui est prévu par le modèle du domaine, l'état projeté peut être constamment mis à jour à l'aide des plus récentes informations extraites ou déduites de l'environnement.

Enfin, l'algorithme ConfPlan a été légèrement modifié pour séparer l'espace de recherche en deux. Comme présentée à la figure 8.2, la construction du graphe de recherche se fait normalement à l'exception de l'application de l'action courante du robot. Dans ce cas, au lieu d'appliquer les effets de la spécification du domaine, l'état projeté est utilisé. Pour toutes les autres actions, les effets sont appliqués tel que prescrit dans la spécification du domaine. Ainsi, le planificateur peut décider de poursuivre l'action courante tout en tenant compte de sa progression, puisque l'état projeté est calculé en tenant compte de l'exécution courante.

8.1.3 Mise à jour de l'état projeté

Les lignes 27 et 28 de la boucle de planification réactive traitent la réception d'indices sur la progression de la tâche en cours d'exécution. Par exemple, pour les tâches de dépla-

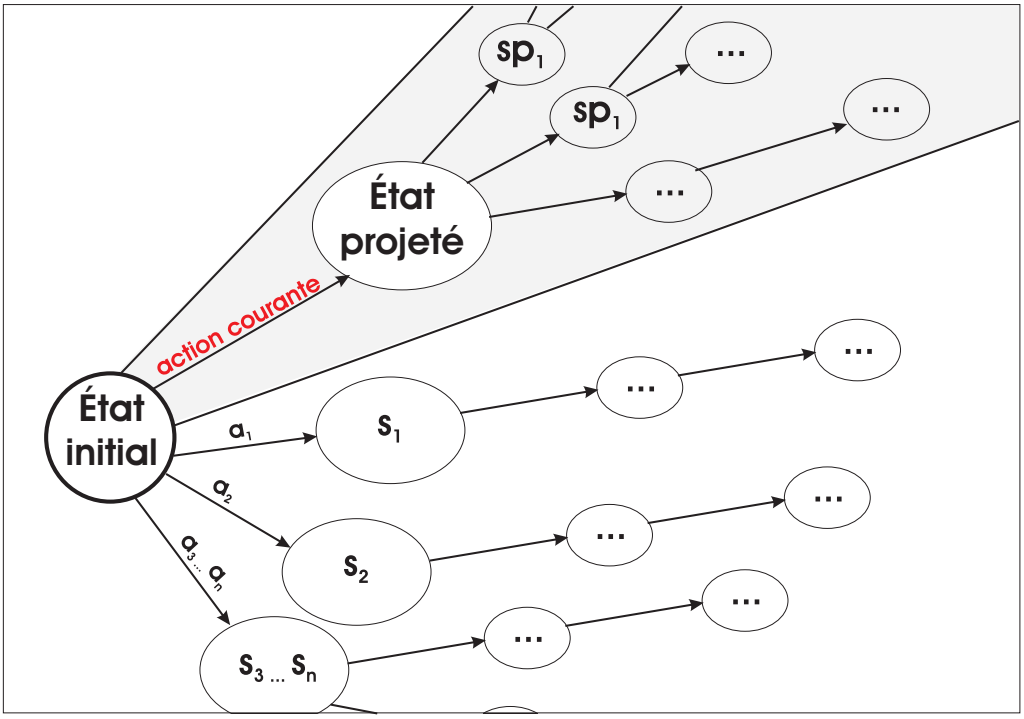


Figure 8.2 – ConfPlan avec action courante.

cement, le module motivationnel *Navigateur* estime constamment la distance restante à franchir pour atteindre le but courant. En connaissant la vitesse du robot, le module de planification est donc en mesure d'estimer à quel moment cette tâche sera complétée pour ainsi construire l'état projeté. Puisque cette information est rafraîchie régulièrement, elle est d'une grande précision et permet ainsi de générer des plans de meilleure qualité et aussi faire une meilleure validation du plan courant.

8.1.4 Validation du plan courant

La validation du plan courant est d'une très grande importance pour l'intégration d'un planificateur dans une architecture décisionnelle robotique. Elle permet de s'assurer que le plan courant reste toujours une solution valide à la mission courante. De façon continue (toutes les 5 secondes), notre algorithme simule l'exécution du plan courant à partir de l'action courante afin de s'assurer que toutes les préconditions des actions suivantes sont satisfaites. Dans le cas contraire, l'algorithme détecte alors que le plan cou-

rant est invalide, et le processus *MBA_ExecutionSurveil* (ligne 31) informe le processus *MBA_Planification* qu'un nouveau plan doit être généré.

Pour une validation précise, la fonction de validation de plan utilise la notion d'état projeté, tout comme le planificateur. Cette stratégie permet de détecter le plus rapidement possible les plans invalides.

Entre l'instant où un plan est déclaré invalide et l'instant auquel le planificateur retourne un nouveau plan, le robot poursuit l'exécution de la tâche courante. Nous procédons ainsi afin d'éviter toute interruption non souhaitée du robot. Souvent, le nouveau plan commence par la tâche qui est déjà en exécution, de sorte que cette stratégie permet au robot de poursuivre sa mission comme si de rien n'était. De plus, le temps de replanification étant très court, au pire, le robot continue l'ancienne action quelques secondes en trop.

8.1.5 Compromis entre fiabilité et efficience

Une qualité recherchée pour un robot est sa fiabilité dans l'accomplissement de ses missions : le robot doit réussir ses missions correctement tout en respectant les contraintes de temps prescrites. Une autre qualité aussi recherchée est l'efficience : le robot doit accomplir un maximum de tâches dans un laps de temps donné et les réaliser le plus rapidement possible. Dans beaucoup de situations, ces deux qualités sont en conflit.

Puisqu'un robot évolue dans un environnement dynamique et incertain, il n'est pas possible de prédire à quels moments il sera en difficulté. Un compromis doit être établi entre fiabilité et efficience au moment de la planification, en faisant des hypothèses prudentes au niveau de la spécification du domaine. Afin de prévenir les situations où le robot aurait de la difficulté à se déplacer, nous avons donc spécifié une vitesse inférieure à celle réelle dans la spécification du domaine de planification. Cela dégage une certaine marge de manœuvre à l'exécution et augmente ainsi la probabilité de succès des plans. Le prix à payer de cette stratégie est que le robot risque d'avoir un comportement légèrement moins efficient, de rater certaines opportunités et dans le pire cas, il peut aller jusqu'à abandonner certaines tâches ne pensant pas être en mesure de toutes les réaliser.

8.1.6 Détection d'opportunisme

Puisque nous avons fait une hypothèse prudente sur la vitesse du robot dans la spécification du domaine, il est alors fort probable que les tâches de déplacement progresseront plus rapidement à l'exécution par rapport à ce qu'elles ont été planifiées. Lorsque le robot est en avance sur son plan, il est alors possible qu'il existe une meilleure solution que le plan courant en permettant de réaliser des tâches plus rapidement. Dans cette situation, il est alors souhaitable de replanifier afin de saisir des opportunités si elles existent. Cette stratégie ressemble un peu à ce qu'une personne ferait dans certaines situations où il est difficile de prévoir la durée des actions. Par exemple, si l'on doit se rendre de Sherbrooke à Montréal pour une réunion importante, on peut alors compter 100 minutes pour le trajet de base et au moins 30 minutes supplémentaires pour les imprévues, comme des travaux routiers ou une circulation plus dense. Ainsi, dans le plan, 130 minutes seront prévues pour le déplacement. Par contre, une fois arrivé à Montréal, si on est en avance, rien n'empêche de réviser le plan en insérant de nouvelles tâches avant la réunion.

Cette même stratégie est intégrée au planificateur du robot. La ligne 35 de l'algorithme 3 vérifie si l'exécution du plan courant progresse plus rapidement que ce qui est prévu. Cette vérification est faite en calculant la différence entre le temps auquel l'action courante doit se terminer (temps de l'état projeté) et l'attribut *minend* de l'action courante. Une valeur négative indique que l'exécution est en avance, tandis qu'une valeur positive signifie que l'exécution est en retard. Lorsque l'exécution du plan est en avance par un certain seuil (120 secondes) sur ce qui fut planifié, l'algorithme décide alors de générer un nouveau plan. Par contre, l'algorithme évite de replanifier lorsque la prochaine action est sur le point de commencer (voir ligne 36).

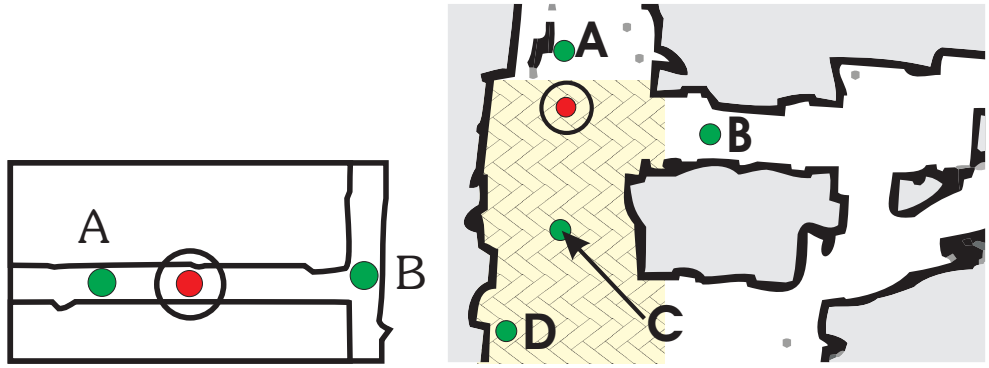
8.2 Adaptations au domaine de la conférence

Puisqu'un robot réel évolue dans un environnement dynamique et incertain, plusieurs modifications ont dû être apportées à la spécification du domaine du robot conférencier

présentée à la section 7.7.

8.2.1 Monde continu versus représentation discrète

Certaines difficultés surviennent par le fait que le robot évolue dans un monde continu alors que la représentation du planificateur est du domaine discret. Par exemple, comme présenté à la section 7.7, la position courante du robot est représentée par un symbole qui prend la valeur du point le repère le plus près du robot. Ainsi, le planificateur considère que le robot ne peut que se retrouver en un nombre fini de points, soit aux points de repère définis sur la carte. Cette discrétisation est nécessaire afin de limiter la taille de la spécification du domaine. Par contre, cette discrétisation pose des problèmes dans certains cas limites. Supposons qu'un robot soit entre les points A et B mais légèrement plus près du point A, comme présenté à la figure 8.3(a). Selon la règle du point le plus près, le planificateur considère alors que le robot est au point A même s'il ne s'y trouve pas. Ainsi, si les deux points sont assez éloignés et que le robot doit réaliser une tâche à un moment précis au point A, le robot pourrait alors arriver en retard, car il croit être déjà au point A. De façon similaire, le cas inverse pourrait s'appliquer pour le point B. Comme le planificateur pense être au point A, il croit alors qu'il doit franchir la totalité de la distance séparant les deux points, alors qu'en réalité, il n'a qu'un peu plus de la moitié à parcourir. Dans ce cas, le planificateur pourrait refuser une tâche ne parvenant pas à satisfaire les contraintes de temps.



(a) Robot entre deux points (b) Robot près d'une zone

Figure 8.3 – Robot dans des positions ambiguës.

Un autre problème se pose avec la représentation des salles. Par exemple, à la figure 8.3(b), la salle C couvre une vaste surface contrairement à la représentation interne du planificateur qui considère les lieux comme des points. Dans ce cas, le planificateur considérerait que le robot est en position A puisque A est le point le plus près du robot malgré que le robot se trouve à l'intérieur de la zone C. Enfin, d'autres cas à traiter sont les points de repère se situant à l'intérieur d'une zone, les zones imbriquées et les zones se chevauchant. Par exemple, à la figure 8.3(b), le point D se situe dans la salle C. Ainsi, le robot peut être à la fois dans la salle C et au point D.

Pour remédier à ce problème, une première solution consiste à changer la façon de représenter la position courante du robot. Chaque point de repère est dorénavant représenté par son nom, sa position et sa surface (actuellement limitée à un rectangle). Dans la classe d'état (**State**), en plus d'utiliser la variable *pos* pour la position la plus près du robot, l'ensemble *zones* est ajouté pour indiquer les zones dans lesquelles le robot se retrouve. La nouvelle définition de la classe **State** est :

Classe State

- int **time** *// temps courant de l'état*
- Location* **pos** *// point de repère le plus près du robot*
- set<Location*> **zones** *// zone(s) où se retrouve le robot*
- bool **registered** *// est-ce que le robot est inscrit à la conférence*
- double **battery** *// niveau de charge des batteries*

Au niveau des préconditions des tâches primitives (opérateurs), au lieu de vérifier la position courante du robot, l'algorithme vérifie plutôt à ce que le robot soit à l'intérieur de la zone désirée. Par exemple, pour l'opérateur *Assister(p, conf)*, la précondition *s.zones.contains(p)* remplace l'ancienne précondition *s.pos=p*. Cela donne la description de tâche suivante :

Assister(p, conf)

Préconditions :

- $s.zones.contains(p)$
- $s.time \leq minend$

Effets :

- $s.time = minend$
- $s.battery$ -= $(minend - s.time) \times w.energyPerSec$

Nous avons aussi légèrement modifié la tâche de déplacement *AllerÀ(dest)* en traitant le cas particulier où le robot est à proximité du point *dest* mais sans s'y retrouver à l'intérieur, c'est-à-dire que $state.pos = dest$ mais que $dest \notin state.zones$. Pour ce cas particulier, nous considérons une distance restante arbitraire (actuellement 3 mètres) pour atteindre la zone p. La nouvelle spécification de l'opérateur *AllerÀ(dest)* est :

AllerÀ(dest)

Calcul de la variable temporaire d :

if($s.pos$!= $dest$)
    $d = w.distances(state.pos, dest)$
else
    $d = 3$;

Préconditions :

- (aucune)

Effets :

- $s.zones = set < Location* > (dest)$
- $s.pos = dest$
- $s.time$ += $d/w.speed$
- $s.battery$ -= $d \times w.energyPerMeter$

Ces modifications sur la façon de représenter la position courante du robot et de traiter certains cas particuliers permettent d'éviter un bon nombre de problèmes. Par contre, certains problèmes demeurent, et ce, principalement quand le planificateur croit que le robot est trop loin pour aller réaliser une tâche à un temps précis alors qu'en réalité, il y est assez proche. Pour régler ces derniers problèmes, nous avons donné au robot une certaine période de grâce lui permettant d'être légèrement en retard (deux minutes). Afin d'éviter que ça ne dégénère, nous avons imposé la contrainte suivante : le planificateur peut planifier une tâche en retard seulement s'il n'existe pas d'autre plan permettant au robot d'être à l'heure. Ainsi, le robot ne pourrait pas arriver en retard à une conférence puisqu'il aurait décidé d'aller livrer un message sachant qu'une période de grâce de deux minutes est permise.

8.2.2 Points de repère inconnus sur la carte

Puisqu'un robot évolue dans un environnement dynamique et partiellement connu, les positions des points de repère sur la carte ne sont pas nécessairement toutes connues à

l'avance. Par exemple, à son arrivé au *AAAI Robot Challenge*, un robot conférencier peut se voir indiquer le nom de la salle où il doit faire sa présentation, sans forcément savoir à quel endroit cette salle se situe. Malgré cela, le robot doit tout de même être capable de planifier ses tâches afin d'arriver à temps pour sa présentation. La solution proposée consiste à introduire la nouvelle tâche *TrouverEndroit(p)*. Cette tâche est systématiquement insérée à la mission lorsqu'une tâche a pour paramètre un lieu inconnu. La tâche *TrouverEndroit(p)* est définie comme suit :

TrouverEndroit(dest)

Préconditions :

- (aucune)

Effets :

- $s.endroitsTrouves.insert(dest)$
- $s.pos$ -= $1.5/dest$
- $s.time$ -= $1.5/w.plusLongueDistance$
- $s.battery$ -= $1.5 \times w.plusLongueDistance$

Puisque nous ne connaissons pas où se situe l'endroit en question, nous ne pouvons donc pas déterminer avec précision à quelle distance il se trouve de la position courante du robot, car l'endroit peut être très près ou très loin. Afin de prévoir le pire cas, nous considérons cette distance comme étant la distance maximale dans la table des distances. De plus, comme il faut prévoir un certain temps pour trouver cet endroit, nous multiplions cette distance par 1.5 afin d'obtenir une certaine marge de sécurité. Cette stratégie doit aussi être utilisée pour l'action *AllerÀ(dest)* puisqu'une fois rendu à un endroit inconnu, le même problème se pose pour revenir à un endroit connu. Par contre, cette fois-ci, comme l'endroit est censé être trouvé, nous ne multiplions pas par le facteur de 1.5.

À l'exécution, lorsque le module de planification insère une action *TrouverEndroit()* dans le conteneur de tâches, le module motivationnel responsable de l'interface graphique attire l'attention en affichant un message de demande à l'aide à l'écran. En même temps, le robot se déplace aléatoirement pour chercher des gens prêts à l'aider. Une personne peut utiliser l'écran tactile du robot afin de lui indiquer où se trouve l'endroit recherché. Une fois la position connue, le module de planification met à jour sa table des distances et génère un nouveau plan afin d'obtenir un plan de meilleure qualité étant donné qu'il est maintenant possible d'estimer convenablement la distance à parcourir.

8.3 Discussion

L'intégration du planificateur ConfPlan dans l'architecture MBA est originale en plusieurs points [6]. Premièrement, il s'agit d'une avancée significative de l'intégration d'un planificateur dans une architecture décisionnelle robotique. Nos travaux se comparent à ceux reliés à l'intégration du planificateur Prodigy dans le robot Xavier [22]. Dans ce dernier, le planificateur sélectionne la prochaine action à exécuter en triant une liste d'actions éligibles selon des pondérations basées sur des priorités de tâches, des échéances temporelles, etc. Le planificateur Prodigy ne construisant pas de plan complet, il ne permet pas de vérifier la validité d'une mission. De plus, il ne permet pas de gérer des contraintes de temps et de garantir leur respect. En présence de contraintes temporelles, il ne fait qu'ordonner les tâches en fonction de leur proximité avec leur échéance.

Deuxièmement, nous avons proposé des solutions simples, mais efficaces pour faire le lien entre la vision symbolique du planificateur et la réalité de l'environnement du robot, qui est dynamique et imprécise. Nous avons aussi proposé des méthodes permettant de planifier en même temps que l'exécution d'un plan, et ce, tout en considérant la progression des activités courantes du robot.

Troisièmement, l'intégration de ConfPlan dans MBA est particulière puisque, contrairement aux architectures hybrides classiques, le planificateur ne joue pas un rôle central dans MBA. En effet, l'architecture MBA est conçue pour permettre à plusieurs modules motivationnels de prendre des décisions au même niveau que le planificateur. Par exemple, pour certaines tâches, d'autres sources motivationnelles peuvent s'occuper de la réalisation des tâches et court-circuiter le plan courant du module de planification. Pour ces raisons, nous avons mis en place des mécanismes de validation de plan et de surveillance de la progression des tâches afin que le planificateur puisse corriger son plan lorsque d'autres motivations exécutent des tâches à la place du planificateur. Cet aspect étant plus associé à l'architecture MBA, il n'est pas étudié davantage dans ce mémoire.

Une partie de ces idées d'intégration ont été présentées à la conférence AAAI'05 [6].

CHAPITRE 9

Expérimentations avec le robot Spartacus

Les expérimentations ont été réalisées à l'aide du robot Spartacus, (voir figure 9.1), un robot mobile entièrement conçu à l'Université de Sherbrooke. Ce dernier est entre autres équipé d'un capteur de proximité à balayage laser (*laser range finder*), d'un ordinateur embarqué Mini-ITX (processeur Pentium M 1.7 GHz et mémoire vive de 512 Mo) et d'un écran tactile. Présentement, Spartacus a une autonomie d'énergie variant entre 45 et 90 minutes selon l'utilisation de ses ressources. Sa vitesse de déplacement moyenne est d'environ 130 millimètres par seconde.

MARIE [11], un environnement d'intégration de logiciels robotiques, est utilisé pour l'assemblage des différents modules de contrôle du robot. RobotFlow[1] est utilisé pour la programmation des modules comportementaux. De plus, les modules de localisation et de navigation de CARMEN[2], une boîte à outils initialement développée par le laboratoire de robotique au *Carnegie Mellon University*, sont utilisés pour leurs fonctions respectives.

Deux séries de tests ont été réalisées, la première au *AAAI Robot Challenge 2005* et l'autre dans une conférence simulée dans les corridors du 5^e étage de la Faculté de génie

[1] http ://robotflow.sourceforge.net/
[2] http ://carmen.sourceforge.net/

Figure 9.1 – Le robot Spartacus.

de l'Université de Sherbrooke.

9.1 AAAI Robot Challenge 2005

En juillet 2005, notre équipe s'est rendue à Pittsburgh, en Pennsylvanie aux États-Unis, pour la *National Conference on Arfiticial Intelligence*. À cette conférence, nous avons participé au *AAAI Robot Challenge* (voir figure 9.2). À cette occasion, nous avons fait la démonstration de plusieurs capacités du robot Spartacus, dont sa capacité de planifier et de coordonner ses tâches.

Notre robot était l'un des premiers participants à être équipé d'un planificateur de tâches pour la gestion de ses missions, puisque les précédents [33, 39] ont principalement utilisé des machines à états finis pour la coordination des tâches.

Les démonstrations effectuées ont eu lieu dans l'espace autour des salles de conférence de l'hôtel du *Westin Convention Center* de Pittsburgh. La figure 9.3) illustre la carte générée avec le robot pour cet environnement.

Figure 9.2 – Notre équipe au AAAI Robot Challenge.

9.1.1 Démonstration

Au niveau des capacités de planification du robot, notre meilleure démonstration fut celle faite lors de la troisième journée de la conférence, soit le 12 juillet 2005. Lors de cette démonstration, au temps *t=00m00s*, le robot démarre dans l'aire centrale située entre les points A et B et il sait qu'il n'est pas inscrit à la conférence. Il ne sait pas encore où sont situés les points A et B sur la carte. Sa mission initiale est d'assister à une conférence fictive au temps *t=15m00s* à l'endroit A. Le planificateur génère alors un premier plan qui consiste à s'inscrire, demander de l'aide pour trouver le lieu A, aller à A et assister à l'exposé. Au temps *07m39s*, le robot termine son inscription et se voit donner une nouvelle tâche, soit de livrer un message du point A au point B. À ce moment, en générant un nouveau plan, le planificateur ajoute les actions suivantes au plan : aller à A, prendre le message, aller à B et livrer le message. Le robot a suivi correctement ce plan jusqu'à la dernière action, où il y a eu une panne logicielle.

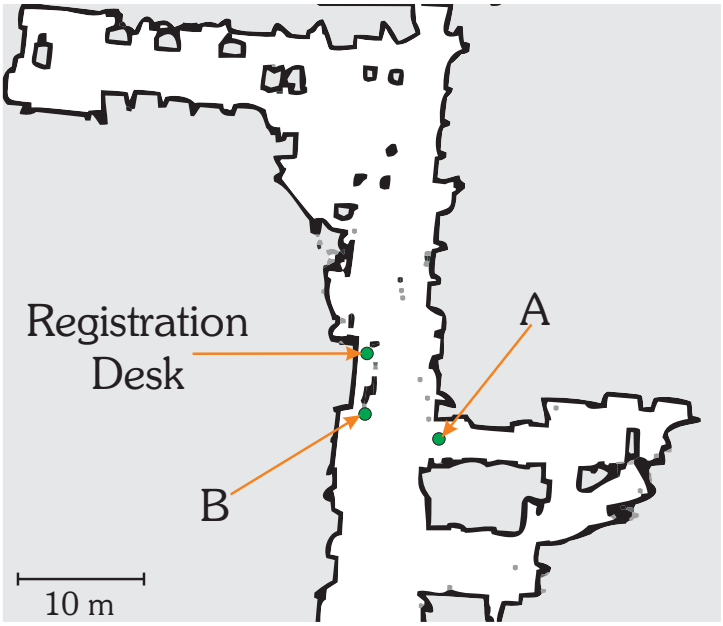


Figure 9.3 – Carte pour le AAAI Robot Challenge au Westin Convention Center.

9.1.2 Difficultés rencontrées

Une première difficulté rencontrée fut la disponibilité des juges pour évaluer notre démonstration. Afin de bien démontrer les capacités de planification du robot, il est nécessaire de lui donner une mission ayant une complexité minimale, c'est-à-dire de lui donner des missions d'au moins quatre à cinq tâches et s'étalant sur au moins 30 à 45 minutes. De plus, nous devions garder un certain contrôle sur l'expérience afin d'y faire ressortir des éléments intéressants comme des situations où le robot a à réorganiser sa mission pour respecter les contraintes soumises. Puisque le robot était fréquemment interrompu pour diverses raisons, comme des gens touchant à l'écran tactile, ce dernier avait souvent à replanifier sa mission. Il était donc difficile de comparer le plan initial avec le plan courant. Enfin, en raison de ces détails d'organisation, nos démonstrations n'ont pu être convaincantes au niveau des capacités de planification de tâches.

Paradoxalement, une autre difficulté majeure de notre démonstration fut le grand intérêt des gens présents à la conférence. Pendant les sorties du robot, beaucoup de gens l'approchaient et se mettaient sur son chemin pour mieux l'observer. Or, lorsque beaucoup d'individus sont près de lui, le robot a de la difficulté à naviguer normalement puisqu'il

oscille entre la navigation et l'évitement d'obstacles. De plus, la forte densité de personnes devant le robot rend d'autant plus difficile sa localisation puisque le capteur de proximité à balayage laser n'arrive plus à capter les murs de l'environnement. Puisque le module de localisation n'arrive plus à trouver des propriétés caractéristiques dans l'environnement, ce dernier a tendance à perdre la position estimée du robot. De façon plus technique, le filtre à particules du localisateur devient de plus en plus ambigu et les particules ont tendance à s'étaler dans un nuage devenant de plus en plus grand. Bref, après un certain temps, le robot n'est plus à l'endroit où il pense être et les déplacements qu'il effectue ne sont plus appropriés. Pour ces raisons, le robot n'arrive plus à naviguer correctement et, dans des cas limites, il peut effectuer des tâches aux mauvais endroits.

Pour mieux comprendre, la figure 9.4 illustre le parcours du robot entre les points A et B. Les petits ronds de couleurs indiquent les positions estimées du robot sur son parcours. Ceux en rouges appartiennent au trajet emprunté par le robot pour se rendre du bureau d'inscription au point A et ceux en bleus représentent le trajet de A à B. Malgré que le navigateur du système tente de diriger le robot en ligne droite, le robot n'a pu suivre une trajectoire droite puisque des gens se trouvaient sur son chemin. De plus, les points sur la figure étant des positions estimées et non des positions réelles, à chaque fois que le localisateur change son estimation de façon trop brutale, cela se répercute sur le comportement de suivi de chemin (*Goto*) qui doit alors réaligner, tout aussi brutalement, le robot avec la cible courante. Ceci s'illustre bien en examinant les variations de vitesse estimée par le localisateur montrées à la figure 9.5.

On peut aussi analyser l'évolution de la vitesse estimée du robot par le localisateur, tel que montré sur le graphique de la figure 9.5. On peut y constater que la vitesse instantanée, illustrée en gris pâle, varie très fortement. Même la vitesse moyenne, calculée sur deux secondes, a aussi tendance à varier et souvent dépasser la vitesse maximale réelle du robot qui est d'environ 0.13 m/s. La même analyse pourrait être faite sur l'orientation estimée du robot. Cela explique pourquoi le robot avait tendance à se déplacer en zigzaguant.

D'autres difficultés rencontrées étaient au niveau de la fiabilité logicielle. Puisque le robot est contrôlé par un système complexe, basé sur plusieurs modules logiciels répartis sur

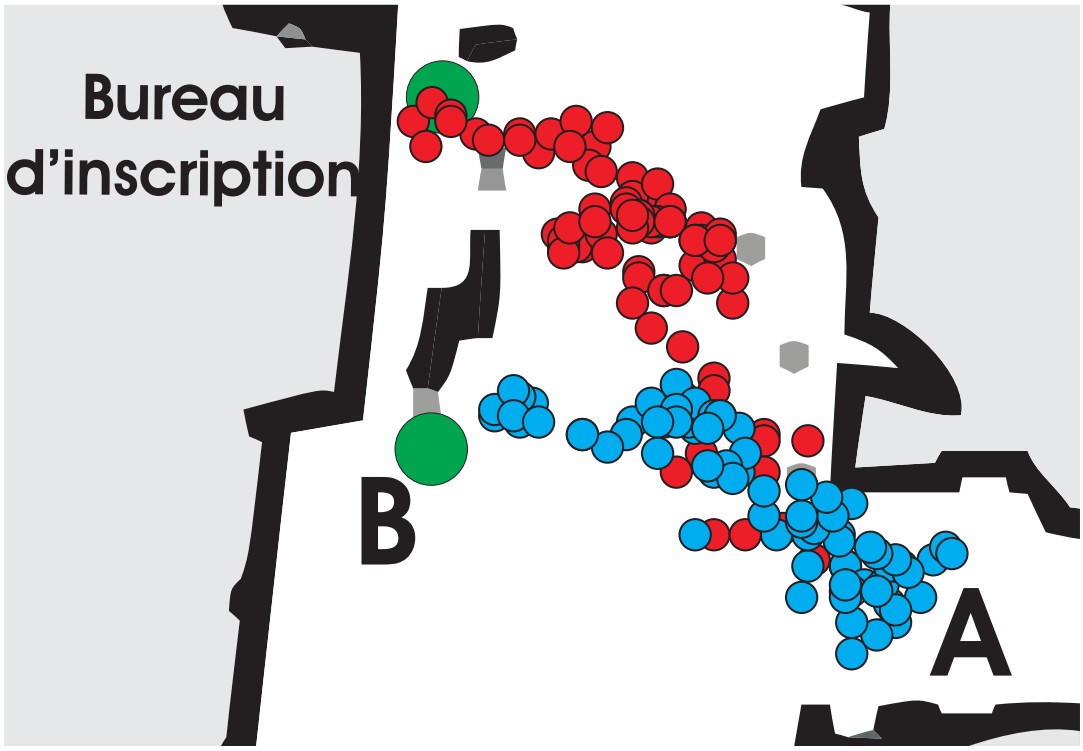


Figure 9.4 – Positions estimées du robot au AAAI Robot Challenge.

plusieurs ordinateurs portables, et puisque le temps de préparation fut très limité, alors des problèmes techniques étaient difficiles à éviter. Parfois, le robot cessait subitement de fonctionner suite à une panne logicielle, ce qui posait de nombreux problèmes pour réaliser des expériences et des démonstrations intéressantes.

9.2 Conférence simulée avec le robot

Plus récemment, en février 2006, une nouvelle série de tests d'intégration a été réalisée au 5^e étage de la Faculté de génie de l'Université de Sherbrooke (voir la carte à la figure 9.6). Lors de ces tests, Spartacus avait à réaliser des tâches de livraison de messages, de surveillance de pièces et d'assistance à des présentations fictives.

Puisque l'environnement de test était mieux contrôlé qu'à la conférence AAAI'05, le robot a pu fonctionner plus normalement. Il est beaucoup plus facile pour un robot de se localiser dans des corridors avec peu de personnes que dans un espace vaste avec

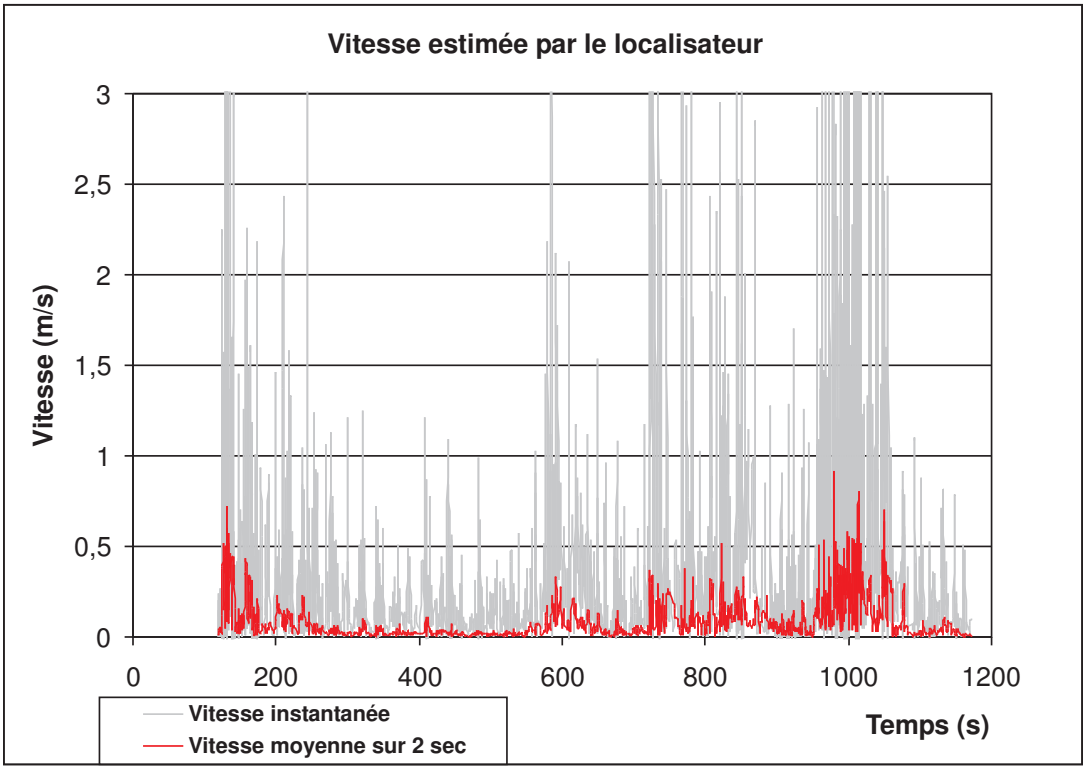


Figure 9.5 – Vitesse estimée du robot au AAAI Robot Challenge.

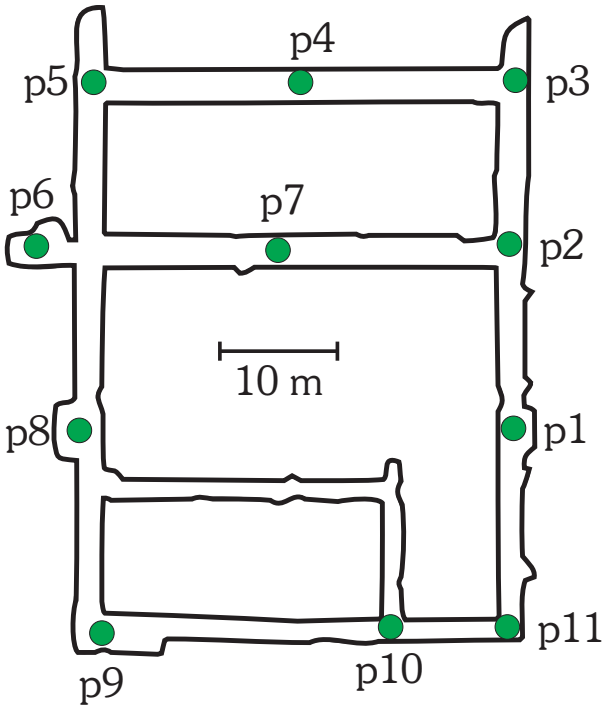


Figure 9.6 – Carte du 5^{e} étage avec 11 points de repère.

beaucoup de gens. De plus, des améliorations ont été apportées au niveau du paramétrage du comportement de suivi de chemin (*Goto*). À la figure 9.7, on peut remarquer que les positions estimées sont plus linéaires et stables comparativement à l'expérience du *AAAI Robot Challenge*. Enfin, la stabilité de la plateforme logicielle a aussi été grandement améliorée, ceci facilitant l'exécution des tests.

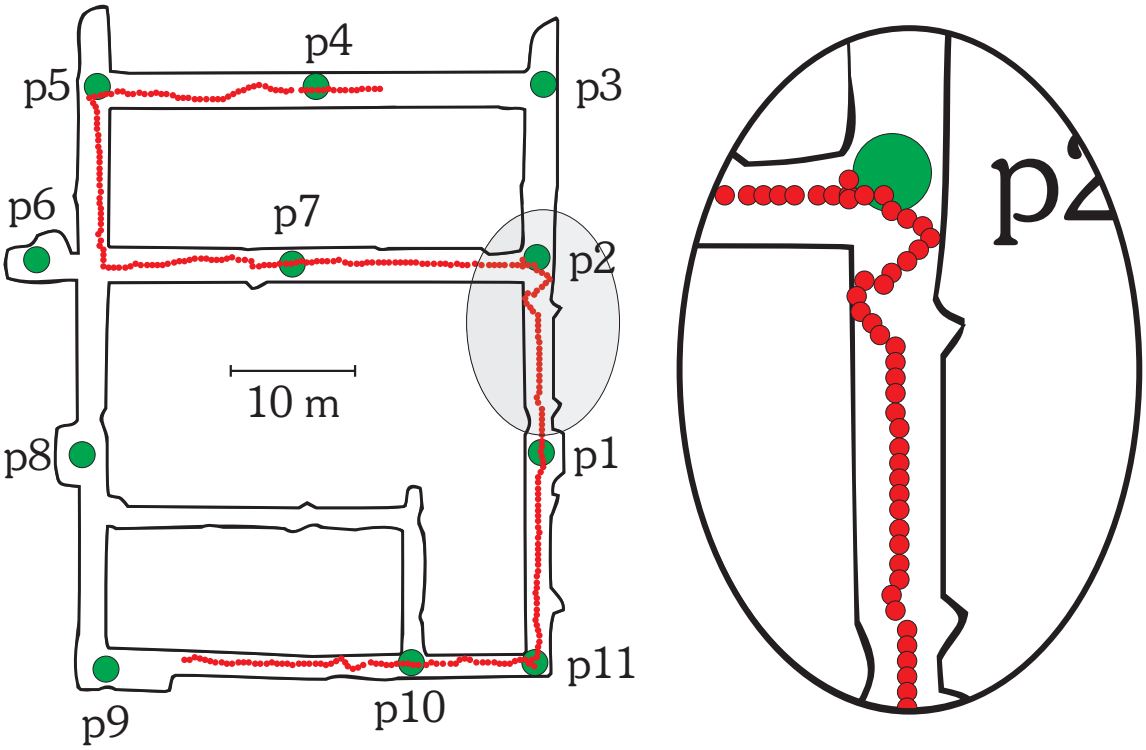


Figure 9.7 – Trace du robot lors d'une expérience au 5^e étage.

L'objectif de cette présente section est de mettre l'accent sur les diverses fonctionnalités de ConfPlan ainsi que sur les solutions de son intégration dans l'architecture décisionnelle MBA. Pour cela, plusieurs scénarios d'expérimentation ont été élaborés pour mettre ses fonctionnalités en valeur. Chacun de ces scénarios a été testé plusieurs fois dans l'environnement simulé[3], et puis validé au moins une fois avec le robot Spartacus. Pour un même scénario, les résultats peuvent varier d'un test à l'autre selon plusieurs facteurs comme le niveau de circulation dans les corridors, la présence d'obstacles différents et notre vitesse d'exécution pour donner les tâches et les messages au robot. Généralement, les variations d'un test à l'autre ne sont pas significatives et se limitent souvent à de

[3]Avec le simulateur Stage (http ://playerstage.sourceforge.net/

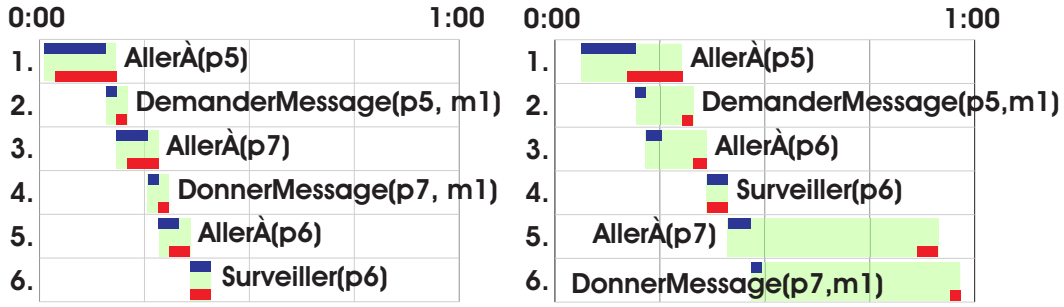
légères fluctuations dans les temps d'exécution des diverses étapes des plans. Par contre, dans de rares cas, l'ordre d'exécution des actions a changé d'un test à l'autre, par exemple, en raison d'une navigation trop lente pouvant être causée par la présence de personnes ou d'obstacles dans les corridors.

9.2.1 Violation de contraintes temporelles

Le robot est initialement placé à l'endroit $p1$ au temps *t=00m00s* et reçoit la mission suivante au temps *t=01m11* : livrer un message $m1$ de $p5$ à $p7$; surveiller l'endroit $p6$ au temps *t=22m00s* pour cinq minutes. Au départ, le planificateur génère le plan illustré à la figure 9.8(a). Comme l'illustrent les petits rectangles dans le plan, les contraintes de temps de ce dernier sont assez serrées. Au commencement de l'expérience, le chemin du robot a été volontairement obstrué pour simuler une situation imprévue où le robot est ralenti. Puisque sa vitesse effective est alors réduite, le robot prend ainsi un certain retard sur son plan initial. Au temps *t=03m50s*, en analysant la distance restante calculée par le navigateur, le module de planification met à jour l'état projeté (voir section 8.1.3) et détecte que son plan est invalide puisqu'il ne pourra arriver à temps pour surveiller la pièce $p6$. En réaction à cette situation, le planificateur génère un nouveau plan, celui présenté à la figure 9.8(b). Puisque le planificateur planifie en même temps que l'exécution et tient compte de la progression de l'action courante (voir section 8.1.2), l'action *AllerÀ(p5)* du nouveau plan a une durée inférieure à ce qu'elle avait dans le premier plan. Pour le reste de cette expérience, le chemin du robot n'étant pas obstrué, le robot a terminé sa mission correctement comme illustré à la figure 9.9.

9.2.2 Détection d'opportunisme

Pour ce test, le robot reçoit la mission suivante : livrer trois messages ($p3 \overset{m1}{\rightarrow} p10$, $p5 \overset{m2}{\rightarrow} p2$ et $p7 \overset{m3}{\rightarrow} p9$) et surveiller l'endroit $p8$ au temps *t=36m00s* pour 10 minutes. Dans un monde idéal où le robot avancerait toujours à sa vitesse optimale, ce dernier aurait le temps de livrer les trois messages avant d'aller surveiller la pièce $p8$. Comme expliqué à



(a) Premier plan à *t=01m11s*

(b) Deuxième plan à *t=03m50s*

Figure 9.8 – Plans pour l'expérience de violation de contraintes temporelles.

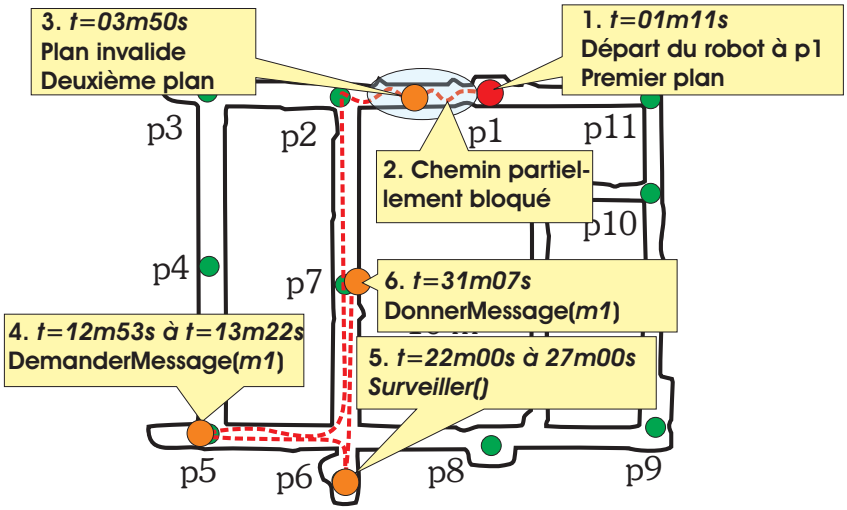
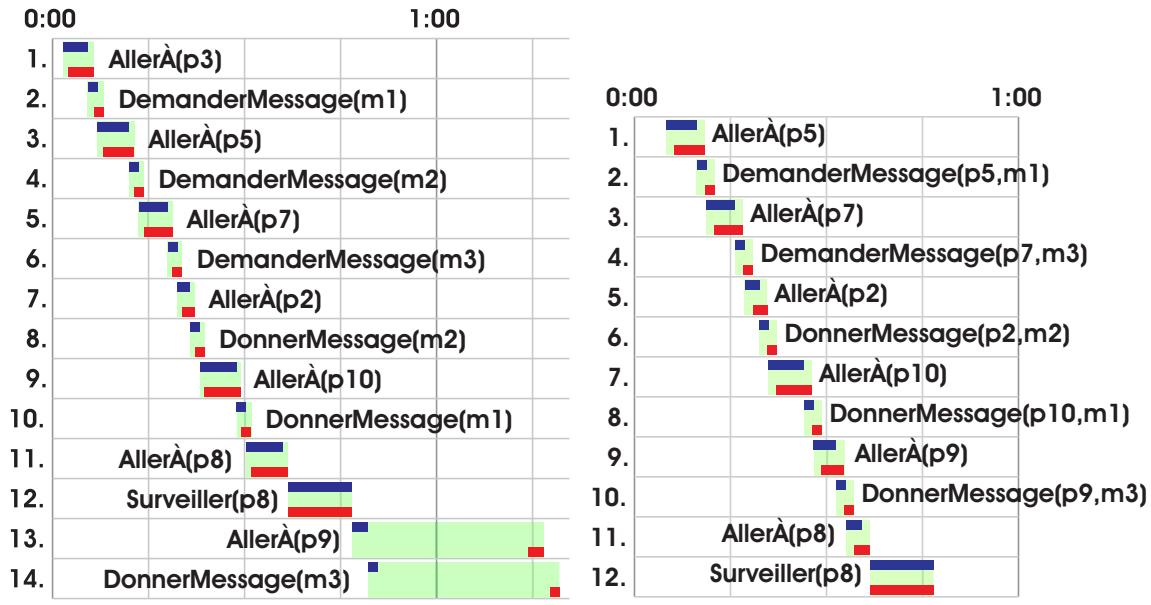


Figure 9.9 – Trace d'exécution pour l'expérience de violation de contraintes temporelles.

la section 8.1.5, nous avons pris une hypothèse prudente quant à la vitesse du robot en la fixant à 0.12 m/s dans la spécification du domaine, alors que la vitesse réelle est de 0.13 m/s. En raison de cette sous-estimation de la vitesse, le planificateur ne peut générer un plan parfaitement optimal conformément aux réelles capacités du robot. Pour la présente mission, il génère alors le plan illustré à la figure 9.10(a).

Le robot débute en se dirigeant vers l'endroit $p3$ et demande le message $m1$. Il le reçoit très rapidement, c'est-à-dire en moins de 20 secondes. Dans la spécification du domaine, la durée de l'action de demander un message est fixée à 90 secondes. Puisque le message a été reçu plus rapidement que prévu et que la navigation jusqu'à $p3$ a été plus rapide, le robot est maintenant en avance d'environ deux minutes sur son plan initial. Ainsi, au temps *t=05m10s*, avant de passer à l'étape #3, la boucle de planification réactive détecte



(a) Premier plan à *t=00m41s* (b) Deuxième plan à *t=04m20s*

Figure 9.10 – Plans pour l'expérience de détection d'opportunisme.

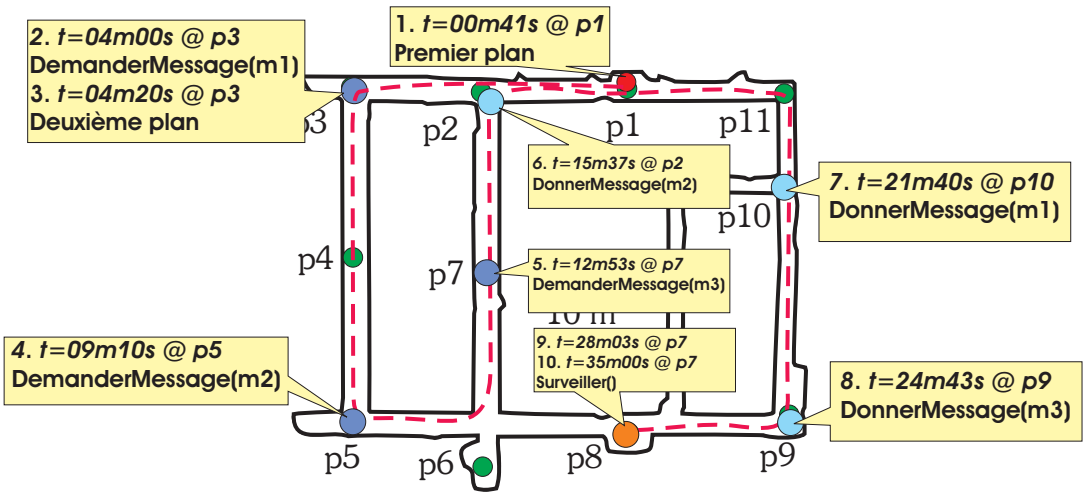


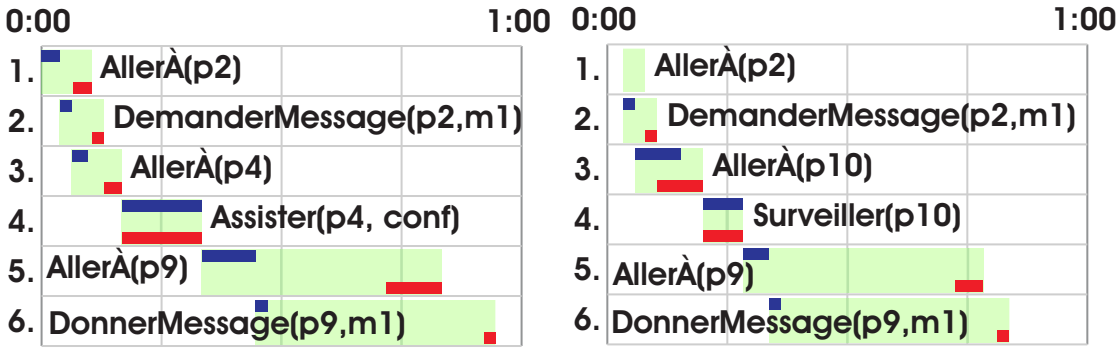
Figure 9.11 – Trace d'exécution pour l'expérience de détection d'opportunisme.

que l'exécution du plan est en avance (voir section 8.1.6). Dans ce cas, puisque l'existence d'un meilleur plan est possible, le planificateur est relancé. Le nouveau plan (voir figure 9.10(b)) prévoit alors de livrer tous les messages avant de procéder à la surveillance de $p8$. Pour la suite de l'expérience, l'exécution s'est déroulée de façon conforme à ce plan, comme illustré à la figure 9.11. Cette séquence d'exécution est optimale conformément au critère énoncé à la section 7.7.3 qui demande au planificateur de réduire au minimum

les déplacements et les délais d'attente pour la livraison des messages.

9.2.3 Simplification de missions

Le robot est initialement à $p1$ et reçoit la mission suivante : assister à une conférence à l'endroit $p4$ au temps *t=10m00s* pour 10 minutes et livrer un message de $p2$ à $p9$. Le planificateur génère alors le plan montré à la figure 9.12(a).



(a) Premier plan à *t=00m00s* (b) Deuxième plan à *t=02m08s*

Figure 9.12 – Plans pour l'expérience de la simplification d'une mission.

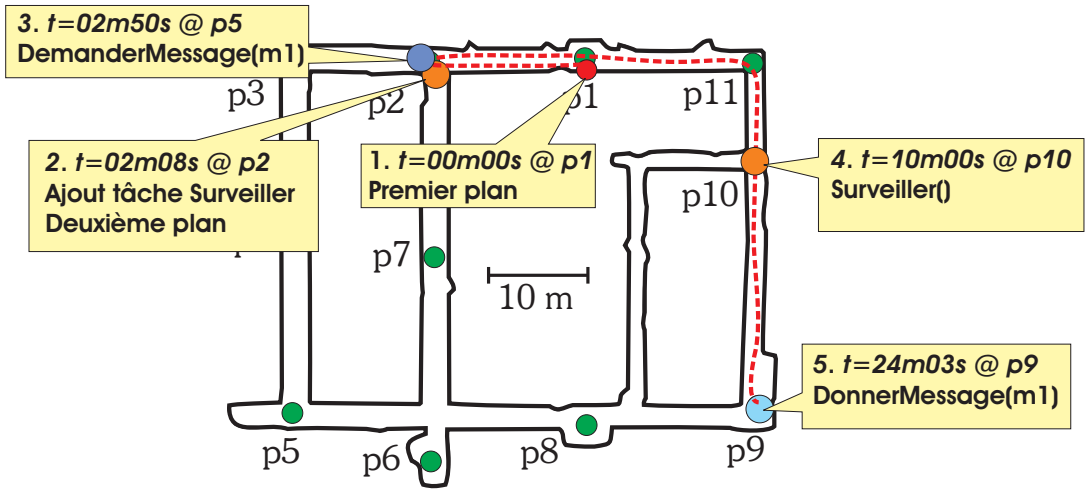


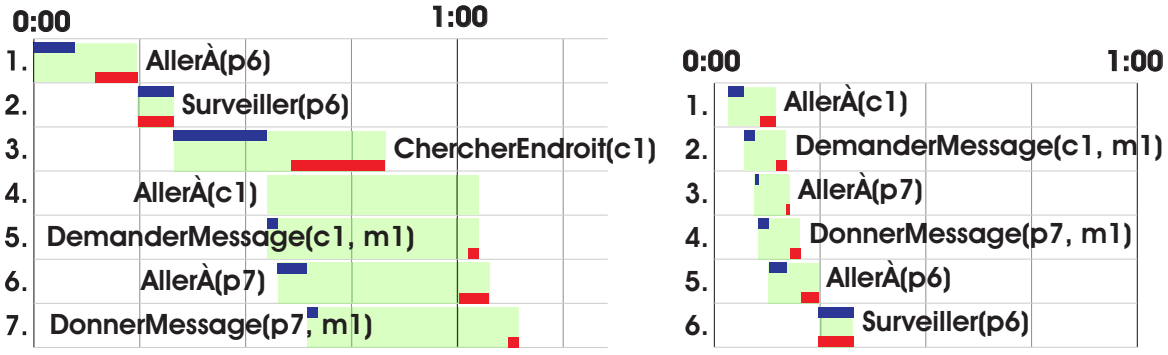
Figure 9.13 – Trace d'exécution pour l'expérience de la simplification d'une mission.

Au temps *t=02m08s*, le robot arrive à $p2$ et reçoit une nouvelle tâche de surveiller l'endroit $p10$ au temps *t=12m00s*. Un nouveau plan doit être généré. Comme décrit dans la section 7.2, le planificateur commence par vérifier la faisabilité de la mission et détecte

les tâches mutuellement exclusives. Alors, pour toutes paires de tâches possibles de la mission, le planificateur tente de générer un plan. Pour la paire *{AssisterPrésentation(p4, 10m00s, 10m00s), SurveillerEndroit(p10, 12m00s, 5m00s)}*, il n'existe pas de plan possible puisque ces deux tâches sont en conflit d'horaire. Pour résoudre ce problème, le planificateur décide d'annuler la tâche d'assister à une présentation puisqu'elle est moins prioritaire que de surveiller un endroit et génère le plan affiché à la figure 9.12(b). Ce plan permet de compléter la séquence d'exécution montrée à la figure 9.13.

9.2.4 Planification avec des endroits inconnus

L'exécution de missions avec des endroits inconnus est un excellent exemple du bénéfice engendré par la combinaison des capacités de planification avec d'autres modules décisionnels. Pour cette expérience, le robot reçoit la mission d'aller surveiller la pièce $p6$ et de livrer un message de $c1$ à $p7$. Le point $c1$ étant non référencé sur la carte, le robot doit demander de l'aide pour le trouver. Comme il ne connaît pas la position de ce point, le planificateur n'est pas en mesure de faire une bonne estimation sur comment atteindre ce point et sur le temps requis pour y arriver. Alors, le planificateur assume le pire cas, soit que le point $c1$ est très loin (voir section 8.2.2). Par conséquent, le plan de la figure 9.14(a) est obtenu, spécifiant qu'il n'est pas possible d'aller prendre et livrer le message avant la tâche de surveillance.



(a) Premier plan à *t=0m00s*

(b) Deuxième plan à *t=2m10s*

Figure 9.14 – Plans pour l'expérience avec un endroit inconnu.

Le robot commence par exécuter le plan en se dirigeant vers l'endroit $p6$ afin d'aller sur-

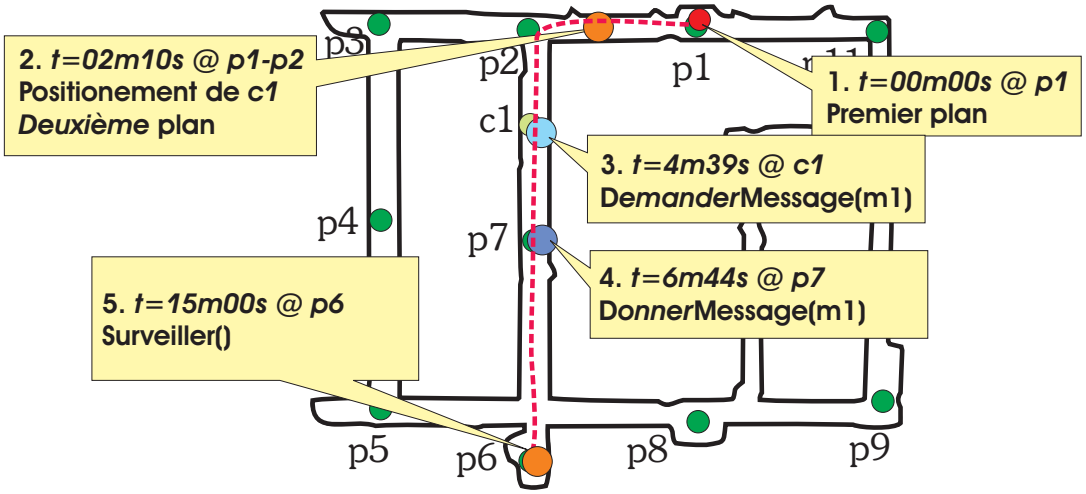


Figure 9.15 – Trace d'exécution pour l'expérience de la simplification d'une mission.

veiller cet endroit. Pendant ce temps, dans l'architecture MBA, le module motivationnel d'instinct s'intéresse à trouver l'endroit $c1$. Pour y arriver, ce module ajoute une tâche de demande d'aide dans le système, ce qui déclenche un indicateur visuel sur l'écran du robot. Quelques instants plus tard, alors que le robot est entre les points $p1$ et $p2$, une personne à proximité du robot vient l'aider. Elle informe le robot de la position du point $c1$ en cliquant sur la carte affichée sur son écran tactile. Puisqu'un nouveau point est maintenant connu, le module de planification décide de générer un nouveau plan, soit celui illustré à la figure 9.14(b). En suivant ce dernier, le robot arrive à ramasser le message à $c1$ et à le livrer à $p6$ avant la surveillance de l'endroit $p6$ dans les temps prescrits, comme illustré à la figure 9.15.

9.2.5 Performances de planification

Pour tous les tests précédents avec le robot, les performances du planificateur (voir tableau 9.1) sont très bonnes et les plans furent tous générés sous la barre des 100 millisecondes. Puisque les temps de planification sont inférieurs au temps minimum de planification, les plans générés sont tous optimaux conformément à la spécification du domaine (voir la section 7.7.3). Par contre, comme l'autonomie d'énergie du robot est faible et que la recharge automatique n'est pas encore implantée, les missions testées sont d'une

complexité très faible au niveau de ce qu'elles exigent pour être planifiées. Pour mieux mettre à l'épreuve le planificateur, il faudrait augmenter l'autonomie d'énergie du robot et intégrer la capacité de recharge du robot (voir section 10.5) afin de gérer des missions plus complexes, c'est-à-dire des missions ayant une dizaine de tâches contrairement aux tests présentés qui n'ont jamais plus de 4 tâches. Cela permettrait de mieux valider l'aspect en tout temps (*anytime*) du planificateur puisqu'il arriverait des situations où il ne serait pas possible d'obtenir une solution optimale dans les délais de planification prescrits. Par contre, les tests présentés à la section 7.8 laissent entrevoir que des missions de taille raisonnable (jusqu'à 10 tâches) seraient très bien gérées par ConfPlan.

Tableau 9.1 – Temps de planification en milliseconde.

	Expériences				
Statistiques	Contraintes temporelles	Détection d'opportunisme	Simplification de mission	Avec endroit inconnu	AAAI Challenge 2005
N	23	15	11	13	12
Minimum	0,41	0,42	0,41	0,45	0,15
Maximum	53,38	29,09	2,11	24,16	12,93
Moyenne	11,57	3,91	0,71	5,41	1,26
Écart-type	17,14	7,52	0,47	7,88	3,68

Somme toute, les tests présentés dans ce chapitre montrent que l'intégration de ConfPlan dans l'architecture MBA est réussie et que notre planificateur est capable de coordonner correctement le fonctionnement du robot en lui permettant de gérer efficacement des tâches structurées. Cette intégration nous a en partie permis de participer au *AAAI Robot Challenge* de 2005 et elle sera aussi mise à contribution en 2006.

CHAPITRE 10

Travaux futurs et améliorations à apporter

Au cours de nos travaux, de nombreuses idées d'améliorations ont été notées pour résoudre des problèmes que nous avons identifiés ou pour répondre à d'autres besoins auxquels nous n'avions pas initialement pensé. Ce chapitre présente les principales.

10.1 Localisation active

Comme indiqué au chapitre précédent, à l'occasion du *AAAI Robot Challenge 2005*, un problème fréquent est la difficulté pour le robot de se localiser en présence d'une foule. Jusqu'à présent, lors de tous les tests réalisés avec le robot, ce dernier effectuait une localisation passive. En d'autres mots, le localisateur estime la position courante du robot en interprétant les données perçues par les capteurs sans générer de commandes motrices. Pour améliorer la localisation du robot, une localisation active [17] peut être utilisée. Contrairement à une localisation passive, une localisation active peut influencer le contrôle du robot afin d'améliorer le niveau de certitude de la position estimée du robot.

Intuitivement, on peut faire un parallèle avec une personne qui est perdue dans une ville

et avec une carte en main. Avec une approche passive, cette personne va se promener un peu aléatoirement jusqu'à tant qu'elle se retrouve, tandis qu'avec une approche active, elle va tenter d'effectuer des déplacements vers des endroits stratégiques lui permettant de mieux se localiser, comme aux coins de rue. Pour un robot, ce dernier ayant lui aussi une carte de son environnement, il va tenter de maximiser son niveau de certitude de l'endroit où il pense être en essayant de se déplacer à des endroits lui permettant d'observer des caractéristiques dans l'environnement, comme des angles dans la configuration des pièces.

Le but d'un robot n'étant pas uniquement de se localiser, mais bien de réaliser des tâches, une solution de localisation active doit être intégrée à une architecture d'exécution de tâches. Bien que très peu de travaux aient été réalisés dans ce domaine, une solution simple est d'alterner le mode de localisation active d'avec le mode d'exécution de tâches [8]. Avec cette approche, lorsque le robot se considère perdu, il interrompt l'exécution de son plan courant et bascule en mode de localisation active. Il tente alors de se localiser en se déplaçant à des endroits stratégiques lui permettant d'augmenter son niveau de confiance sur sa position estimée, mais cela de façon indépendante des tâches à réaliser.

Bien que cette solution fonctionne, elle n'est pas optimale puisqu'elle ne tient pas compte des tâches courantes du robot. Intuitivement, cette situation peut se comparer à une personne roulant en automobile et qui se perd en se déplaçant être deux villes éloignées. Avec cette approche, le conducteur basculerait alors en mode de localisation active et tenterait de trouver le meilleur point de repère à proximité. Or, se déplacer à ce point pourrait signifier de faire un demi-tour pour revenir en arrière, et une fois localisé, de refaire demi-tour pour continuer le chemin vers la destination. En plus d'entraîner un demi-tour, cette stratégie de localisation pourrait causer des oscillations non souhaitées. Par exemple, imaginons que notre conducteur passe par un endroit difficile et s'y perd facilement. À cet instant, il ferait possiblement demi-tour jusqu'à un point de repère précédent. Le problème est qu'une fois localisé, le conducteur risque de réemprunter le même chemin que celui dans lequel il s'est perdu et de s'y perdre à nouveau. Dans le pire cas, avec cette stratégie, le conducteur va boucler indéfiniment sur le même parcours.

Une meilleure approche que nous avons commencé à explorer est de faire de la localisation active tout en considérant les tâches à accomplir. Dans l'exemple précédent, au lieu de tenter de rejoindre le point de repère le plus près, notre conducteur pourrait prendre une décision plus audacieuse. S'il est perdu, il peut tout même savoir en gros quels sont les endroits auxquels il peut se retrouver, et dans quelle direction il se dirige. Ainsi, au lieu d'atteindre le point de repère le plus près, il pourrait continuer dans la même direction sachant que des points de repère légèrement plus loin lui permettront de déduire sa position tout en se rapprochant du but courant.

Pour donner cette capacité à un robot, nous avons pris comme point de départ les travaux portant sur la planification sous hypothèse dans des domaines non déterministes et partiellement observables [2]. L'idée est de générer des plans forts (*safe plan*), c'est-à-dire des plans garantissant l'atteinte d'un but, sous certaines hypothèses. Dans le cas présent, nous faisons l'hypothèse que le robot se trouve dans une position parmi un ensemble fini de positions possibles. Pour estimer ces positions, nous avons modifié le module de localisation de CARMEN [40] afin qu'il estime simultanément plusieurs positions possibles plutôt qu'une seule. Les travaux initiés sont présentés dans [1].

10.2 Planification concurrente

Une autre avenue à explorer est de donner au robot la capacité de planifier des tâches concurrentes dans le but d'exécuter des tâches en parallèle afin d'être plus efficace. Actuellement, les plans générés par ConfPlan sont totalement séquentiels. Dans certaines situations, le robot a la possibilité d'exécuter plusieurs tâches. Par exemple, lorsque le robot se déplace d'un endroit à un autre, il pourrait tenir une conversation avec personne qui le suit sur son chemin, diffuser une présentation ou même exécuter une tâche de surveillance. Pour cela, il serait possible d'intégrer une approche de planification concurrente à chaînage avant [4] dans un planificateur de type HTN [42].

10.3 Modèle probabiliste du temps

À présent, le domaine de planification utilisé par ConfPlan est complètement déterministe. Comme nous l'avons expliqué à la section 8.1.5, nous avons spécifié dans le domaine une vitesse moyenne prudente afin d'éviter le non-respect de contraintes de temps lorsque le robot est légèrement ralenti. Or, au cours d'une longue mission, cette hypothèse prudente peut avoir pour effet d'entraîner des abandons de tâches non souhaités puisque le robot prévoira ne pas avoir le temps de réaliser toute sa mission.

L'utilisation d'un modèle probabiliste sur la durée des actions et sur les ressources pourrait être bénéfique tout en conservant la nature déterministe de tous les autres aspects du domaine. Par exemple, la vitesse de déplacement du robot pourrait être modélisée à l'aide d'une fonction de densité comme la loi normale. Pour un plan donné, il serait alors possible de calculer la probabilité que les contraintes de temps et de ressources soient respectées pour toutes les étapes du plan. Ainsi, au lieu de spécifier une vitesse prudente du robot, un niveau de certitude minimal serait plutôt fixé. Si ce niveau n'est pas atteint pour une partie du plan trouvé, l'existence d'un plan de secours plus sûr pourrait être vérifiée. Bref, des plans conditionnels pourraient être construits, permettant d'utiliser des sous-plans efficaces et moins sûrs, tout en gardant la possibilité de basculer vers d'autres sous-plans plus sûrs et moins efficaces lorsque des situations critiques se présentent.

10.4 Planification de tâches passives

Dans ce travail, toutes les missions à planifier sont composées de tâches pouvant être qualifiées de tâches structurées, c'est-à-dire des tâches dont leurs critères de réalisation sont clairement définis. Par exemple, la livraison d'un message se fait en allant premièrement prendre le message pour ensuite le donner à destination, et à la fin, le destinataire doit posséder le message. Ce type de tâche se planifie de façon naturelle puisqu'il est facile de spécifier des préconditions, des effets pour chacune de leurs sous-tâches, et ces sous-tâches sont exécutées par bloc sans être subdivisées.

Par contre, il y a d'autres types de tâches qui sont plus difficiles à planifier, comme des tâches de nature plus passive. Par exemple, un robot conférencier pourrait avoir à circuler librement dans l'environnement lors de ses moments libres. Cela ressemble un peu à comment nous nous comportons lors d'événements comme des conférences ou des congrès. Un autre exemple serait d'animer un kiosque durant une journée. Cette tâche peut être interrompue puisqu'il n'est pas nécessaire d'être présent au kiosque toute la journée. Si le robot doit réaliser d'autres tâches, il pourrait interrompre brièvement son animation pour effectuer ses nouvelles tâches pour enfin retourner à son kiosque.

Si de telles tâches font partie de la mission donnée au robot, elles devront être prises en considération au moment de la planification puisqu'elles peuvent avoir un impact sur les plans. Par exemple, lorsqu'il est permis au robot de circuler librement dans ces temps libres, cela consomme inévitablement des ressources. Donc, le planificateur doit en tenir compte afin de planifier des recharges au besoin ou peut-être même de prévoir des moments de repos afin d'éviter que le robot consomme trop de ressources en circulant librement, ce qui pourrait compromettre d'autres tâches plus importantes.

Les tâches de ce type sont plus difficiles à planifier puisqu'elles ne se décomposent pas en blocs fixes de la même manière que des tâches de déplacement, de livraison de messages ou de prise de photos. De plus, ces tâches n'ont pas de préconditions et de résultats clairement établis. Enfin, ces tâches peuvent être interrompues en tout moment et reprises plus tard.

10.5 Courbes de décharge des batteries

Bien que le planificateur ConfPlan soit capable de planifier des actions de recharge, cette capacité n'a pu être intégrée au robot. Comme une batterie ne se recharge pas et ne se décharge pas de façon linéaire, il faut caractériser ses courbes de chargement et de déchargement afin d'être en mesure d'estimer la quantité d'énergie restante [38]. Puisque ces courbes n'ont pas encore été modélisées pour le robot Spartacus, le planificateur ne peut donc pas estimer avec précision de la quantité d'énergie restante dans les batteries.

Pour cette raison, il n'est donc pas possible de planifier efficacement des plans demandant au robot de se brancher à une prise électrique.

10.6 Approche générique

Comme nous l'avons vu à la section 7.6, ConfPlan est entièrement programmé en C++, incluant la spécification du domaine. Afin d'être plus souple et complètement indépendant du domaine, il serait intéressant de développer un langage de spécification de domaine, un peu comme PDDL. Ce travail permettrait à ConfPlan d'être complètement générique. Il serait alors plus facile d'expérimenter d'autres domaines et d'effectuer des ajustements rapidement dans ces derniers, puisque la phase de compilation serait évitée. De plus, une approche générique permettrait d'explorer d'autres applications que la robotique mobile. Par exemple, le planificateur ConfPlan pourrait être intégré dans des jeux vidéo, ce qui permettrait de planifier automatiquement des actions pour des personnages virtuels. Ceci rendrait les jeux vidéo encore plus réalistes et donnerait de meilleures expériences aux amateurs de jeux.

CONCLUSION

La conception d'un robot intelligent et autonome capable de réaliser des tâches complexes est un défi audacieux. Parmi les nombreuses capacités devant être mises à contribution pour relever un tel défi, un robot doit avant tout être capable de prendre ses propres décisions et de planifier lui-même comment il doit réaliser ses missions. La solution que nous avons présentée dans ce mémoire répond à ce besoin en permettant au robot de prendre ses propres décisions quant à comment coordonner et réaliser ses tâches.

Suite à l'analyse de diverses techniques de planification, allant de la planification classique jusqu'à la planification non déterministe et en passant par les graphes de planification, nous avons choisi de baser notre solution sur le concept de réseaux hiérarchiques de tâches (HTN). En y greffant des capacités pour la gestion du temps et des ressources, nous avons donné naissance au planificateur ConfPlan. Ce dernier a été développé dans un contexte bien précis, soit dans le contexte d'une participation au *AAAI Robot Challenge*. Le planificateur a donc été spécialement conçu pour répondre spécifiquement à des missions s'apparentant à celles qui sont proposées à ce défi. D'ailleurs, dans ce domaine particulier, nous avons montré que notre planificateur performait mieux que les planificateurs existants que nous avons pu évaluer.

Dans ce mémoire, nous avons également présenté l'intégration du planificateur ConfPlan dans une architecture décisionnelle robotique de type hybride. Une telle intégration n'est pas facile, étant donné que plusieurs défis importants propres au monde réel ont dû être surmontés. Nous avons entre autres proposé des solutions pour planifier et exécuter un plan de façon simultanée, tout en tenant compte de la progression des activités courantes

du robot.

Suite à cette intégration, nous avons pu réaliser diverses démonstrations et expériences. Parmi les démonstrations effectuées, nous avons entre autres participé au *AAAI Robot Challenge* de 2005. De plus, nous avons testé, avec succès, plusieurs scénarios mettant en valeur les capacités de notre planificateur. Parmi ceux présentés dans ce mémoire, nous avons la replanification suite à l'anticipation de violation de contraintes de temps, la détection de tâches mutuellement exclusives, la détection d'opportunisme et la prise en charge d'informations incomplètes quant à la position de certains lieux.

Avec les possibles améliorations présentées à la fin du présent mémoire, notre planificateur pourrait être utilisé au-delà de la robotique mobile. Il pourrait entre autres être intégré à des jeux vidéo pour rendre les personnages animés plus vivants. Nos techniques de planification pourraient aussi être mises au service de l'environnement et du développement durable en contribuant à réduire les émissions de gaz à effet de serre afin de prévenir les changements climatiques. Par exemple, notre planificateur pourrait être intégré dans des systèmes de génération d'échéanciers de production et de planification d'itinéraires pour la distribution et la livraison de colis. Grâce aux techniques de planification réactive, ConfPlan pourrait générer des plans pour gérer efficacement la consommation d'énergie d'une maison intelligente. En gérant mieux les ressources, ConfPlan pourrait alors aider à réduire la consommation d'énergie, ce qui permettrait de mieux concilier le développement économique et l'environnement.

ANNEXE A

Code PDDL 2.1 pour le domaine de la conférence simulée

A.1 PDDL 2.1 niveau 2 - avec numériques

```
(define (domain ScientificConference)
  (:requirements :typing :fluents)
  (:types location - object
          pres-session - object
          post-session - object
          )
  (:predicates
    (robot-at ?loc - location)
    (link ?x ?y - location)
    (rechargeable-place ?loc - location)
    (registration-place ?loc - location)
    (socializing-place ?loc - location)
    (presentation-done ?ses - pres-session ?loc - location)
    (assisted-to-pres  ?ses - pres-session ?loc - location)
    (picture-taken-poster ?ses - POST-SESSION ?loc - location)
    (poster-fixed ?ses - post-session ?loc - location)
    (poster-done ?ses - post-session ?loc - location)
    (registered)
    )
  (:functions
      (distance ?l1 ?l2 - location)
      (currenttime)
      (battery-level)
      (time-pres-begin ?ses - pres-session)
      (time-post-begin ?ses - post-session)
      (duration-pres)
      (min-poster-duration)
      (energyconsumed-per-meter)
      (energyconsumed-per-second)
```

```
    (max-speed)
    (max-battery-capacity)
  )
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; GOTO : Navigate between 2 Waypoints
;;
;; Goto consume time and energy proportionnal to distance
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:action GOTO
  :parameters(?from ?to - location)
  :precondition(and
        (robot-at ?from)
        (link ?from ?to)
        (>= (battery-level) (* (distance ?from ?to) (energyconsumed-per-meter)))
     )
  :effect(and
            (not (robot-at ?from))
            (robot-at ?to)
            (increase (currenttime) (/ (distance ?from ?to) (max-speed)))
            ;(increase (currenttime) 20)
            (decrease (battery-level) (* (distance ?from ?to) (energyconsumed-per-meter)))
   )
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; REGISTER : Register the robot at the conference
;;
;; Consume 5 miniutes and energy for 5 mins
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:action REGISTER
  :parameters(?loc - location)
  :precondition(and
      (robot-at ?loc)
      (registration-place ?loc)
      (not (registered))
      (>= (battery-level) (* 300 (energyconsumed-per-second)))
      )
  :effect(and
      (registered)
      (increase (currenttime) 300)
      (decrease (battery-level) (* 300 (energyconsumed-per-second)))
      )
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; RECHAGE : Recharge the battery of the robot
;;
;; Takes 1 hour and full charge de battery
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:action RECHARGE
  :parameters(?where - location)
  :precondition(and
      (robot-at ?where)
      (registered)
      (rechargeable-place ?where)
    )
  :effect(and
      (assign (battery-level) (max-battery-capacity))
      (increase (currenttime) 3600)
    )
```

```
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; MAKE-PRESENTATION : Robot make his technical presentation
;;
;; This action must begin exactly a time of presentation session
;; Takes 20mins and consume energy
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:action MAKE-PRESENTATION
  :parameters(?room - location ?ses - pres-session)
  :precondition(and
    (robot-at ?room)
    (registered)
    (= (currenttime) (time-pres-begin ?ses))
    (>= (battery-level) (* (duration-pres) (energyconsumed-per-second)))
    )
  :effect(and
    (increase (currenttime) (duration-pres))
    (presentation-done ?ses ?room)
    (decrease (battery-level) (* (duration-pres) (energyconsumed-per-second)))
    )
 )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; WAIT-PRES-SESSION : Standby the robot
;;
;; This action is necessary for time synchronization.
;; Wait until a presentation session
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 (:action WAIT-PRES-SESSION
   :parameters(?ses - pres-session)
   :precondition(and
       (< (currenttime) (time-pres-begin ?ses))
       (>= (battery-level) (* (- (time-pres-begin ?ses) (currenttime)) (energyconsumed-per-second)))
       )
   :effect(and
       (increase (currenttime) (- (time-pres-begin ?ses) (currenttime)))
       (decrease (battery-level) (* (- (time-pres-begin ?ses) (currenttime))
                                    (energyconsumed-per-second)))
       )
 )
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; WAIT-POST-SESSION : Standby the robot
;;
;; This action is necessary for time synchronization.
;; Wait until a poster session
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 (:action WAIT-POST-SESSION
   :parameters(?ses - post-session)
   :precondition(and
       (< (currenttime) (time-post-begin ?ses))
       (>= (battery-level) (* (- (time-post-begin ?ses) (currenttime))
                              (energyconsumed-per-second)))
       )
   :effect(and
       (increase (currenttime) (- (time-post-begin ?ses) (currenttime)))
       (decrease (battery-level) (* (- (time-post-begin ?ses) (currenttime))
                                    (energyconsumed-per-second)))
       )
 )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; ASSIST-PRESENTATION : Assit to a presentation
;;
;; This action is similar to MAKE-PRESENTATION
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 (:action ASSIST-PRESENTATION
   :parameters(?room - location ?ses - pres-session)
   :precondition(and
     (robot-at ?room)
     (registered)
     (= (currenttime) (time-pres-begin ?ses))
     (>= (battery-level) (* (duration-pres) (energyconsumed-per-second)))
     )
   :effect(and
     (increase (currenttime) (duration-pres))
     (assisted-to-pres ?ses ?room)
     (decrease (battery-level) (* (duration-pres) (energyconsumed-per-second)))
     )
 )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; TAKE-PICTURE-POSTER : Takes a picture of a poster
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 (:action TAKE-PICTURE-POSTER
   :parameters(?loc - location ?ses - post-session)
   :precondition(and
       (robot-at ?loc)
       (registered)
       (not (picture-taken-poster ?ses ?loc))
       (>= (currenttime) (time-post-begin ?ses))
       (<= (currenttime) (+ (min-poster-duration) (time-post-begin ?ses))))
   :effect(and
      (picture-taken-poster ?ses ?loc)
      (increase (currenttime) 300))
 )
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; FIX-POSTER : Fix a poster
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 (:action FIX-POSTER
   :parameters(?loc - location ?ses - post-session)
   :precondition(and
       (robot-at ?loc)
       (registered)
       (not (poster-fixed ?ses ?loc))
         ; we can fix poster 1h before session
       (>= (currenttime) (- (time-post-begin ?ses) 3600))
         ; we permit to be 2 min late...
       (<= (currenttime) (+ (time-post-begin ?ses) 120))
       (>= (battery-level) (* 120 (energyconsumed-per-second)))
       )
   :effect(and
       (poster-fixed ?ses ?loc)
       (increase (currenttime) 120)
       (decrease (battery-level) (* 120 (energyconsumed-per-second)))
       )
  )
```

```
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;; REMOVE-POSTER : Remove the poster that we previously fix
  ;;
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
   (:action REMOVE-POSTER
      :parameters(?loc - location ?ses - post-session)
      :precondition(and
         (robot-at ?loc)
         (poster-fixed ?ses ?loc)
            ; poster has to be fixed for a minimum duration
         (>= (currenttime) (+ (time-post-begin ?ses) (min-poster-duration)))
            ; poster can be fixed 1h more that min duration
         (<= (currenttime) (+ (time-post-begin ?ses) (+ (min-poster-duration) 3600)))
         (>= (battery-level) (* 120 (energyconsumed-per-second)))
         )
      :effect(and
         (not (poster-fixed ?ses ?loc))
         (poster-done ?ses ?loc)
         (increase (currenttime) 120)
         (decrease (battery-level) (* 120 (energyconsumed-per-second)))
         )
   )
)
```

A.2 PDDL 2.1 niveau 5 - valeurs numériques et temporelles

```
(define (domain ScientificConference)
  (:requirements :strips :equality :typing :fluents :durative-actions )
  (:types location - object
          session - object
          message - object
     )
  (:predicates
    (robot-at ?loc - location)
    (link ?x - location ?y - location)
    (registration-place ?loc - location)
    (rechargeable-place ?loc - location)
    (presentation-done ?pses - session ?loc - location)
    (presentation-assisted ?pres - session ?loc - location)
    (picture-taken-poster ?ses - session ?loc - location)
    (poster-fixed ?ses - session ?loc - location)
    (poster-done ?ses - session ?loc - location)
    (sessionenabled ?pses - session)
    (fix-enabled ?ses - session)
    (remove-enabled ?ses - session)
    (registered)
    (message-here ?msg - message  ?loc - location)
    (robot-have-msg ?m - message)
    )
  (:functions
      (distance ?l1 - location ?l2 - location)
      (battery-level)
```

```
    (energyconsumed-per-meter)
    (energyconsumed-per-second)
    (speed)
    (duration-pres)
    (max-battery-capacity)
  )
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; GOTO : Navigate between 2 Waypoints
;;
;; Goto consume time and energy proportionnal to distance
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:durative-action GOTO
  :parameters(?from - location ?to - location)
  :duration (= ?duration (/ (distance ?from ?to) (speed)))
  ;;:duration (= ?duration (distance ?from ?to) )
  :condition(and
        (at start (robot-at ?from))
        (over all (link ?from ?to))
        (over all (>= (battery-level) (* (distance ?from ?to) (energyconsumed-per-meter))))
     )
  :effect(and
            (at end (not (robot-at ?from)))
            (at end (robot-at ?to))
            (at end (decrease (battery-level) (* (distance ?from ?to) (energyconsumed-per-meter))))
   )
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; REGISTER : Register the robot at the conference
;;
;; Consume 5 miniutes and energy for 5 mins
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:durative-action REGISTER
  :parameters(?loc - location)
  :duration (= ?duration 300)
  :condition(and
      (over all (robot-at ?loc))
      (over all (registration-place ?loc))
      ;;(over all (not (registered)))
      (over all (>= (battery-level) (* 300 (energyconsumed-per-second))))
      )
  :effect(and
      (at end (registered))
      (at end (decrease (battery-level) (* 300 (energyconsumed-per-second))))
      )
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; RECHAGE : Recharge the battery of the robot
;;
;; Takes 1 hour and full charge de battery
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:durative-action RECHARGE
  :parameters(?where - location)
  :duration (= ?duration 3600)
  :condition(and
      (over all (robot-at ?where))
      (over all (registered))
      (over all (rechargeable-place ?where))
    )
  :effect(and
```

```
      (at end (assign (battery-level) (max-battery-capacity)))
    )
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; MAKE-PRESENTATION : Robot make his technical presentation
;;
;; This action must begin exactly a time of presentation session
;; Takes 20mins and consume energy
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:durative-action MAKE-PRESENTATION
  :parameters(?ses - session ?room - location)
  :duration (= ?duration (duration-pres))
  :condition(and
    (over all (robot-at ?room))
    (over all (sessionenabled ?ses))
    (over all (registered))
    (over all (>= (battery-level) (* (duration-pres) (energyconsumed-per-second))))
    )
  :effect(and
    (at end (presentation-done ?ses ?room))
    (at end (decrease (battery-level) (* (duration-pres) (energyconsumed-per-second))))
    )
 )

(:durative-action ASSIST-PRESENTATION
  :parameters(?ses - session ?room - location)
  :duration (= ?duration (duration-pres))
  :condition(and
    (over all (robot-at ?room))
    (over all (sessionenabled ?ses))
    (over all (registered))
    (over all (>= (battery-level) (* (duration-pres) (energyconsumed-per-second))))
    )
  :effect(and
    (at end (presentation-assisted ?ses ?room))
    (at end (decrease (battery-level) (* (duration-pres) (energyconsumed-per-second))))
    )
 )

(:durative-action TAKE-PICTURE-POSTER
   :parameters(?ses - session ?loc - location)
   :duration (= ?duration 120)
   :condition(and
       (over all (robot-at ?loc))
       (over all (registered))
       (over all (sessionenabled ?ses))
       ;;(at start (not (picture-taken-poster ?ses ?loc)))
       (over all (>= (battery-level) (* 120 (energyconsumed-per-second))))
       )
   :effect(and
      (at end (picture-taken-poster ?ses ?loc))
      (at end (decrease (battery-level) (* 120 (energyconsumed-per-second))))
      )
 )

 (:durative-action FIX-POSTER
   :parameters(?ses - session ?loc - location)
   :duration (= ?duration 120)
   :condition(and
```

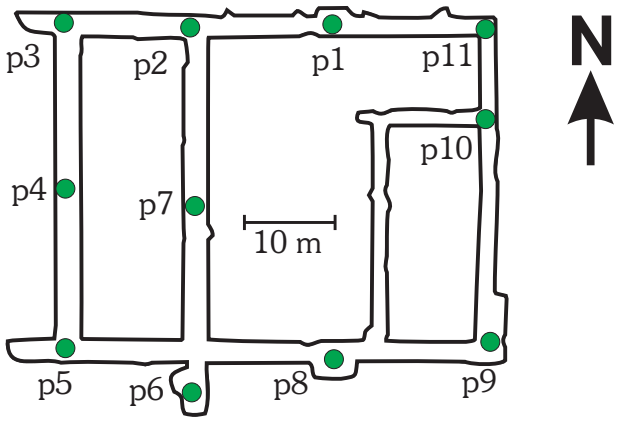
```
        (over all (robot-at ?loc))
        (over all (registered))
        (over all (fix-enabled ?ses))
        ;;(not (poster-fixed ?ses ?loc))
        (over all (>= (battery-level) (* 120 (energyconsumed-per-second))))
        )
     :effect(and
        (at end (poster-fixed ?ses ?loc))
        (at end (decrease (battery-level) (* 120 (energyconsumed-per-second))))
        )
    )
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;; REMOVE-POSTER : Remove the poster that we previously fix
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
   (:durative-action REMOVE-POSTER
      :parameters(?ses - session ?loc - location)
      :duration (= ?duration 120)
      :condition(and
        (over all (robot-at ?loc))
        (over all (remove-enabled ?ses))
        (at start (poster-fixed ?ses ?loc))
        (over all (>= (battery-level) (* 120 (energyconsumed-per-second))))
        )
      :effect(and
        (at end (not (poster-fixed ?ses ?loc)))
        (at end (poster-done ?ses ?loc))
        (at end (decrease (battery-level) (* 120 (energyconsumed-per-second))))
        )
   )

  (:durative-action ASKMESSAGE
    :parameters(?msg - message ?loc - location)
    :duration (= ?duration 90)
    :condition (and
       (over all (robot-at ?loc))
       (over all (message-here ?msg ?loc))
      )
    :effect(at end (robot-have-msg ?msg))
   )

  (:durative-action GIVEMESSAGE
    :parameters(?msg - message ?loc - location)
    :duration (= ?duration 90)
    :condition (and
       (over all (robot-at ?loc))
       (over all (robot-have-msg ?msg))
      )
    :effect(at end (message-here ?msg ?loc))
   )
)
```

ANNEXE B

Expériences au 5^e étage



	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11
p1	0,0	16,3	28,3	44,3	63,3	53,3	31,3	62,1	50,7	16,2	26,2
p2	16,3	0,0	14,0	28,0	47,0	37,0	15,0	49,0	65,4	31,5	40,9
p3	28,3	14,0	0,0	16,0	35,0	51,0	29,0	63,0	79,0	44,5	54,5
p4	44,3	28,0	16,0	0,0	19,0	35,0	42,4	47,0	63,4	59,5	68,9
p5	63,3	47,0	35,0	19,0	0,0	16,0	32,0	28,0	46,0	78,5	69,5
p6	53,3	37,0	51,0	35,0	16,0	0,0	22,0	18,0	36,0	68,5	59,5
p7	31,3	15,0	29,0	42,4	32,0	22,0	0,0	34,0	50,4	46,5	55,9
p8	62,1	49,0	63,0	47,0	28,0	18,0	34,0	0,0	18,0	50,5	41,5
p9	50,7	65,4	79,0	63,4	46,0	36,0	50,4	18,0	0,0	34,5	24,5
p10	16,2	31,5	44,5	59,5	78,5	68,5	46,5	50,5	34,5	0,0	10,0
p11	26,2	40,9	54,5	68,9	69,5	59,5	55,9	41,5	24,5	10,0	0,0

Figure B.1 – Carte du 5^e étage avec table de distances

Bibliographie

[1] A. Albore, E. Beaudry, P. Bertoli et F. Kabanza : Using a contingency planner within a robot architecture. *à paraître dans Proceedings of Workshop on Planning, Learning and Monitoring with Uncertainty and Dynamic Worlds, in conjunction with the 17th European Conference on Artificial Intelligence*, 2006.

[2] A. Albore et P. Bertoli : Generating safe assumption-based plans for partially observable, nondeterministic domains. Dans *National Conference on Artificial Intelligence (AAAI)*, pages 495–500. 2004.

[3] R.C. Arkin : *A Behavior-based Robotics*. MIT Press, Cambridge, MA, USA, 1998.

[4] F. Bacchus et M. Ady : Planning with resources and concurrency : A forward chaining approach. Dans *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 417–424, 2001.

[5] F. Bacchus et F. Kabanza : Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

[6] E. Beaudry, Y. Brosseau, C. Côté, C. Raïevsky, D. Létourneau, F. Kabanza et F. Michaud : Reactive planning in a motivated behavioral architecture. Dans *National Conference on Artificial Intelligence (AAAI)*, pages 1242–1249, 2005.

[7] E. Beaudry, F. Kabanza et F. Michaud : Planning for a mobile robot to attend a conference. Dans B. Kégl et G. Lapalme, éditeurs : *Canadian Conference on AI*, vol. 3501 de *Lecture Notes in Computer Science*, pages 48–52. Springer, 2005.

[8] M. Beetz, W. Burgard, D. Fox et A. Cremers : Integrating active localization into high-level robot control systems. *Robotics and Autonomous Systems*, 23:205–220, 1998.

[9] A. Blum et M. Furst : Fast planning through planning graph analysis. Dans *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1636–1642, 1995.

[10] R.A. Brooks : A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[11] C. Côté, D. Létourneau, F. Michaud et Y. Brosseau : Software design patterns for robotics : Solving integration problems with MARIE. Dans *Workshop of Robotic Software Environment, IEEE International Conference on Robotics and Automation*, 2005.

[12] E.W. Dijkstra : A note on two problems in connexion with graphs. Dans *Numerische Mathematik*, vol. 1, pages 269–271, 1959.

[13] M.B. Do et S. Kambhampati : SAPA : A scalable multi-objective metric temporal planner. *Journal of Artificial Intelligence Research (JAIR)*, 20:155–194, 2003.

[14] S. Edelkamp et J. Hoffman : PDDL 2.2 : The languages for the classical part of the 4th international planning competition. Dans Internet : http ://ipc.icaps-conference.org, 2004.

[15] R. Fikes et N.J. Nilsson : STRIPS : A new approach to the application of theorem proving to problem solving. Dans *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 608–620, 1971.

[16] R.W. Floyd : Algorithm 97 : Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[17] D. Fox, W. Burgard et S. Thrun : Active Markov localization for mobile robots. *Robotics and Autonomous Systems*, 25(3-4):195–207, 1998.

[18] D. Fox, W. Burgard et S. Thrun : Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research (JAIR)*, 11:391–427, 1999.

[19] M. Fox et D. Long : PDDL 2.1 : An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, 20:61–124, 2003.

[20] A. Gerevini et I. Serina : LPG : A planner based on local search for planning graphs with action costs. Dans *International Conference on Artificial Intelligence Planning Systems*, pages 13–22, 2002.

[21] R.C. Gonzalez et R.E. Woods : *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[22] K.Z. Haigh et M. Veloso : Interleaving planning and robot execution for asynchronous user requests. *Autonomous Robots*, 5(1):79–95, March 1998.

[23] P.E. Hart, N.J. Nilsson et B. Raphael : A formal basis for the heuristic determination of minimum cost paths. Dans *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pages 100–107, 1968.

[24] J. Hoffmann : The Metric-FF planning system : Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research (JAIR)*, 20:291–341, 2003.

[25] J. Hoffmann et B. Nebel : The FF planning system : Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)*, 14:253–302, 2001.

[26] S. Koenig, R Goodwin et R Simmons : Robot navigation with Markov models : A framework for path planning and learning with limited computational resources. Dans *International Workshop on Reasoning with Uncertainty in Robotics*, December 1995.

[27] P. Laborie et M. Ghallab : IxTeT : an integrated approach for plan generation and scheduling. Dans *IEEE-INRIA Symposium on Emerging Technologies and Factory Automation (ETFA 95)*, pages 485–495, Parie, France, Octobre 1995.

[28] M. Likhachev, D.I. Ferguson, G.J. Gordon, A. Stentz et S. Thrun : Anytime dynamic A* : An anytime, replanning algorithm. Dans S. Biundo, K.L. Myers et K. Rajan, éditeurs : *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 262–271. AAAI, 2005.

[29] M. Likhachev, G. Gordon et S. Thrun : ARA* : Anytime A* with provable bounds on sub-optimality. Dans *Advances in Neural Information Processing System 16*, pages 767–774, Cambridge, MA, 2003. MIT Press.

[30] D. Létourneau, F. Michaud et J.-M. Valin : Autonomous robot that can read. Dans *EURASIP Journal on Applied Signal Processing, Special Issue on Advances in Intelligent Vision Systems : Methods and Applications*, vol. 3, pages 340–361, 2004.

[31] B.A. Maxwell, W. Smart, A. Jacoff, J. Casper, B. Weiss, J. Scholtz, H. Yanco, M. Micire, A. Stroupe, D. Stormont et T. Lauwers : 2003 AAAI robot competition and exhibition. *AI Magazine*, 25(2):68–80, Summer 2004.

[32] F. Michaud : EMIB - Computational architecture based on emotion and motivation for intentional selection and configuration of behaviour-producing modules. *Cognitive Science Quarterly*, 3(4):340–361, 2002.

[33] F. Michaud, J. Audet, D. Létourneau, L. Lussier, C. Théberge-Turmel et S. Caron : Experiences with an autonomous robot attending the AAAI conference. *IEEE Intelligent Systems*, 16(5):23–29, 2001.

[34] R.R. Murphy : *Introduction to AI Robotics*. MIT Press, Cambridge, MA, USA, 2000.

[35] D. Nau, Y. Cao, A. Lotem et H. Muñoz-Avila : SHOP : Simple hierarchical ordered planner. Dans *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 968–973, 1999.

[36] D. Nau, M. Ghallab et P. Traverso : *Automated Planning : Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[37] D.S. Nau, T.C. Au, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu et F. Yaman : SHOP2 : An HTN planning system. *Journal of Artificial Intelligence Research (JAIR)*, 20:379–404, 2003.

[38] D. Panigrahi, C. Chiasserini, S. Dey, R. Rao, A. Raghunathan et K. Lahiri : Battery life estimation of mobile embedded systems. Dans *International Conference on VLSI Design (VLSID 2001)*, 2001.

[39] R.G. Simmons, D. Goldberg, A. Goode, M. Montemerlo, N. Roy, B. Sellner, C. Urmson, A.C. Schultz, M. Abramson, W. Adams, A. Atrash, M.D. Bugajska, M. Coblenz, M. MacMahon, D. Perzanowski, I. Horswill, R. Zubek, D. Kortenkamp, B. Wolfe, T. Milam et B.A. Maxwell : GRACE : An autonomous robot for the AAAI robot challenge. *AI Magazine*, 24(2):51–72, 2003.

[40] S. Thrun, D. Fox et W. Burgard : Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2001.

[41] E. Trucco et A. Verri : *Introductory Techniques for 3-D Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[42] F. Yaman et D.S. Nau : TimeLine : An HTN planner that can reason about time. Dans M. Fox et A.M. Coddington, éditeurs : *AIPS Workshop on Planning for Temporal Domains*, pages 75–81, 2002.

[43] S. Zilberstein et S. Russell : Approximate reasoning using anytime algorithms. Dans *Imprecise and Approximate Computation*, pages 43–62. Kluwer Academic Publishers, 1995.